PROFESSIONAL SERVICES

Migrating from Snowflake to BigQuery

Data and analytics

Date: February 2021

Google Cloud

Contents

About this document

This document provides guidance on how to migrate your database from a Snowflake data warehouse to BigQuery. It describes a migration process for moving your database to BigQuery, as well as best practices for ensuring a smooth transition with minimal service impact.

Document highlights	
Purpose	To cover important concepts and topics that help you migrate your databases from Snowflake to BigQuery.
Intended audience	Enterprise architects, DBAs, application developers, and IT security.
Key assumptions	That you are familiar with Snowflake and want guidance on transitioning to BigQuery.
Prerequisites	Migrating data warehouses to BigQuery, which this document builds on.
Delivery note	Use this document to guide discussions and answer questions related to moving your Snowflake data warehouse to BigQuery.

1. Introduction

This document provides a technical background on data migration from Snowflake to BigQuery as a part of the enterprise data warehouse (EDW) migration series. It covers the foundational differences between Snowflake and BigQuery and provides guidance on a successful migration. This document also covers the EDW and online analytical processing (OLAP) use cases. For online transaction processing (OLTP) use cases, explore databases that are suited for high input/output operations per second (IOPS), such as Cloud SQL or Spanner.

1.1 Terminology map

This document uses Snowflake and BigQuery terminology to describe the functionality that each product provides. The following table maps Snowflake terms to equivalent BigQuery terms:

Snowflake	BigQuery
Database	Dataset
Table	Table
Schema	Schema
Session-specific temporary or transient table	Anonymous or temporary table
View	View
Secure views	Authorized views
Virtual warehouse	Reservations
Snowflake columnar storage format	Capacitor
Materialized view	Materialized view
No equivalent for partitioning (because *micro-partitioning* is used)	Partitioning
Clustering	Clustering
Security-enhanced user-defined functions (UDFs)	Authorized UDFs

1.2 Architectural comparison

Although Snowflake and BigQuery are both analytic data warehouses, they have some key architectural differences.

Snowflake's architecture is a hybrid of traditional shared-disk database architectures and shared-nothing database architectures. Similar to a shared-disk architecture, the compute nodes in Snowflake have access to a central data repository for persisted data. Similar to a shared-nothing architecture, the queries use massively parallel processing (MPP) compute clusters, where each node has local access to a portion of the whole dataset. For more information, see Snowflake's architecture.

As Figure 1 illustrates, BigQuery's architecture is vastly different from traditional node-based cloud data warehouse solutions or MPP systems. It decouples storage and compute, allowing them to scale independently on demand. Storage is done on Colossus, Google's global storage system, which uses the columnar storage format and a compression algorithm. Compute uses Dremel, a large multi-tenant cluster that executes SQL queries by dynamically allocating slots to queries on an as-needed basis. This structure provides flexibility and can improve cost control because the compute resources don't need to be running all the time.

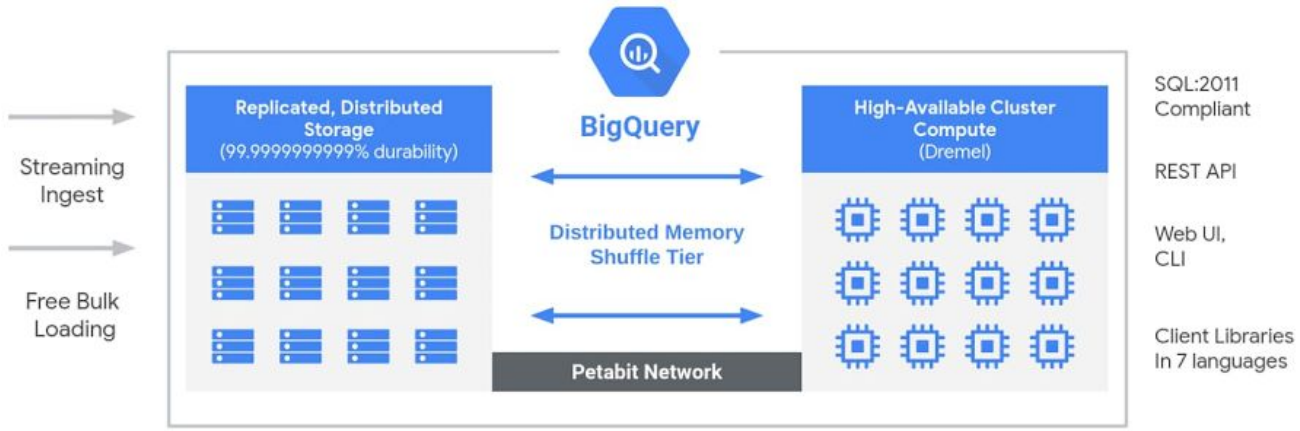

Figure 1: BigQuery architecture.

2. Pre-migration

A data migration involves several pre-migration steps and considerations. The earlier you begin pre-migration tasks, the easier it is to evaluate how Google Cloud features can suit your needs.

