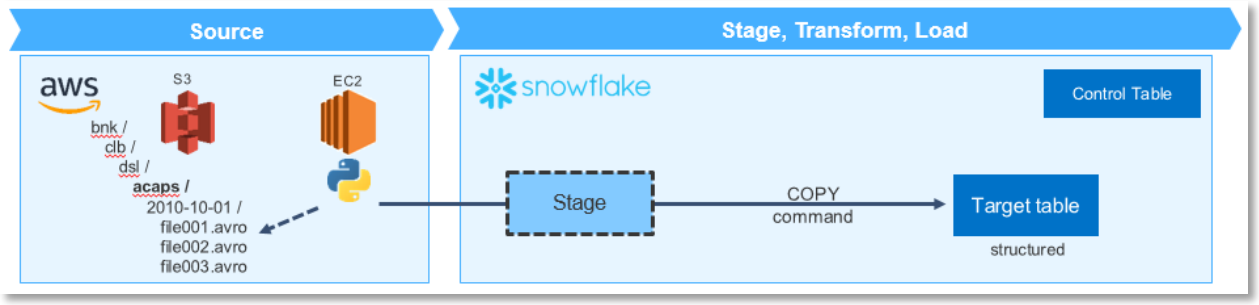
Table of Contents

Overview

The scope of this document is to cover the design and approach to the Initial ("Backfill") data load of historical data from Teradata into S3 and loaded into Snowflake's cloud data warehouse.

Data Flow Diagram



- Data from Teradata is extracted by Ab Initio ETL and landed in S3 as Avro files
- Python script connects to S3, traverses the bucket, pulls metadata into metadata and control tables, and dynamically generates DDL (e.g., create table) and DML (e.g., copy into) statements to load data from S3 into Snowflake

Snowflake Stage Object

- A single Stage object will be created in Snowflake to point to bucket **dfs-analytics-dlt-use1-ent-etl**

Metadata Tables

- Generally, these tables contain supplemental metadata from Teradata which is not available in Avro schema
- Tables are found in the ETL_METADATA schema

Tables

INFO_TBL

Columns

1. Table Name -- from source filepath
2. Avro schema -- JSON format extracted from a single Avro file per table
3. Hash key for Avro schema JSON, used to quickly compare against existing schema to uncover any changes
4. S3 bucket -- from source filepath
5. S3 bucket prefix -- from source filepath
6. Target Database -- target database name, (e.g., Snowflake)
7. Target Database Schema (e.g., T_SOT_DIRBNK_CARD)
8. Table DDL -- create table statement
9. COPY Command -- statement to load data to the table from staging area in S3

INFO_CLMN

- Static table extracted from Teradata, provided by @Praveen Guduri, DFS Cloud Data Enablement Team

Columns

1. Database Name
2. Table Name
3. Column Name
4. Data Type (including length, precision, scale as applicable)
5. Nullable Indicator
6. Default Values (including casting where applicable)

INFO_TBL_PK_FK

Primary Keys and Foreign Keys

- Primary and Foreign Keys will be replicated from Teradata
- For reference, the Teradata query used to populate this table is:
  - `select db_nm, chld_tbl_nm, indx_nm, indx_typ_cde, chld_clmn_nm, indx_clmn_seq_nbr, prnt_tbl_db_nm, prnt_tbl_nm, prnt_clmn_nm from dfs_stg.DQ_MDL_INDX_STRCTR where db_nm like 'T_%' and indx_typ_cde in ('P','F') order by db_nm, chld_tbl_nm, indx_typ_cde desc, indx_nm,indx_clmn_seq_nbr;`

Columns

1. Database Name - database for the primary key (PK) or foreign key (FK)
2. Table Name - table for the PK or FK
3. Key Name - always populated for both PK and FK, sourced from Teradata index name
4. Column Name - always populated for both PK and FK, sourced from Teradata child column name
5. Column Sequence - always populated for both PK and FK, sourced from Teradata index column sequence number
6. Foreign Key Referenced Database Name - Database referenced by FK
7. Foreign Key Referenced Table Name - Table referenced by FK
8. Foreign Key Referenced Column Name - Column referenced by FK

