****

-----------------------------------------------------------------------------------------------------------------

**Design Documentation**

* [Motivation](#bookmark)
* [Applicability](#bookmark1)
* [Structure](#bookmark2)
* [Implementation](#bookmark3)
* Job Status Table

- Structure

- General Flow

* [Consequences](#bookmark4)

**Motivation**

For storing IMS files, processed files and transformed files we follow the basic structure for the clear and better understanding of the storage. We will be following Extract-Transform-Load that takes the data from source, transforms and stores it in Snowflake. This document defines how we will be storing data in different stages and its flow through the process.

**Applicability**

This pattern applies to all the data stored in Snowflake. We will be using a package called ETL, the package contains a class called “Etl” with three methods in it namely:

1. extractData
2. loadSnowflake
3. loadStage

The class takes two arguments.

* First the path of config file which is a yaml type file.
* Second the path where the data extracted for a table is to be stored.

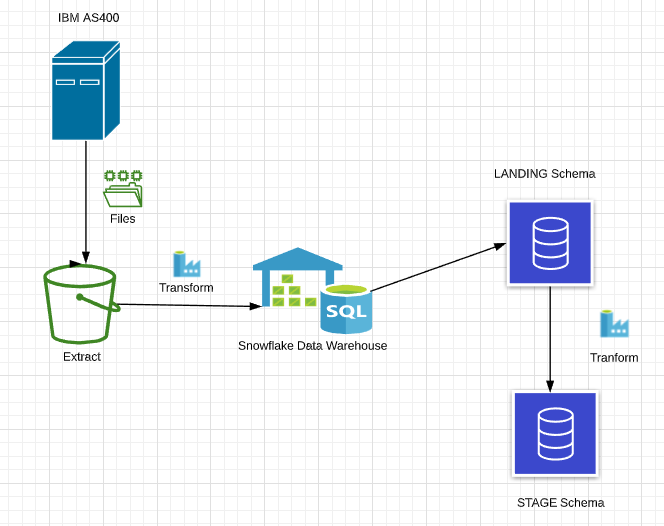
The config file contains the connection credentials of IBM and Snowflake as well as a table’s custom/trigger query if any and the number of partitions to make while extracting data for a table.

The initial flow of the data is given below.

Extract job 🡪 Load in Landing of Snowflake Job 🡪 Load in Stage of Snowflake

Raw files are extracted from the source IBM with the help of a python job which connects the source IBM and destination Snowflake. When the files are successfully extracted from the source, they are transformed and loaded in the Landing schema of Snowflake. After which a stored procedure is called which identifies the files as UPSERT, TYPE\_2 or INSERT and loads it into STAGE schema of Snowflake. There are different users which have access to different schemas. There are two types of users. API and BI users.

**Overall Structure­­­­­**

****

We will be considering the IBM files as source files in the below . We will be running the Extract job in order to get the files from Source. Transform it in load stage and finally differenciate as UPSERT, INSERT and TYPE\_2 in STAGE schema in snowflake.

**Implementation**

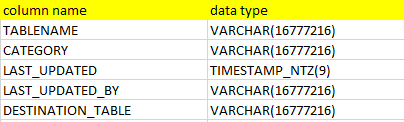
Following are the steps of how each class works:

Following are the steps how each method of class works: -

1. extractData:
   1. The method takes name of table to be extracted as the argument.
   2. A connection variable for both ETL server and Snowflake is formed using the config file.
   3. Next the latest extract start date is obtained of the table’s last successful ETL job from the job status table present in Snowflake which will be used for delta tables if any. If it is the first ETL job then the last extract date is taken to be 1900-01-01.
   4. If the table has a custom/trigger SQL query then that is used else SQL \* query is used for extracting. The custom/trigger query is taken from the config file.
   5. The number of records to be extracted and number of partitions to make are stored in two variables. The number of partitions is taken from config file.
   6. For each extract happening, the files of previous extract are overwritten. So, if the number of partitions change in future, then the files are deleted from the folder.
   7. If the number of records is less than 1000 only one partition is made, else if less than 10000 then 10 partitions are made. Else the given number of partitions is used.
   8. Next iterating in the range of number of partitions, using pandas dataframe, limit and offset, records are partitioned and saved as a csv file in the destination folder.
   9. If the extract is completed successfully, then a record is inserted in the job status table in Snowflake with new job id, job name as table name, start and end time of extract and status as SUCCESS. If the extract fails, then end time is kept as null and status as FAIL.
2. loadSnowflake:
   1. The method takes name of table to be loaded in landing schema as an argument.
   2. The connection variable of Snowflake using snowflake-connector-python module is formed using config file.
   3. Then the extracted data is inserted in respective table in landing schema of database using PUT and COPY Snowflake commands executed by cursor.
   4. If the loading is completed successfully, then the respective extract record is updated with load start and end time and status as SUCCESS. If the load fails, then load end time is kept null and status as FAIL.
3. loadStage:
   1. The method takes name of table to be loaded in stage schema from landing as an argument.
   2. The connection variable of Snowflake using snowflake-connector-python module is formed using config file.
   3. Then a Snowflake stored procedure called SP\_LOAD\_STAGE is executed which loads data in stage tables with load strategies being TYPE 2, UPSERT or INSERT ONLY.
   4. If the loading in stage is completed successfully, then the respective extract record is updated with load start and end time and status as SUCCESS. If the load fails, then load end time is kept null and status as FAIL.