The following table outlines conceptual differences between Snowflake and BigQuery that play into the pre-migration tasks:

Conceptual comparison	
Snowflake tools and concepts	**Google Cloud equivalent**
Snowflake data warehouse	BigQuery For more details, see Service model comparison and Costs.
Snowflake web interface • Manage databases • Manage warehouses • Manage queries/worksheets • View historical queries	BigQuery web user interface bq command-line tool For more details, see Managing data.
Security-related features • Network and site access • Account and user authentication • Object security • Data security • Security validations	IAM For more details, see Granting permissions.

2.1 Capacity planning

Behind the scenes, BigQuery uses slots to measure the computational capacity that is needed to execute a SQL query. There are two pricing models for data processing that use slots: the on-demand pricing model and the flat-rate pricing model.

With the on-demand pricing model, each project is subject to a slot quota that has transient burst capability. With the flat-rate pricing model, you choose how many slots to reserve. Your queries will use the allocated slots, and you pay for that slot capacity until the commitment is deleted. For more information, see pricing for BigQuery.

To help with slot estimation, we recommend exploring the option of using the flex slots, which are 60-second commitments that help you estimate how your workload performs with flat-rate billing. After you determine the necessary slots for optimal utilization, you can shut down your flex commitments and make a longer-term commitment.

Another method of capacity planning is to use BigQuery monitoring using Cloud Monitoring and to analyze your audit logs using BigQuery. Many companies use Google Data Studio (here's an open source example of a Data Studio dashboard), Looker, or Tableau as a frontend to visualize BigQuery's audit log data, specifically for slot usage across queries and projects. You could also leverage BigQuery's system tables data for monitoring slot utilization across jobs and reservations (here's an open source example of a Data Studio dashboard). Regularly

monitoring and analyzing your slot utilization helps you estimate how many total slots your organization needs as you grow on Google Cloud.

If you want to purchase slots yourself through flex, monthly, or yearly commitments, you can get started with BigQuery Reservations by using the Cloud Console or the `bq` command-line tool. If you have any questions regarding your current plan or options, contact your sales representative.

2.2 Data security

Many companies prioritize security and privacy to protect their own and their customers' data. When migrating from Snowflake to BigQuery, you must consider the way that Google Cloud in general, and BigQuery in particular, handles security differently from Snowflake. This section discusses how to configure your security-related options in Google Cloud.

One consideration is that BigQuery does not have different tiers or editions like Snowflake has. For example, in Snowflake, column-level security is supported in the Enterprise edition, and customer-managed encryption keys are supported in the Business Critical edition. In BigQuery, all features and enhanced security measures mentioned in the next section are offered at the same price.

The parallel to Snowflake's access control privileges are Identity and Access Management (IAM) roles in Google Cloud. IAM roles let you control which users and system accounts can access your BigQuery database. You can grant access to accounts at the project, resource, or dataset level. For more information, read an overview of how IAM roles work.

Snowflake supports system-defined roles and custom roles. System-defined roles consist of system and security-related privileges, whereas custom roles can be created by using the `SECURITYADMIN` roles or by using any role that has the `CREATE ROLE` privilege. Each custom role in Snowflake is composed of privileges. The equivalent of privileges in Snowflake is permissions in IAM. In IAM, permissions are grouped into roles.

IAM provides three types of roles:

- **Basic roles.** These roles include the Owner, Editor, and Viewer roles. You can apply these roles at the project or service resource levels by using the Cloud Console, the IAM API, or the `gcloud` command-line tool. In general, for the strongest security, we recommend that you use BigQuery-specific roles to follow the principle of least privilege.

- **Predefined roles.** These roles provide more granular access to features in a product (such as BigQuery) and are meant to support common use cases and access control patterns.
- **Custom roles.** These roles are composed of user-specified permissions.

You can also create authorized views to limit data access so that specified users can query a view without having read access to the underlying tables. For more fine-grained access control, you can create views in your dataset that are filtered based on user roles.

Snowflake lets you grant roles to other roles, creating a hierarchy of roles. IAM does not support a role hierarchy but implements a resource hierarchy. The IAM hierarchy includes the organization level, folder level, project level, and resource level. You can set IAM roles at any level of the hierarchy, and resources inherit all the policies of their parent resources.

Within IAM, BigQuery also provides table-level access control. Table-level permissions determine the users, groups, and service accounts that can access a table or view. You can give a user access to specific tables or views without giving the user access to the complete dataset. For more granular access, you can also use column-level access control. This type of control provides fine-grained access to sensitive columns by using policy tags or type-based classification of data.

Snowflake provides end-to-end encryption in which it automatically encrypts all stored data. Google Cloud provides the same feature by encrypting all data at rest and in transit by default.

Similar to Snowflake Enterprise edition, BigQuery supports customer-managed encryption keys (CMEK) for users who want to control and manage key encryption keys (KEKs) in Cloud Key Management Service. For more details about encryption in Google Cloud, see Encryption at rest in Google Cloud and Encryption in transit in Google Cloud.

2.3 Migration considerations

There are a few Snowflake features that you cannot port directly to BigQuery. You can replicate some of these features by using other Google Cloud offerings with BigQuery.