INFO_TBL_CLSTR_KEYS

Cluster Keys

1. From an approach perspective, Cluster Keys will be applied if and only if they are necessary. Per Snowflake documentation, Cluster Keys are only necessary for tables of size greater than or equal to 2 TB. We will first generate Create Table DDL's without Cluster Keys, then apply via ALTER TABLE statements if needed. To determine if tables need to be re-clustered, an exercise will need to be conducted to monitor query performance after completion of the backfill load for any given table.
2. Cluster Keys are determined as follows…
   a. First column is the partition key (e.g., PROC_DT) used to load from S3 to Snowflake. This is determined by parsing the prefix of the files in S3. So, for fin_el_dly_sum, we pull the highlighted… `s3://dfs-analytics-dlt-use1-ent-etl/cs/fin/cfr/fin_el_dly_sum/trnsfrm_dt_pc=2016-12-`

02/*.avro. Note the column name does not include the "_pc" suffix, which was added as a workaround. The actual column name does not include that.

b. Second column is the primary key for the table (e.g., ACCT_KEY) pulled from Teradata using query:

```
select databasename, tablename, columnname from
dbc.indicesv where databasename like 'T_%' and
indextype in ('Q','P') and columnposition = 1
order by databasename, tablename;
```

Columns

1. Table Name
2. Primary Cluster Key
3. Secondary Cluster Key

Audit, Balance, and Control (ABAC) Tables

- Generally, these tables contain operational metadata for the data loads
- Tables are found in the ETL_METADATA schema

Tables

CTRL_MAIN

- Table Name -- from source filepath
- S3 Bucket Name -- from source filepath
- S3 Bucket Prefix -- from source filepath
- Source Filename -- from S3
- Source File Row Count -- S3 metadata from file, showing row count
- Source Name -- S3 metadata from file, showing source name
- Source Load Type -- S3 metadata from file, showing load type (e.g., INSERT, UPSERT, REPLACE, DELETE)
- Source Type -- S3 metadata from file, showing source type (e.g., table, eds, etc)
- Source Sort Field List -- S3 metadata from file, showing sort field list
- Source Publish Mode -- S3 metadata from file, showing publish mode (e.g., APPEND)
- Source Org Code -- S3 tag from file, showing Organization Code
- Target Snowflake Load Status-- default to 'REQUESTED', update to 'FAILED' on error, or 'SUCCESSS' on success

CTRL_SNWFLK

- Stage Load Start Datetime -- delta load start datetime (only applicable for MERGE or DELETE operations)
- Stage Load End Datetime -- delta load end datetime (only applicable for MERGE or DELETE operations)
- Stage Load Row Count Parsed -- delta load source records parsed count (from INFORMATION_SCHEMA)
- Stage Load Row Count Loaded -- delta load source records loaded count (from INFORMATION_SCHEMA)
- Stage Load Row Count Errored -- delta load source records errored count (from INFORMATION_SCHEMA)

- Stage Load First Error Message -- delta load source first error message (from INFORMATION_SCHEMA)
- Stage Load Status -- delta load status (from INFORMATION_SCHEMA)
- Target Load Start Datetime -- target load start datetime
- Target Load End Datetime -- target load end datetime
- Target Load Row Count Parsed -- target load source records parsed count (from INFORMATION_SCHEMA)
- Target Load Row Count Loaded -- target load source records loaded count (from INFORMATION_SCHEMA)
- Target Load Row Count Errored -- target load source records errored count (from INFORMATION_SCHEMA)
- Target Load First Error Message -- target load source first error message (from INFORMATION_SCHEMA)
- Target Load Status -- target load status (from INFORMATION_SCHEMA)

Pseudocode

DDL Generation Script (controller_ddl.py)

Manually create CSV (driving table) with tables and partitions to scan

Python script iterates through partitions
Divides work by Table and then by Partition
For each Table…

