HEVO

Best Practices

High Performance ETL to Snowflake

Table of Contents

1 Snowflake Introduction

Snowflake is a modern-day, easy to use analytics Data Warehouse designed for the cloud. Built on top of AWS, Snowflake's unique architecture uses a SQL Database engine. This makes Snowflake fast and flexible. One of the biggest challenges to set up the Snowflake Data Warehouse is to bring real-time data from all the different applications into Snowflake. This ebook covers vital Snowflake best practices while migrating data to Snowflake cloud data warehouse.

→ Data Warehouse Considerations

→ Table Design Considerations

→ Data Storage Considerations

→ Data Loading Considerations

2 Data Warehouse Considerations

Before you jump into optimizing the warehouse, you would have to understand the levers that matter.

(A) Understanding Credit charges on Snowflake

Snowflake offers pay per second billing. They allow the different size of the warehouse (Large, X-Large, 2X-Large, etc). Credit charges are calculated based on:

- The number of Clusters (if using a multi-cluster warehouse)
- The number of servers per cluster (a derivative of warehouse size)
- The time duration that each cluster in each server runs for

Having an understanding of this gives you the flexibility to run as many clusters as you want and suspend them when not in need. This, in turn, would help you act smartly and save costs. Try out combinations of different query types and warehouse sizes. This will help you arrive at the right combination for your workloads.

(B) Impact of Query Composition on Snowflake

Like in most Data Warehouses, the size and the complexity of the query determines the number of servers needed to process the query. For complex queries, the number of rows has less impact in comparison to the overall data size. The number of table joins, filtering using predicates, etc. has an impact on query processing hence utilize them carefully.

(C) Impact of Caching on Queries

Caching is standard practice in Data Warehousing as it improves the performance of the Warehouse as it enables subsequent queries to read from the cache instead of source tables.

While you consider suspending a warehouse to save credits, remember that you would be letting go of the data in the cache too. This will have an impact on query performance.

3 Table Design Considerations

While designing your tables in Snowflake, you can take care of the following pointers for efficiency:

Date Data Type: DATE and TIMESTAMP are stored more efficiently than VARCHAR on Snowflake. Hence, instead of a character data type, Snowflake recommends choosing a date or timestamp data type for storing date and timestamp fields. This, in turn, helps in improving query performance.

(A) Referential Integrity Constraints

It is recommended to implement referential integrity constraints in Snowflake. They provide valuable metadata that users can use to understand the schema and the relationships defined between tables.

In Snowflake, referral integrity constraints are not enforced by default. When created, they are disabled. NOT NULL is an exception and is enforced by default.

(B) Utilizing Clustering Keys

For big data sets, clustering is a good practice and helps improve query performance.

In Snowflake, there exists an automatic tuning and micro-partitioning feature. Most often, users load the data into Snowflake, organize into micro-partitions by timestamp or date and query it along the same dimension.

Clustering keys can be helpful in the following scenarios:

- Consider an example where data is loaded into Snowflake by timestamp, but the data is queried by ID. In cases where data is queried in a dimension different from what is loaded, clustering will be helpful.
- In case you figure through Query Profiling that a significant amount of the total query time is spent only in scanning the table, clustering can be of help. Often this applies to queries that filter many columns.

Here are a few things to note when clustering:

- One will have to manually run a DML statement to recluster a table.
- The existing data will be re-written in a different order upon reclustering. Snowflake saves the previous order for 7 days in order to provide Fail-safe protection, one of Snowflake's most lauded feature.
- Reclustering a table on Snowflake costs additional dollars. This is directly proportional to the size of the data set that is re-ordered.

4 Data Storage Considerations

Snowflake provides an array of features for data that is stored. Continuous Data Protection (CDP) which includes Fail Safe and Time Travel is given to all Snowflake accounts for no additional cost. This does not mean that CDP will not have an impact on your storage costs. On the contrary, it will.

Note, that your account will be charged for all the data stored in schemas, tables, and databases created in your Snowflake architecture. Based on the data stored and the duration for which it is stored, CDP has an impact on the storage costs.

Until the data leaves the Fail-safe state, storage costs will be incurred. This means that you pay for the data storage irrespective of whether it is in Active, Time-travel or Fail-safe State. Hence, it is important to make the right storage considerations.

Staged File Storage While Loading Data

To assist loading bulk data into tables, Snowflake has a feature called stages where files that have the data to be loaded are staged. Snowflake allows both internal (within Snowflake) and external (S3, Azure) stages. No additional cost is charged for Time Travel and Fail-safe features for data stored in internal stages within Snowflake. However, standard data storage costs apply. For bulk data, you could utilize this feature.

Cloning Tables, Schemas, and Databases

Snowflake has a zero-copy cloning feature that gives an easy way to take a “snapshot” of any schema, table, or database. This feature creates a derived copy of that object which initially shares the underlying storage. This can come handy when creating instant backups. This

does not incur any additional costs as long as you do not need to make any changes to the cloned object.