Additionally, BigQuery limits the maximum rate of incoming requests and enforces appropriate quotas on each project. Specific policies vary depending on resource availability, user profile, service usage history, and other factors, and the policies are subject to change without notice. For more details on the current rate limits and quota limits for BigQuery, see Quotas and limits.

2.3.1 Fully supported scenarios

With fully supported scenarios, you can migrate your data directly from Snowflake without rearchitecting applications.

Fully supported scenarios include the following:

- Simple tables.
- External data sources (Cloud Storage, BigTable, Drive, and Cloud SQL). For more information, see External data sources.
- Public dataset integration.
- Encrypted traffic.
- Applications requiring more than 100 concurrent on-demand interactive queries with Standard SQL. If your application requires more than 100 concurrent on-demand interactive standard SQL queries, work with your sales representative to adjust your quotas accordingly. These quotas apply to each project.

2.3.2 Partially supported scenarios

With partially supported scenarios, you might need to redesign your application architecture to fit with BigQuery's unique architecture. You might also need to integrate your application with other Google Cloud offerings.

The only partially supported scenario is time travel. In BigQuery, you can use time travel to access data from any point within the last seven days. If you need to go beyond seven days, consider exporting regularly scheduled snapshots.

2.3.3 Unsupported scenarios

BigQuery doesn't offer built-in support for the following scenarios. In these scenarios, you might need to use other services in Google Cloud.

- **Streams.** A stream object records data manipulation language (DML) changes made to tables and also metadata about each change so that actions can be taken using the changed data. BigQuery does not have built-in support for change data capture (CDC), but you can use CDC software such as Debezium to write the records to BigQuery by using Dataflow. For more details on manually designing your own CDC pipeline with BigQuery, see the EDW to BigQuery migration guide.
- **Tasks.** You can combine tasks with table streams for continuous extract, load, and transfer (ELT) workflows in order to process recently changed table rows. Currently, BigQuery supports scheduled queries but does not support streams or stream

integration into queries. As an alternative, you can set up a scheduled query to trigger a Cloud Function by using Pub/Sub.

- **External functions.** An external function calls code that runs outside Snowflake. BigQuery itself does not support external function calls, but you can integrate Cloud Functions with BigQuery by using the Cloud Functions API.

3. Migrating your data

This section describes how to configure and initiate your migration from Snowflake to BigQuery based on the framework outlined in the EDW migration guide.

3.1 Migration strategy

When migrating a complex data warehousing operation to the cloud, it's best to take an iterative approach as described in the EDW migration framework. This framework presents an organized structure for migrating data for each workload. Figure 2 illustrates the phases that are involved. The following sections summarize the migration process for Snowflake to BigQuery.

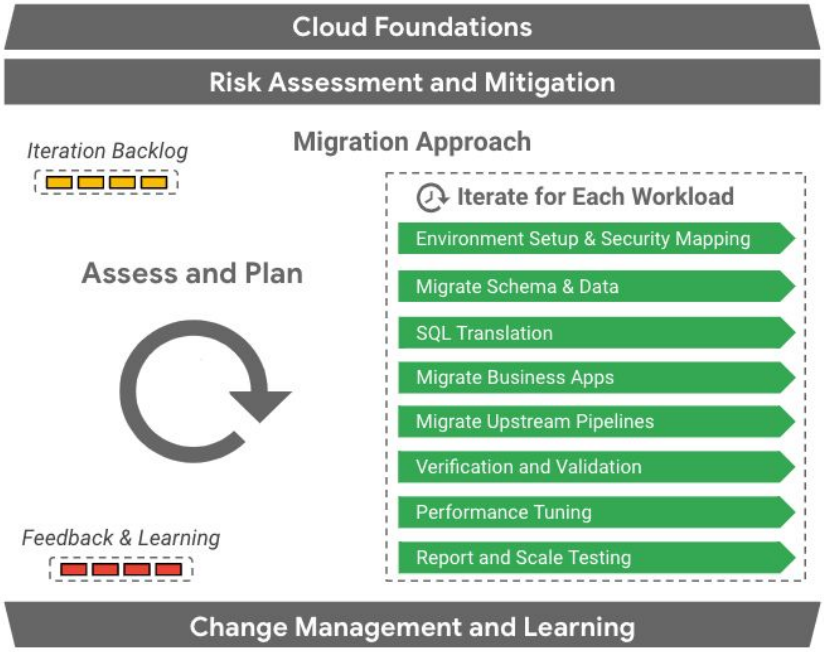

Figure 2: EDW migration framework.

3.1.1 Migration architecture

When you begin the migration, you run both Snowflake and BigQuery. Figure 3 presents an architecture that minimally impacts existing operations. By transferring clean,

quality-controlled data, you can reuse existing tools and processes while offloading workloads to BigQuery. You can also validate reports and dashboards against old versions. Nevertheless, because OLAP data is maintained in redundant places, this operation isn't cost effective. It also extends the processing time.