1. Extract schema from Avro file using earliest partition date prefix (sub-directory), first file found
   - Column Names
2. Lookup Teradata metadata in INFO_CLMN for each table/column to enhance DDL with additional attributes:
   - Data Types
   - Nullability
   - Default value
3. Generate DDL for Target Table based on inputs from #1 and #2 above
4. Generate COPY statement to load Target Table based on inputs from #1 and #2 above

For each Partition…

1. Iterate through all Avro files within
2. Extract metadata and tags

Write extracted source file list to CTRL_MAIN table
Write extracted schema, metadata, and S3 tags to INFO_TBL metadata table

Build Tables Script (controller_build_tables.py)

Reads from INFO_TBL and executes generated DDL's to create Target tables
Execute GRANT Permissions Script sql

Load Script - Backfill (controller_load_initial.py)

Read list of files from CTRL_MAIN where operation type is INSERT and load status = REQUESTED

For each partition in CTRL_MAIN…

a. Lookup in INFO_TBL to pull the corresponding COPY statement
b. Execute COPY statement for all files in the partition
c. Capture load status (SUCEEDED / FAILED) and update record in CTRL_MAIN

Grant Permissions

- Looks for new tables that do not already have permissions granted and grants permission for those tables
- @Krishna Govindan (Business Technology Data Platform Team) can provide additional details

Helper Scripts

a. abac_utilities - series of SQL statements that may be called to insert / update metadata tables
b. db_communicator.py - abstraction for Snowflake database connector
    a. Reads file
    b. Logging In/Out
    c. Ensures correct timezone
    d. Ensures correct schema
    e. Ensures correct warehouse
c. extractor.py - returns Avro schema in JSON format for the parameters: S3 bucket, Prefix, and Filename
d. s3iterator.py - handles communication with S3

Error Handling

- Schema comparison
- Snowflake stores load history (COPY commands only) in view INFORMATION_SCHEMA.LOAD_HISTORY. Each record stores the success / failure of each file attempted to be loaded to Snowflake. For example, if the regex pattern ('.*.avro') in a given S3 bucket folder (prefix) found 10 files, the COPY statement would execute only one time
- Snowflake stores INSERT statement results in view INFORMATION_SCHEMA.QUERY_HISTORY

Archiving

- No purging of data is anticipated from the "ETL" S3 bucket (**dfs-analytics-dlt-use1-ent-etl**) for at least one year from the time of implementation (November 2018). Data files (Avro) will remain in bucket throughout the backfill, catchup, and ongoing loads.

Operational Reporting

- Platform team has a process in place to copy data from INFORMATION_SCHEMA.LOAD_HISTORY to a permanent table
- Talk to @Krishna Govindan (Business Technology Data Platform Team) can provide additional details

Backfill Utility Manual

The backfill utility is a command line tool that can be leveraged by Migration and Data Ops team to kickoff data backfill processes and migration from Teradata to Snowflake.

The tool currently only supports tables with INSERT only operations (No support for Deletes/upserts etc…) and can be run either one table at a time or several tables at a time (Available in CSV list)

Understanding main utility command

The backfill utility can be run with several arguments depending on the use case. Always reference the helper file available to understand what each one of the arguments does. The main command to run the backfill utility is below, with parameters in brackets:

python –m sfloadutil [--help] [--table TABLE] [--schema SCHEMA][--bucket BUCKET] [--noscan] [--noddl] [--noload] [--forcefiles]

Where each of the components in brackets are arguments to be added depending on use case (see below)

Parameter	Usage
-h, --help	Shows help message and exit.
--table TABLE	Table to load. **(Required if no csv is specified)**
--schema SCHEMA	Schema to load into. **(Required if no csv is specified)**
--bucket BUCKET	S3 bucket to look for data. Defaults to ETL (Avro) bucket. **(Optional)**
--csv CSV	Load from a CSV. **(Required if no table/schema is specified)**
--noscan	Do not scan S3 for DDL and file information. Currently implies noddl, as well. **(Optional)**
--noddl	Do not generate DDLs or GRANTs for specified tables. Loads will fail if table does not exist. **(Optional)**
--noload	Do not load tables from S3. **(Optional)**
--forcefiles	Forces a reload of files into Snowflake regardless if they have been loaded previously. If table contains data successfully loaded previously from the same files, then this parameter would introduce duplicate records into Snowflake. **(Optional)**

