Use Case

Trace3 was engaged in a sales cycle with Snowflake Sales Engineering on demonstrating speed, efficiency and cost optimizations in data transformations that can be gained in Snowflake when compared with other on premises technologies such as SQL Server. The customer had primarily two challenges:

1. Data Transformation: Performing data transformations in SQL Server on premises took 34 hours.
2. Data Access: Time taken to render Tableau dashboards on was too long (~1 minute for 1 metric) over Hadoop. If Tableau extracts were used instead of Hadoop, complex calculations could not be achieved.

In parallel effort with Snowflake Sales Engineering team, Trace3 Data Intelligence team not only enabled the customer to ingest data from their SQL Server on premises into Snowflake by using SSIS and SnowSQL, but also orchestrated transformations in Snowflake through Snowflake's Python connector and SSIS. This was done to avoid rewriting SQL server stored procedures into Snowflake Javascript stored procedures, and also leverage parallel python constructs to optimize ETL timelines at unparalleled cost-performance. The details of the transformation would be posted in Part 2 of this article.

The results were as follows:

- Data Transformation: Compared to 34 hours in SQL server, Snowflake transformations were completed in 4.33 minutes with parallel Large Snowflake virtual warehouses that used only 4.53 credits. This equated to around $18.12 in Snowflake’s Business Critical Edition following Snowflake’s Cost calculator ($4 per credit).
- Data Access: Performance gains were demonstrated over a live connection to Snowflake vs Hadoop and Tableau extracts. With a Snowflake live connection, much more complex Tableau calculations could be done with Snowflake.

The end to end architecture designed in the evaluation cycle is shown below:

![Figure 1: End to End Architecture](image-url)
**Technical Details of SSIS Data Ingestion pipeline into Snowflake:**

The SSIS Ingest pipeline into Snowflake was comprised of two components:

1. **SSIS Script Task** that exported SQL server tables (masked/unmasked fields) into CSV GZIPPED files on local filesystem.
2. **SSIS Execute Process task** that called SnowSQL script to upload the CSV GZIPPED files from local filesystem into Snowflake.

The whole pipeline is shown below:

![SSIS Ingest Pipeline into Snowflake](image)

*Figure 2: SSIS Ingest Pipeline into Snowflake*

The SSIS script task iterated over the required fields and tables from SQL Server, cut the tables into chunks of 200,000 rows each and exported out multiple CSV GZIPPED files around 15-20 MB compressed, depending upon the number of the columns in the SQL server tables. This was done in order to stay within the best practices for file based data loading into Snowflake. Once the CSV GZIPPED files were available, SnowSQL script was called by SSIS to load the files into Snowflake.

**Example**

Assume we want to move two tables from SQL server into Snowflake: **SPOTIFY_RANKING** (3,441,200 records i.e. 3.4 M records) and **TOP_TRACKS_OF_2018** (100 records). Once the above pipeline is run, the SSIS script task creates the following folder hierarchy on the local filesystem. The two folders below in Figure 3 mark the names of the tables in SQL server.
Within each folder, SSIS script task generates multiple CSV GZIPPED files as shown in Figure 4. SSIS then calls SnowSQL script (documented on github) to put all the files into Snowflake's table stage, creates temp tables, loads data from table stages into temp tables and then runs merge statement to merge data from temp tables into the final tables. Once the SnowSQL script call is done, SSIS cleans up the exported files from the local filesystem. Figure 5 shows the load history into Snowflake.
Figure 5: Load History in Snowflake

Figure 6 shows the SPOTIFY_RANKING data loaded into Snowflake through the SSIS pipeline. As seen below, the TRACK_NAME is intentionally masked before data is loaded into Snowflake. This was done within the script task before the data was exported to CSV GZIPPED files such that any sensitive data can be masked before being loaded into Snowflake. The function used for this masking purpose was: convert(varchar(64), HASHBYTES('SHA2_512', Track_Name), 2) as Track_Name.

Figure 6: SPOTIFY_RANKING data in Snowflake
For the example listed above, loading two tables (SPOTIFY_RANKING: 3.4 M records and TOP_TRACKS_OF_2018: 100 records) from SQL Server into Snowflake took about 3 minutes on my home WIFI network over 1 single X-Small Snowflake virtual warehouse. I would have tested the load the ingest performance on bigger tables, but this is anyway much faster than Snowflake ODBC connector which does row by row inserts as documented by Charles Yorek, Senior Sales Engineer at Snowflake on Snowflake Blog.

Closing Comments:

There are several ways that this architecture can be enhanced, such as integrating with an object store for files and leveraging Snowflake’s Tri-secret secure and Storage integrations. However, the intent of this article is to present a story where even traditional integration engines like SSIS can also be used to ingest and transform data into Snowflake (more on transformations in Part 2 of the article) and achieve blazing performances. At Trace3, we value our partner relationship with Snowflake and look forward to helping customers on their journey from data to insights.