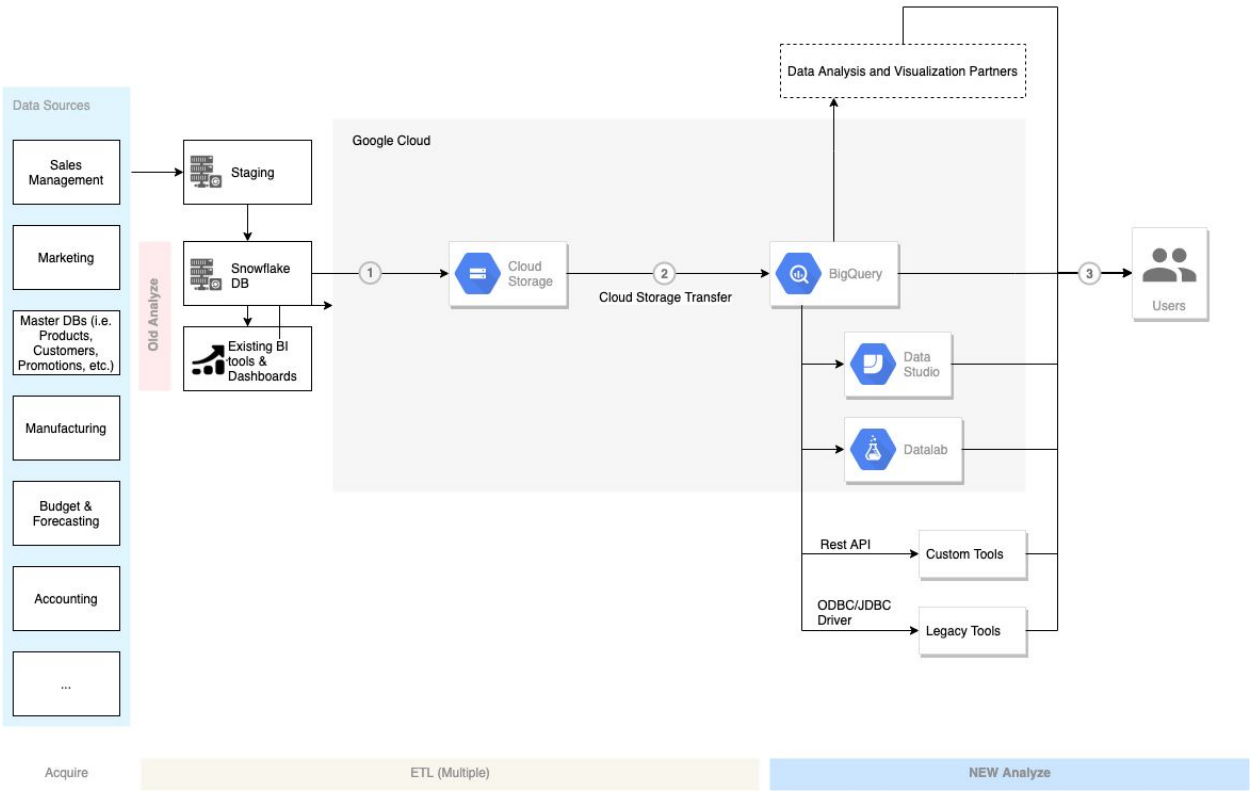

Figure 3: An ongoing migration of Snowflake to BigQuery.

3.1.2 Final state

Figure 4 illustrates the architecture for the end state of your data warehouse migration to Google Cloud. In this stage, all the data from source systems is directly persisted in Google Cloud. Depending on the number and complexity of the source systems, delivering this architecture can be further staged by tackling source systems one at a time according to priority, interdependencies, integration risks, or other business factors.

