Part 1 went into the details of how the SSIS ingestion pipeline was designed into Snowflake. In this article, we will go into the details of how the existing SQL statements were reutilized with minimal change to perform transformations in Snowflake.

**Result Overview**
34 hours transformation time in SQL Server was brought down to 4.33 minutes in Snowflake at a fraction of the cost which equated to around $18.12 in Snowflake’s Business Critical Edition following Snowflake’s Cost calculator ($4 per credit). Additionally, the customer was able to perform much more complicated calculations within reasonable latencies from Tableau over a live connection to Snowflake, when compared with connections to Tableau Extracts or Hadoop.

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

![End to End Architecture](image)

*Figure 1: End to End Architecture*

As shown in Figure 1, SSIS was used to call a python script, which in turn called the Snowflake Connector for Python to perform all transformations in Snowflake.
Details of Snowflake Setup

The Snowflake setup done for the customer during the evaluation cycle is shown below:

1. Databases: The customer had two regional databases on SQL Server which were part of the evaluation criteria. For consistency purposes, the same database constructs were created in Snowflake and the data was brought into schemas within these two separate regional databases using the SSIS ingestion framework described in Part 1.

2. SQL Transformations: When the SQL Server stored procedure scripts were analyzed, it was noted that all the SQL statements in the stored procedure were independent of each other, since they were all computing GROUP BY's on different dimensions and inserting into a final table. Each regional database had 386 such independent SQL queries, thus resulting in total 772 queries to be executed in Snowflake for these two stored procedures. Below is an example of two independent SQL statements:

   - INSERT INTO Schema.ResultTable
     (Dim1, Dim2, Dim3, Dim4, Dim5, Expr1, Expr2, Expr3, Expr4)
     select 'V1', Dim2, Dim3, Dim4, Dim5, COUNT(DISTINCT Dim6) as Expr1, COUNT(DISTINCT Dim7) as Expr2,
     COUNT(DISTINCT Dim8) as Expr3, 0
     from Schema.BaseTable
     group by Dim2, Dim3, Dim4, Dim5;

   - INSERT INTO Schema.ResultTable
     (Dim1, Dim2, Dim3, Dim4, Dim5, Expr1, Expr2, Expr3, Dim6, Dim7, Dim8, Expr4)
     select 'V1', Dim2, Dim3, Dim4, Dim5, COUNT(DISTINCT Dim9) as Expr1, COUNT(DISTINCT Dim10) as Expr2,
     COUNT(DISTINCT Dim11) as Expr3, Dim6, Dim7, Dim8, 3 as Expr4, 'Dim6, Dim7, Dim8' as Expr5
     from Schema.BaseTable
     group by Dim2, Dim3, Dim4, Dim5, Dim6, Dim7, Dim8;

3. Virtual Warehouses: The customer wanted to see the maximum performance that could be achieved in transforming the data, while also minimizing and keeping track of how much Snowflake credits are being spent for each regional database. While Snowflake allows us to create virtual warehouses up to 4X-Large, spinning such huge virtual warehouses for doing the needed transformations would have been an overkill since we wanted to keep the costs low. Hence,

   - X-Small, Small, Medium and Large virtual warehouses were chosen as the baseline virtual warehouses and the transformations were performed on each warehouse type to see which warehouse size would be optimal for the customer.

   - Additionally, two separate virtual warehouses were created to keep track of transformation costs on each regional database. Having two separate virtual warehouses for transforming two different regional databases not only helped us track costs separately, but also helped us isolate transformation workloads from each other without affecting performance, which is a strong suite of Snowflake.

   - Snowflake provides a capability to setup Multi-Cluster warehouses to enable queries to be run concurrently on Snowflake virtual warehouses. Hence, we wanted to use that benefit and enabled auto-scale capability on virtual warehouses with minimal 1 cluster and maximum 10 auto-scaled clusters to be scaled with a Standard scaling policy decided by Snowflake depending upon the number of queries and the query complexity.

   - Auto-suspension and auto-resumption was also enabled on the virtual warehouses to keep the cost of the Snowflake virtual warehouses low, which again is another fantastic benefit of Snowflake.
Technical Details of SSIS Data Transformations:
The SSIS Transformation pipeline in Snowflake was comprised of one single component:

1. SSIS Process Task per regional database which would call the Python executable with arguments such as:
   - Path of the python script: Contained the logic to call Snowflake Connector for Python and execute all transformations
   - Path of the SQL Server stored procedure SQL script: Contained all semicolon delimited SQL queries.
   - Type of the file (sql vs csv): Support was added to parse SQL scripts delimited by semicolon, as well as if all those SQL statements were available as separate lines in a CSV file. The sql method was preferred to make the process easy.
   - Path type (relative vs absolute): Just to identify if paths to SQL scripts/CSVs were relative versus absolute.
   - Path to the results CSV file: Contained columns such as query number (0 to 385 in our example), start time of the query (example: 2019-10-31T15:37:20), total execution time of the query and the Query ID returned by Snowflake (example: 018fec4d-007f-7beb-0000-58750007e436).
   - Pool size: Parameter to control how many parallel queries to send to Snowflake. Default value = 20.
   - Region Number (1 vs 2): Parameter to control which region would be run. This controls the snowflake database name, schema and warehouse to run in the python script.
   - Query Order (parallel vs sequential): Parameter to control if all the SQL queries should be run in a parallel fashion if they independent of each other, or sequentially if the order of queries matters.