Backfilling table(s): Three steps utility usage

When needing to backfill one or more tables, the backfill utility can be used by following the below steps. Please read carefully as the steps must be done in sequence and require different users to execute the different steps depending on the access and permissions granted to each user

Step 1: Scan S3 and prep file metadata in Snowflake

The first step is to scan S3 to fetch all avro file information and construct DDL statements for the table(s) to be migrated. DDL and Loading are not executed during this first step. This is why the --noload and --noddl parameters of the utility must be specified like the below:

```
python -m sfloadutil --table ced_proc --schema t_sot_pymt_pls --noddl --noload
```

Note that in the above python statement, we are scanning S3 to find all avro files for the ced_proc table which belongs to the t_sot_pymt_pls schema in Snowflake.
The schema in Snowflake usually matches the location of the table folder in S3 (in this example, the S3 folder "ced_proc" would be in the etl bucket under /pymt/pls which translates to the Snowflake schema t_sot_pymt_pls)

If you are a user who has DDL privileges, please do not perform this step using your DDL Role in Snowflake (See "Credentials for Utility" section below), use your default role given to you to load data (Also used in Step 3). The separation of roles is necessary for SOX compliance

Note that you can, at any step of this process, pass a csv to the utility to run the utility for multiple tables at a time, you will have to replace the --table and --schema definitions in the above python command with --csv followed by the location of the csv file containing table name and schema name.

Step 2: Execute DDL in Snowflake

For this step, you will have to be a utility user that is able/permitted to run DDL statements in Snowflake. Users who are allowed to run DDLs are given special roles/warehouses in Snowflake that they will have to use when the utility prompts them for credentials (See section below "Credentials for utility" section for more information). If you do not have permissions to run DDLs in Snowflake, you must hand off this step to a user who does before moving to step 3

To execute the DDL the --noload and --noscan parameters of the utility must be specified like the below:

```
python -m sfloadutil --table ced_proc --schema t_sot_pymt_pls --noscan --noload
```

Step 3: Load data

Finally, in this step data is being copied from S3 to Snowflake.

If you are a user who has DDL privileges, do not perform this step using your DDL Role in Snowflake (See "Credentials for Utility" section below), use your default role given to you to load data (Also used in Step 1)

Run the DDL with the --noddl and --noscan parameters of the utility to load data like the below:

```
python -m sfloadutil --table ced_proc --schema t_sot_pymt_pls --noscan --noddl
```

Credentials for Utility

Every time and for every step the utility command is run (See above three steps), you will be prompted for a username, password, role and warehouse. Please provide the Snowflake username and password role and

warehouses assigned to you in Snowflake.
If you have the permission to both load data and run DDLs, the role and warehouse will be different for running DDLs (Step 2 in previous section). Please make sure to choose the correct Roles and Warehouses when using the utility. Steps 1 and 3 use the same role/Warehouse whereas step2 (Which is the step where DDLs are run) use different Snowflake role/warehouse. All steps require the same username and password.

If you do not believe you have permission to execute DDL, please see previous section above for step by step migration of table using this utility and when to hand off the DDL task to fellow peers who do have that privilege.