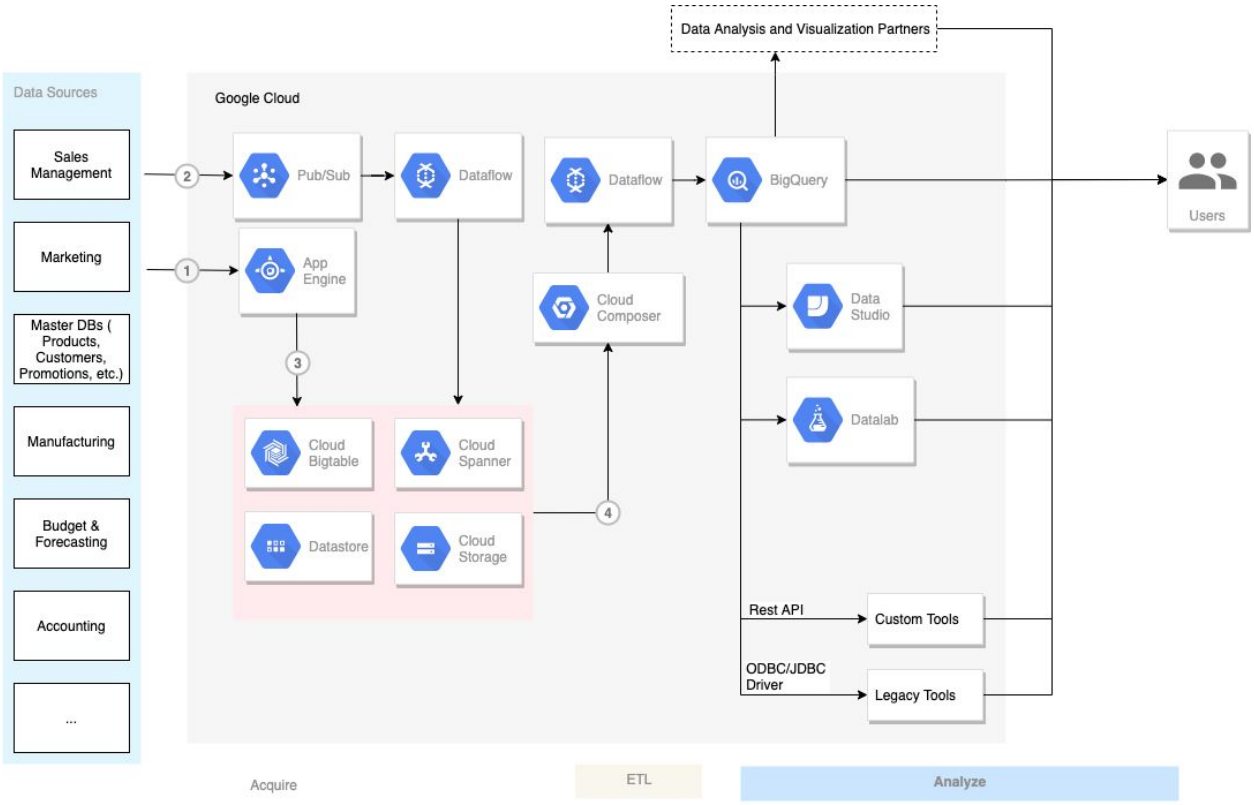

Figure 4: Snowflake to BigQuery post-migration.

The architecture in Figure 4 assumes the migration of modernized data pipelines and ingestion to Google Cloud. Step 1 of the figure shows synchronous integration between data sources and App Engine when dealing with use cases that require explicit user actions as a part of the flow. The figure also shows asynchronous integration (Step 2) using Pub/Sub for large volumes of concurrent event data. Step 3 persists data using one or more Google Cloud products, depending on the nature of the data. Finally, Step 4 displays ETL to BigQuery.

3.2 Preparing your Cloud Storage environment

You can use a Cloud Storage bucket for both staging your data for initial loading and for querying as an external data source. As part of your migration, if the BigQuery dataset location is set to a value other than the United States, you must provision a regional or multi-regional Cloud Storage bucket in the same region as your BigQuery instance.

When you set up your BigQuery dataset, take into account the following location considerations:

- Colocate your BigQuery dataset and your external data source.
- Colocate your Cloud Storage buckets for loading data.
- Colocate your Cloud Storage buckets for exporting data.

- Develop a data management and retention plan.

For more information, see Location considerations.

When you choose the location of your data, make sure that your BigQuery dataset and your external data source or Cloud Storage bucket are in the same region. For more details on geographical location considerations for loading data from Cloud Storage, see Introduction to loading data from Cloud Storage.

3.3 Building your schema

When you import your data in bulk from a file like a JSON, an Avro, or a CSV file, BigQuery auto-detects the schema, so you do not need to predefine it. To get a detailed overview of the schema migration process for EDW workloads, see Schema and data transfer overview.

3.3.1 Schema changes

If you are planning schema changes in your migration to BigQuery, we recommend that you first migrate your schema as-is. BigQuery supports a wide range of data model design patterns such as star schema or snowflake schema. Because of this support, you don't need to update your upstream data pipelines for a new schema, and you can use automated migration tools to transfer your data and schema.

After the schema migration, you can test performance and make optimizations based on the results. For example, you can introduce partitioning to make your data more efficient to manage and query. Partitioning in BigQuery refers to a special table that is divided into segments called partitions. Partitioning is different from Snowflake's micro-partitioning, which happens automatically as data is loaded. BigQuery's partitioning lets you improve query performance and cost control by partitioning by ingestion time, timestamp, or integer range. For more information, see Introduction to partitioned tables.

3.3.2 Clustered tables

Clustered tables is another schema optimization that you can implement. BigQuery, like Snowflake, lets you cluster tables, enabling you to automatically organize table data based on the contents of one or more columns in the table's schema. BigQuery uses the columns that you specify to colocate related data. Clustering can improve the performance of certain types of queries, such as queries that use filter clauses or queries that aggregate data. For more information on how clustered tables work in BigQuery, see Introduction to clustered tables.

3.3.3 Updating a schema

After your data is in BigQuery, you can always make updates to the schema, such as adding columns to the schema definition or relaxing a column's mode from `REQUIRED` to `NULLABLE`.

Something to remember is that BigQuery uses case-sensitive naming conventions for the table name, whereas Snowflake uses case-insensitive naming patterns. This convention means that you might need to revisit any inconsistencies in the table-naming conventions that might exist in Snowflake, and rectify any inconsistencies that arose during the move to BigQuery. For more information on schema modification, see Modifying table schemas.