5 Data Loading Considerations

Preparing Your Data Files

(A) General File Sizing Recommendations

In order to optimize the number of parallel loads into Snowflake, it is recommended to create compressed data files that are roughly 10 MB to 100 MB in size.

Aggregate the smaller files to reduce processing overhead. Split the large files into a number of smaller files for faster load. This allows you to distribute the load between servers in the active Snowflake warehouse.

(B) Data Size Limitations for Semi-Structured Data

The VARIANT data type has a 16 MB (compressed) size limit on the individual rows.

Often JSON and Avro are the most commonly used data formats. Both JSON and Avro are a concatenation of many documents. The source software that provides JSON or Avro output will provide the output in the form of a single huge array having multiple records. Both line breaks and commas are supported for document separation.

For efficiency enhancement, while executing the COPY INTO <table> command it is recommended to enable the STRIP_OUTER_ARRAY file format option. This will load the records into separate table rows by removing the outer array structure. Below is an example:

```
COPY INTO <table_name>
FROM @~/<file_name>.json
file_format = (type = 'JSON' strip_outer_array = true);
```

(C) Data Size Limitations of Parquet files

It is recommended to split parquet files that are greater than 3GB in size into smaller files of 1GB or lesser for smooth loading. This will ensure that the loading does not timeout.

(D) Preparing Delimited Text Files

The following points must be considered while preparing CSV/Delimited text files for loading:

- Files must have data in ASCII format only. The default character set is UTF-8. However, additional encodings can be mentioned using ENCODING file format option.
- Within the files, records and fields should be delimited by different characters. Note, that both should be a single (necessarily not same) character. Pipe (|), caret (^), comma (,), and tilde (~) are common field delimiters. Often the line feed (\n) is used as a row delimiter.

- Fields that have delimiter should be enclosed in single or double quotes. If the data being loaded contains quotes, then those must be escaped.
- Fields that have carriage returns (\r \n) should be enclosed in single or double quotes too. In windows system, carriage returns are commonly introduced along with a line feed character to mark the end of a line.
- Each row should have the same number of columns.

Planning a Data Load

When dealing with large data sets, it is advised to dedicate individual Snowflake Clusters for loading and querying operations. This helps in optimizing the performance for both activities.

A standard virtual warehouse is enough to load data as loading requires fewer resources. Based on the speed at which you want to load data, you can choose the size of the warehouse. Remember to split large data files for faster loading.

Staging Data

Both Snowflake and your data source (Azure/S3) allow stage references via paths. It is a good practice to stage regular data sets by partitioning them into logical paths. This could include details such as source identifiers or geographical location, etc., along with the date when the data was written.

This exercise will provide you the flexibility to copy files by path using a single command. This will allow you to to take advantage of Snowflake's parallel operations by letting you execute concurrent COPY statements that match a subset of files.

Loading Data

Data is loaded into Snowflake using the COPY INTO <table> command. Understanding the levers of this command can go a long way in helping you optimize load in your data warehouse environment.

In order to execute the COPY command data from staged files to an existing table. The possible staging locations are as follows:

- Internal stage (table/user stage). PUT command can be used to stage files.
- AWS S3 bucket or Microsoft Azure container can be referenced from an external stage.
- An External location (AWS S3 bucket or Microsoft Azure container)

Options for Selecting Staged Data Files:

The COPY command accepts several options for loading data from a stage. This will enable you to bring only a fraction of data that is staged into Snowflake. Here are some options:

- Mention internal stages by path
- Use prefix to load data from Amazon S3 bucket
- Mention specific files to load
- Load data from specific files by using pattern matching

For details on the implementation of each of these, refer to Snowflake documentation.

Note that, as a general practice, while performing data loads for ETL, it is important to partition the data in your Snowflake or external locations like S3 buckets or Azure containers using logical, granular paths. By creating a partition using details like location or application

along with the date when this data was written, you are optimizing for a later data loading activity. When you load the data, you can simply copy any fraction of the partitioned data into Snowflake with a single command. You can copy data into Snowflake by the hour, day, month, or even year when you initially populate tables.

There is An Easier Way To Perform ETL!

The detailed steps of ETL processing using Snowflake mentioned involves multiple complex stages and can be a cumbersome experience. If you want to load any data easily into Snowflake without any hassle, you can try out **Hevo**. Hevo automates the flow of data from various sources to Snowflake in real time and at zero data loss. In addition to migrating data, you can also build aggregates and joins on Snowflake to create materialized views that enable faster query processing.

Hevo integrates with a variety of data sources ranging from SQL, NoSQL, SaaS, File Storage Base, Webhooks, etc. with the click of a button.

Sign up for a free trial or view a quick video on how Hevo can make ETL easy.

Published on: 1st March 2019