Since there were two regional databases that needed to be transformed in Snowflake, that resulted in two separate process tasks that would run in parallel in SSIS. This is shown in Figure 2.

![Figure 2: Parallel Process Tasks in SSIS](image-url)
The parameters in the SSIS process task are shown in Figure 3.

As mentioned above, SSIS process task called a python script which contained reference to the Snowflake Connector for Python and used python’s Threading library to parallelize the queries. Depending upon the pool size parameter set in the SSIS process task (default 20), 20 queries would be all running in parallel and soon as 1 query finished, another query would be added into the pool so that at each virtual warehouse is always running 20 queries at a time. In case any query was complex and a queue of queries would start to build up, Snowflake would automatically spin up multiple virtual warehouses due to auto-scaling policy configured during the virtual warehouse setup. The main code snippet that parallelized the execution of the queries 20 queries at a time is shown in Figure 4 below.
Example

In Part 1 of this series, SSIS was used to create a data pipeline from SQL Server into Snowflake where two tables were loaded into Snowflake: SPOTIFY_RANKING (3,441,200 records i.e. 3.4 M records) and TOP_TRACKS_OF_2018 (100 records). This example builds up on the same Spotify music data.

For the purpose of demonstrating parallelism,

- Two separate databases were setup: SSIS_REGION1_DB and SSIS_REGION2_DB
- Two X-SMALL virtual warehouses with autoscaling from 1 to 10 clusters with standard auto scaling, auto suspension after 10 minutes and auto resume were setup: SSIS_REGION1_WH and SSIS_REGION2_WH
- Within each database, MUSIC_LOAD_SCHEMA and MUSIC_TRANSFORMED_SCHEMA schemas were created where the MUSIC_LOAD_SCHEMA schema would contain the base tables in Snowflake and MUSIC_TRANSFORMED_SCHEMA would contain the transformed table. The transformed table SPOTIFY_METRICS seen in Figure 5 has the DDL: create or replace TABLE SPOTIFY_METRICS (UNIQUE_IDENTIFIER VARCHAR(36), METRIC VARCHAR(22), REGION VARCHAR(50), ARTIST VARCHAR(60), CNT NUMBER(18,0));

- Lastly, in order to simulate the customer scenario of independent SQL statements that was evaluated in the use case, 301 SQL statements were generated for each region, thus totaling 702 total queries that need to be run. The queries are shown in Figure 6 and are purely hypothetical and used to demonstrate the concept.
In order to demonstrate this example, the snowflake database names, schema names, warehouse names, account name, username and password were kept in a .env file on local filesystem which is looked up using the python package decouple. However, in production, keeping secrets in Azure Key Vault is highly recommended and can be accessed in python using the azure-keyvault-secrets python module.

When the SSIS job is run, each process task independently calls this python script and passes the runtime parameters to the python script. For this example, it was chosen to run 20 queries at a time in a parallel fashion so that virtual warehouses can parallelize the queries. This is shown in Figure 7.

It can be seen in Figure 7 left side command panel that as soon as query # 1 finished, another query # 21 was added to the pool so that 20 queries are running in the SSIS_REGION1_WH virtual warehouse at all times. In Figure 7 right hand side command panel, it was also observed that as soon as query # 1 finished, query # 32 was added to the pool so that 20 queries are also running in the SSIS_REGION2_WH virtual warehouses at all times. Since the order of the queries in this example is irrelevant, running query # 32 before # 21 did not have any effect on SSIS_REGION2_WH. However, if the need would have been that query # 21 should have been run before # 32 on SSIS_REGION2_WH at all times, it would have been important to run all queries in a sequential manner for SSIS_REGION2_DB database and that could have easily been controlled from the SSIS process task using the sequential value in the query-order parameter.

Figure 8 shows that all 301 queries on the SSIS_REGION1_DB were completed in 40.79 seconds and all 301 queries on the SSIS_REGION2_DB were completed in 40.28 seconds. Since both the SSIS process tasks were executed in parallel, the total time taken to process both the databases SSIS_REGION1_DB and SSIS_REGION2_DB was just max(40.79, 40.28) = 40.79 seconds.
The Snowflake History screen in Figure 9 shows the query execution in process when 20 queries were running each on SSIS_REGION1_WH and SSIS_REGION2_WH virtual warehouse at all times. Figure 9 also shows that the X-SMALL virtual warehouse automatically auto-scaled to 2 clusters due to the concurrent SQL queries. The completed run is shown in Figure 10.

Figure 9: Parallel queries running, few finished in Snowflake

Figure 10: All queries finished running in Snowflake

Closing Comments:

There are several ways that this transformation architecture could be enhanced, such as integrating secrets with Azure Key Vault. This article focused on how we avoided re-writing sql server stored procedure scripts into snowflake’s native javascript code and achieved blazing transformation performance not just technically, but also allowed the customer to keep their existing ETL investments through enhancing their existing code base in both SSIS ingest and transformations. This article also showed how we brought down the data transformation time from 34 hours in SQL Server to 4.33 minutes in Snowflake at a fraction of the cost through traditional tools like SSIS. We believe it was only possible due to the breadth of features Snowflake data platform provides which always accelerates any customer’s journey to cloud. If we haven’t said it enough yet, we value our partner relationship with Snowflake and always look forward to helping customers on their journey from data to insights.