Some schema modifications are not directly supported in BigQuery and require manual workarounds, including the following:

- Changing a column's name.
- Changing a column's data type.
- Changing a column's mode (aside from relaxing `REQUIRED` columns to `NULLABLE`).
- Deleting a column.

For specific instructions on how to manually implement these schema changes, see Manually changing table schemas.

3.4 Supported data types, properties, and file formats

Snowflake and BigQuery support most of the same data types, though they sometimes use different names. For a complete list of supported data types in Snowflake and BigQuery, see the "Data types" section of the SQL translation reference). For more information about BigQuery's supported data types, see Standard SQL data types.

Snowflake can export data to BigQuery in the following file formats:

- CSV
- Parquet
- JSON (newline-delimited)

From the supported options, Parquet is recommended for a quicker load time if you have a choice of file formats. You might also consider compressing CSV or JSON files by using `gzip`. Compressed files are faster to transmit and take up less space than uncompressed files, but they are slower to load into BigQuery. The following sections further describe the considerations that are associated with using each data format.

3.4.1 Considerations for using CSV

When you load CSV data from Cloud Storage, you can load the data into a new table or partition, or you can append or overwrite an existing table or partition. BigQuery converts CSV data into the columnar format for Capacitor, BigQuery's storage format. When you load CSV data, consider the following:

- CSV files do not support nested or repeated data.
- If you use `gzip` compression, BigQuery cannot read the data in parallel. Loading compressed CSV data into BigQuery is slower than loading extracted data.
- Values in `DATE` columns must use the dash (`-`) separator and they must be in the following format: `YYYY-MM-DD` (year-month-day).
- Values in `TIMESTAMP` columns must use the dash separator (`-`) for the date portion of the timestamp, and the date must be in the following format: `YYYY-MM-DD` (year-month-day). The `hh:mm:ss` (hour-minute-second) portion of the timestamp must use a colon (`:`) separator.

BigQuery expects CSV data to be UTF-8 encoded. If your CSV files have data encoded in the ISO-8859-1 (also known as Latin-1) format, you should explicitly specify the encoding when you load your data so that BigQuery can convert it to UTF-8. For more information on using CSV files to load data into BigQuery, see Loading CSV data from Cloud Storage.

3.4.2 Considerations for using Parquet

Parquet is an open source, column-oriented data format that is widely used in the Apache Hadoop ecosystem. When you load Parquet data from Cloud Storage, you can load the data into a new table or partition, or you can append or overwrite an existing table or partition. BigQuery converts Parquet data into the columnar format for Capacitor.

BigQuery automatically detects the table schema from the Parquet file, using the last file alphabetically. BigQuery can convert Parquet data types to BigQuery data types to make them compatible with BigQuery SQL syntax. BigQuery does not support compressed Parquet files, but it does support compressed data blocks by using the Snappy and `gzip` codecs.

Note: Parquet files do not support nested `VARIANT` data. If your Snowflake data contains the `VARIANT` data type, use the JSON format instead.

For more information on using Parquet data, see Loading Parquet data from Cloud Storage.

3.4.3 Considerations for using JSON

When you load newline-delimited JSON data from Cloud Storage, you can load the data in a new table or partition, or you can append or overwrite an existing table or partition. BigQuery converts JSON data into the columnar format for Capacitor. When you load JSON data from Cloud Storage into BigQuery, keep the following considerations in mind:

- JSON data must be newline delimited.
- If you use `gzip` compression, BigQuery cannot read the data in parallel. Loading compressed JSON data into BigQuery is slower than loading uncompressed data. This method could cause significant delays and unnecessary costs.
- BigQuery does not support maps or dictionaries in JSON. For example, `"product_categories": {"my_product": 40.0}` is invalid but `"product_categories": {"column1": "my_product", "column2": 40.0}` is valid.
- Values in `DATE` columns must use the dash (`-`) separator and they must be in the following format: `YYYY-MM-DD` (year-month-day).
- Values in `TIMESTAMP` columns must use the dash separator (`-`) for the date portion of the timestamp, and the date must be in the following format: `YYYY-MM-DD` (year-month-day). The `hh:mm:ss` (hour-minute-second) portion of the timestamp must use a colon (`:`) separator.

BigQuery supports loading nested and repeated data from source formats that support object-based schemas, like JSON and Avro. For more information on using JSON files to load data into BigQuery, see Loading JSON data from Cloud Storage.