See below for example for credential prompts after starting utility with the main command (Password hidden)

```
[jkamel@icld1FOq2jE6cGE Snowflake]$ python3 controller_manual.py --table ced_proc --schema T_sot_pymt_PLS --noddl --noload
Snowflake Username: jkamel
Snowflake Password:
Snowflake Role: SFAAP_ETL_READWRITE
Snowflake Warehouse: BT_AA_ETL_STD_WH
```

On Screen Monitoring

Once the utility is started, messages are displayed on screen to provide high level information on what is happening as the script is running. Below are a few scenarios of what one should see on screen when running the utility:

a. Confirmation messaging showing that the script is scanning S3 for DDL and file information and loading it into Snowflake metadata tables **INFO_TBL** and **CTRL_MAIN** (if --noscan argument is used, the below will not be displayed):

```
[rbajaj@icld1FOq2jE6cGE one_time]$ python3 controller_manual.py --table indst_cl
ss --schema t_sot_refdata --forcefiles
refdata/indst_clss/trnsfrm_dt_pc=2008-12-01/indst_clss.20081201.part_000.avro, i
s a part_000 file so generating its ddl's.
```

b. While loading data from S3 to Snowflake
(if --noload argument is used, the below will not be displayed)

```
Setting Status Succeeded...
End Time: Mon, 19 November 2018 15:48:50
Source File#: 890
Start Time: Mon, 19 November 2018 15:48:51
Command issued:
        Time to copy: 0:00:24.583796
Setting Status Succeeded...
End Time: Mon, 19 November 2018 15:49:15
Source File#: 891
Start Time: Mon, 19 November 2018 15:49:16
Command issued:
        Time to copy: 0:00:19.714093
Setting Status Succeeded...
End Time: Mon, 19 November 2018 15:49:36
Source File#: 892
Start Time: Mon, 19 November 2018 15:49:40
Command issued:
        Time to copy: 0:00:14.608569
Setting Status Succeeded...
End Time: Mon, 19 November 2018 15:49:54
Source File#: 893
Start Time: Mon, 19 November 2018 15:49:56
Command issued:
```

Source File#: Partition counter showing how far along in files the script is at while migrating a table
Start Time: The start time of data loading for a particular file
End Time: The end time of data loading for a particular file
Time to Copy: Total time it took to load data from S3 to Snowflake for the file in

c. Message showing that the table is being created in Snowflake and Grant permissions given to appropriate roles
(if --noddl argument is used, the below will not be displayed)

```
Granting Role: GRANT USAGE ON STAGE SFAAP.T_SOT_DIRBNK_CARD_STG.SFAAP_EXT_ENT_ET
L_S3_AVRO to role SFAAP_T_SOT_DIRBNK_CARD_READWRITE;
```

d. Failure due to table already existing in Snowflake (implying it has been backfilled or attempted to be backfilled once already)

```
[rbajaj@icld1FOq2jE6cGE one_time]$ python3 controller_manual.py --table indst_cl
ss --schema t_sot_refdata
The following errors were found:
Table exists in info_tbl.
```

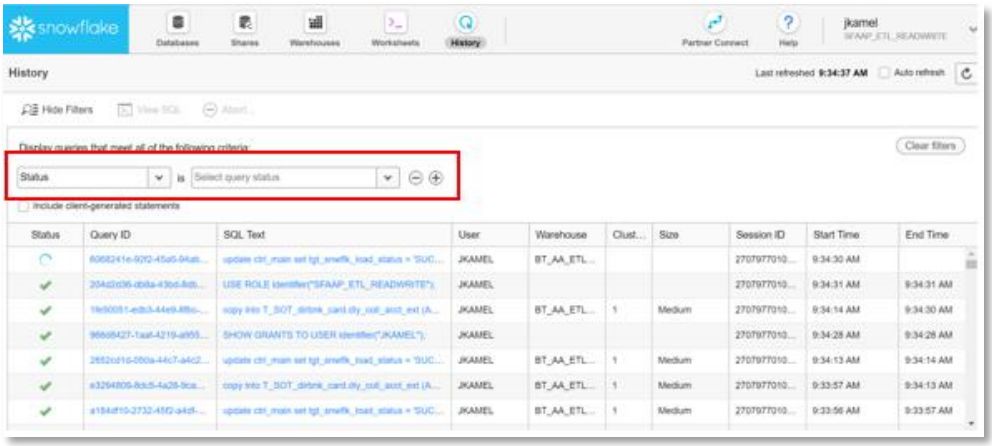
e. If user doesn't specify any argument, the error message below is displayed

```
[rbajaj@icld1FOq2jE6cGE one_time]$ python3 controller_manual.py
No work was given to do.   Exiting.
```