3.5 Migration tools

The following table lists the tools that you can use to migrate data from Snowflake to BigQuery:

Tool	Purpose
`COPY INTO <location>` command	Use this command in Snowflake to unload data from a Snowflake table directly into a specified Cloud Storage bucket. For an end-to-end example, see Snowflake to BigQuery on GitHub.
BigQuery Data Transfer Service	Perform an automated batch transfer of Cloud Storage data into BigQuery with this fully managed service.
`gsutil`	Copy downloaded Snowflake files into Cloud Storage with this command-line tool.
`bq`	Interact with BigQuery using this command-line tool. Common use cases include creating BigQuery table schemas, loading Cloud Storage data into tables, and executing queries.
Cloud Storage client libraries	Copy downloaded Snowflake files into Cloud Storage with a custom tool that uses the Cloud Storage client

	libraries.
BigQuery client libraries	Interact with BigQuery with a custom tool built on top of the BigQuery client library.
BigQuery query scheduler	Schedule recurring SQL queries with this built-in BigQuery feature.
Cloud Composer	Utilize this fully managed Apache Airflow environment to orchestrate BigQuery load jobs and transformations.
Apache Sqoop	Submit Hadoop jobs with Sqoop and Snowflake's JDBC driver to extract data from Snowflake into either HDFS or Cloud Storage. Sqoop runs in a Dataproc environment.

3.6 Migrating the data

You can export data from Snowflake using CSV, JSON, or Parquet files and transfer them to Cloud Storage. You can use the BigQuery Data Transfer Service for Cloud Storage to load the data into BigQuery. For details on how to use this service, see Overview of Cloud Storage transfers.

The following sections discuss the different options and processes to migrate your data from Snowflake to BigQuery.

3.6.1 Migration using pipelines

Depending on your business needs, your data can be migrated to BigQuery in different ways using the multiple tools that are available. The following lists provide several patterns that you can use to move your data.

3.6.1.1 Extract and load (EL)

Pipeline to unload data from Snowflake

1. Unload data from Snowflake directly into Cloud Storage (recommended), or download data and copy it to Cloud Storage using `gsutil` or the Cloud Storage client libraries.
    - Use the Snowflake2BQ tool to migrate data using the Snowflake `COPY INTO <location>` command.
2. Load Cloud Storage data into BigQuery with one of the following:
    - BigQuery Data Transfer Service
    - `bq` command-line tool
    - BigQuery client libraries

Pipeline using Snowflake's JDBC connection

Use any of the following products to export Snowflake data with Snowflake's JDBC driver:

- Dataflow
    - Google-provided JDBC to BigQuery template
- Cloud Data Fusion
    - Using JDBC drivers
- Dataproc
    - Writing to BigQuery with Apache Spark
    - Snowflake Connector for Spark
    - BigQuery Connector for Spark and Hadoop
    - Using Sqoop and Snowflake's JDBC driver to extract data from Snowflake into Cloud Storage:
        - Sqoop on Dataproc example

3.6.1.2 Extract, transform, and load (ETL)

If you want to transform your data before loading it into BigQuery, you can add a transformation step in the pipelines described in the preceding "Extract and load" section.

Pipeline to unload data from Snowflake

1. Follow steps 1 and 2 in the "Extract and load" section to either unload data directly from Snowflake to Cloud Storage or use `gsutil` to copy data over.
2. Transform and load your data into BigQuery with one of the following:
    - Dataproc
        - Reading from Cloud Storage with Apache Spark
        - Writing to BigQuery with Apache Spark
        - Hadoop Cloud Storage Connector
        - Hadoop BigQuery Connector
    - Dataflow
        - Reading from Cloud Storage
        - Writing to BigQuery
        - Google-provided template: Cloud Storage Text to BigQuery
    - Cloud Data Fusion
        - Reading from Cloud Storage
        - Writing to BigQuery
    - Dataprep by Trifacta
        - Reading from Cloud Storage
        - Writing to BigQuery

Pipeline using Snowflake's JDBC connection

Add a transformation step in the pipeline options described in the preceding "Extract and load" section.

- Dataflow
    - Clone the Google-provided JDBC-to-BigQuery template code and modify the template to add Apache Beam transforms.
- Cloud Data Fusion
    - Transform your data using the CDAP plugins.
- Dataproc
    - Transform your data using Spark SQL or custom code in any of the supported Spark languages (Scala, Java, Python, or R).

3.6.1.3 ELT

You might have a use case in which you want to load your data into BigQuery and then transform it. To do this task, you can load your data from Snowflake to a BigQuery staging table by using one of the methods in the "Extract and load" section earlier. Then, you run SQL queries on this table and write the output to the final production table in BigQuery.

3.6.1.4 Partner tools for migration

There are multiple vendors that specialize in the EDW migration space. For a list of key partners and their provided solutions, see Google Cloud's BigQuery partner website.

3.6.2 Example of the export process for migrating Snowflake to BigQuery

The following sections show a sample data export from Snowflake to BigQuery that uses Snowflake's `COPY INTO <location>` command. For a detailed, step-by step process that includes code samples, see the Google Cloud Professional Services Snowflake to BigQuery tool.

3.6.2.1 Preparing for the export

You can prepare your Snowflake data for export by using the following procedure:

- For the unload, use Snowflake SQL statements to create a named file format specification. We recommend using Parquet for tables with a flat schema and JSON for tables containing nested fields.

```
-- reference:
https://docs.snowflake.com/en/user-guide/data-unload-prepare.html#creating-a-named-file-format

create or replace file format my_parquet_unload_format
  type = 'PARQUET'
  field_delimiter = '|'
```

  This tutorial uses `my_parquet_unload_format` for the file format, but you can use a different name.

3.6.2.2 Exporting your Snowflake data

After you prepare your data, you need to move the data to Google Cloud. You can do this step in two ways: exporting your data directly to Cloud Storage from Snowflake, or staging your Snowflake data in either an Amazon S3 bucket or Azure blob storage. Google recommends exporting your data directly to avoid an extra data hop.

Export Snowflake data directly to Cloud Storage

You can use the Snowflake `COPY` command to unload data from Snowflake to Cloud Storage.

1. Configure a storage integration object to allow Snowflake to write to a Cloud Storage bucket referenced in an external Cloud Storage stage. This step involves several steps. First, you create an integration with the `CREATE STORAGE INTEGRATION` command. Next, you retrieve the Cloud Storage service account for Snowflake with the `DESCRIBE INTEGRATION` command and grant the service account permissions to access the Cloud Storage bucket that is selected as the staging area. Then, you can create an external Cloud Storage stage referencing the integration that you created with the `CREATE STAGE` command.

   Create a storage integration:

   ```
   --reference:
   https://docs.snowflake.com/en/user-guide/data-load-gcs-config.html#step-1-create-a-cloud-storage-integration-in-snowflake
   create storage integration gcs_int
     type = external_stage
     storage_provider = gcs
     enabled = true
     storage_allowed_locations = ('gcs://mybucket/unload/')
   ```

   Create an external Cloud Storage stage:

   ```
   --reference:
   https://docs.snowflake.com/en/user-guide/data-unload-gcs.html#creating-a-named-stage
   create or replace stage my_ext_unload_stage
     url='gcs://mybucket/unload'
     storage_integration = gcs_int
     file_format = my_parquet_unload_format;
   ```

2. Use the `COPY INTO <location>` command to copy data from the Snowflake database table into a Cloud Storage bucket by specifying the external stage object from step 1:

```
--reference:
https://docs.snowflake.com/en/user-guide/data-unload-gcs.html#unloading-data-directly-into-a-cloud-storage-bucket
copy into @my_ext_unload_stage/d1
from mytable;
```

Export Snowflake data to Cloud Storage through Storage Transfer Service from Amazon S3

The following example shows how to unload data from a Snowflake table to an Amazon S3 bucket by using the COPY command:

1. Configure a storage integration object to allow Snowflake to write to an Amazon S3 bucket referenced in an external Cloud Storage stage. This step involves configuring access permissions to the Amazon S3 bucket, creating the AWS identity and access management (AWS IAM) role, and creating a storage integration in Snowflake with the CREATE STORAGE INTEGRATION command. Then, you need to retrieve the AWS IAM user with the DESCRIBE INTEGRATION command, grant the AWS IAM user permissions to access the Amazon S3 bucket, and create an external stage with the CREATE STAGE command.

   Create a storage integration:

   ```
   --reference:
   https://docs.snowflake.com/en/user-guide/data-load-s3-config.html#step-3-create-a-cloud-storage-integration-in-snowflake
   create storage integration s3_int
     type = external_stage
     storage_provider = s3
     enabled = true
     storage_aws_role_arn = 'arn:aws:iam::001234567890:role/myrole'
     storage_allowed_locations = ('s3://unload/files/')
   ```

   Create an external Amazon S3 stage:

   ```
   --reference:
   https://docs.snowflake.com/en/user-guide/data-unload-s3.html#creating-a-named-stage
   create or replace stage my_ext_unload_stage url='s3://unload/files/'
       storage_integration = s3_int
       file_format = my_parquet_unload_format;
   ```

2. Use the `COPY INTO <location>` command to copy the data from the Snowflake database into the Amazon S3 bucket by specifying the external stage object that you created in step 1.

```
--reference:
https://docs.snowflake.com/en/user-guide/data-unload-s3.html#unloading-data-to-the-named-stage
copy into @my_ext_unload_stage/d1 from mytable;
```

3. Transfer the exported files into Cloud Storage by using Storage Transfer Service.

Export Snowflake data to Cloud Storage through Azure blob storage
Follow the steps detailed in Unloading into Microsoft Azure. Then, transfer the exported files into Cloud Storage by using Storage Transfer Service.

3.6.2.3 Load data into BigQuery

After you move your data to Cloud Storage, you can load the CSV, JSON, or Parquet files into BigQuery by using BigQuery Data Transfer Service.

4. Post-migration

After you complete your migration, you can use BigQuery's advanced features to use your data in various ways. The following sections describe some of the features available in BigQuery and best practices on how to make the most of your dataset.

4.1 Reporting and analysis

After you migrate your data into BigQuery, you can integrate one or more business intelligence (BI) solutions for reporting and analysis. For more information on how to use these solutions to gain compelling insights from your BigQuery data, see the Reporting and analysis guide.

4.2 Performance optimization

In general, queries that do less work perform better. When evaluating query performance in BigQuery, the amount of work required depends on various factors, including the following:

- Input data and data sources (I/O)
- Communication between nodes (shuffling)
- Computation
- Query anti-patterns

For details about how to optimize your query performance, see the Performance optimization guide in BigQuery.