Snowflake Monitoring

- Query History Tab:

The query History tab in Snowflake displays currently running queries with the ability to filter and find specific queries of interested(Filter Red box in screenshot below) Most commonly used filters are "Status" to check on running/succeeded queries and specific "SQL Text" search (In case looking for a particular table/partition or script)



In the screenshot above, the SQL Text column shows the exact SQL command running in Snowflake. Looking at this column helps identify the different running processes in Snowflake while the utility is running. Click on the SQL text to see the full query running to identify which partition/table is being processed.

"COPY INTO": Indicates the COPY command that is loading a partition in S3 into Snowflake
"Update CTRL_MAIN": Indicates that the status of a partition is being updated to Success/Failure depending on whether the files in the partitions were loaded successfully or not
"CREATE Table": Indicates that the DDL script is running to create a table. Note that the utility attempts to create all tables found in INFO_TBL every time it is ran if the table doesn't exist yet in Snowflake
"GRANT Usage": Indicates that the grant statement is running after a table has been created

- <u>Snowflake Query to check on partition and file load statuses:</u>

  To run a query in Snowflake, first go to the "Worksheet" tab then select your appropriate role, warehouse and database in the context section (Red box below) to be able to run queries:

  Role: Role provisioned to you when your Snowflake account was setup at DFS to oversee migration
  Warehouse: Warehouse available to you for querying when your Snowflake account was setup at DFS
  Database: SFAAP
  Schema: ETL_METADATA

  Users can check on paritions in Snowflake that failed being copied using by querying the CTRL_MAIN table in Snowflake using the SQL statemtn below.
  Note that CTRL_MAIN has additional information (See previous "Audit Balance And Control' section of this documentation for full description of CTRL_MAIN)

```
select
  tbl_nm, -- shows table name
  tgt_snwflk_load_status, -- display the load status
  src_prefix  -- shows the full partition prefix

from sfaap.etl_metadata.ctrl_main
where tgt_snwflk_load_status = 'FAILED';
```

Status can be one of three main values: "SUCCEEDED","FAILED", "REQUESTED-MAN"…

- **SUCCEEDED**: File/Partition loaded successfully in Snowflake
- **FAILED**: File/Partition did not load successfully in Snowflake
- **REQUESTED-MAN**: File/Partition requested by utility to be loaded, pending load attempt

Notes:

- Status is set at the partition level (i.e., if one file under a partition fails to load, then all files under that partition in CTRL_MAIN will be set to 'FAILED').
- Additional filters in the WHERE clause can be added to the SQL statement such as looking for a specific table, file or partition.

Retrying failed partitions

In case partitions being loaded failed (See SQL statement to identify failed partitions under the "Snowflake Monitoring" section above), below are the steps to be taken to reload the partitions needed:

1- Identify the reason why data loading failed (schema change, S3 metadata missing etc…)
2- Update CTRL_Main records for partitions that need to be retried from "FAILED" to "REQUESTED-MAN" using the statement below.

   If all failed files under a table need to be reloaded:

```
UPDATE SFAAP.ETL_METADATA.CTRL_MAIN
SET tgt_snwflk_load_status = 'REQUESTED-MAN'
WHERE tgt_snwflk_load_status = 'FAILED' AND TBL_NM = [Table Name to be re-loaded]
```

   If all failed files under specific partitions need to be reloaded:

```
UPDATE SFAAP.ETL_METADATA.CTRL_MAIN
 SET tgt_snwflk_load_status = 'REQUESTED-MAN'
 WHERE tgt_snwflk_load_status = 'FAILED' AND
SRC_PREFIX IN ([Comma separated list of partition level source prefixes])
```

3- Run Step 3 of utility again (See step by step instructions mentioned previously in this doc), this entails running python utility with -- noscan and -- noddl parameters) for the

wanted table. This will only load data for the files in CTRL_MAIN where the status has been set to 'REQUESTED-MAN' for the specified table.