SP\_LOAD\_STAGE stored procedure:

1. It takes table name as its argument.
2. Which load strategy to use for a table is determined from the REFERENCE\_TABLE in Snowflake landing schema. The load strategy to use for a table should be updated in the REFERENCE\_TABLE along with table name. Following is the table structure of the REFERENCE\_TABLE:

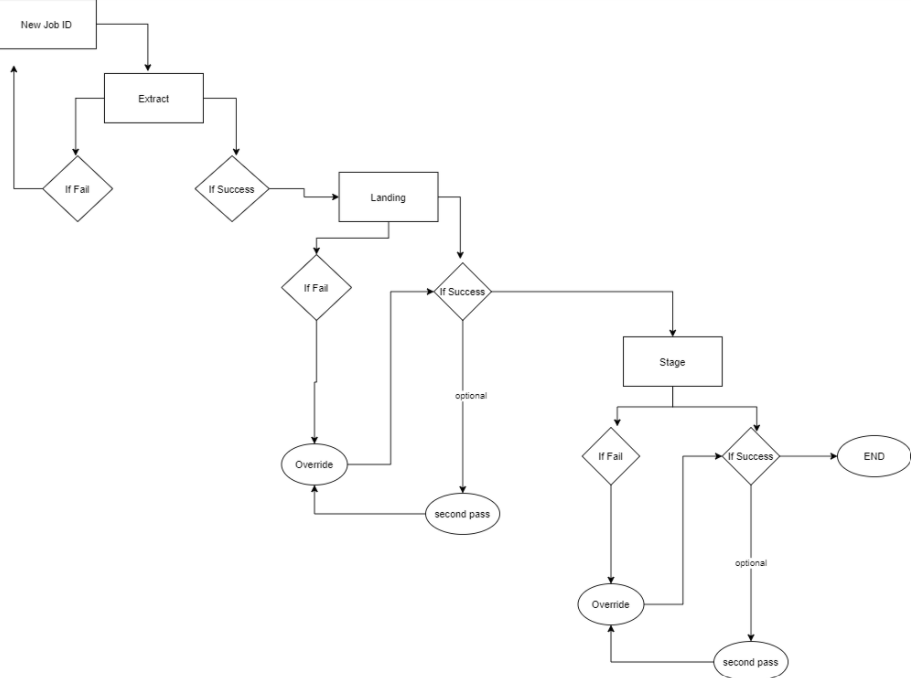


TABLENAME is the name of table in landing schema and DESTINATION\_TABLE is the name of table in stage. There are cases where two tables in landing have the same destination table in stage.

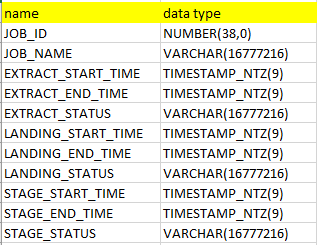
1. The procedure adds 4 columns named EFFECTIVE\_START\_DATE, EFFECTIVE\_END\_DATE, LAST\_UPDATED and LAST\_UPDATED\_BY. The EFFECTIVE\_START\_DATE keeps the timestamp when the record was inserted. At that time EFFECTIVE\_END\_DATE is null indicating the record is latest. When EFFECTIVE\_END\_DATE is not null, it means the record has expired. The LAST\_UPDATED column has the timestamp of when the record was last modified.

LAST\_UPDATED\_BY is a hardcoded value “ETL”.

**Job Status Table**



**Structure :** This is the structure of job status table in snowflake



**General Flow:** The general flow of the ETL job is as follows: -

**Extract job 🡪 Load in Landing of Snowflake Job 🡪 Load in Stage of Snowflake**

1. *Extract*
2. The user gives table name to be loaded as the argument in extract.
3. A new Job ID is created in the Job\_Status\_Table in snowflake each time the extract is run.
4. If the extract fails for certain records it will show a ‘Fail’ status in the Job\_Status\_Table.
5. IF ‘Fail’ the extract needs to be run again which will generate a new Job ID.
6. The failed records will be reflected in Daily\_Error\_Table
7. IF ‘SUCCESS’ then the process moves to next stage that is load Landing.
8. *Load Landing*
9. After extract success, The user gives table name to be loaded in the landing schema in Snowflake as the argument.
10. This job uses PUT and COPY\_INTO snowflake commands to load in the extracted table in the Landing Schema.
11. If the Load fails for certain records it will show a ‘Fail’ status in the Job\_Status\_Table.
12. IF ‘Fail’ , then the Load job is run again which overrides the previous Fail status until it shows ‘Success’ .
13. The failed records will be reflected in Daily\_Error\_Table
14. IF ‘Success’ then the process moves to next stage that is load Stage.
15. If the user wants to load the table which has shown ‘Success’ previously (second pass), it will override this record in the Job\_Status\_Table.
16. *Load Stage*
17. After Load success, the user gives table name to be loaded in the Stage schema in Snowflake as the argument.
18. Next, a Snowflake stored procedure called SP\_LOAD\_STAGE is executed which loads data in stage tables with load strategies being TYPE 2, UPSERT or INSERT ONLY.
19. If the Load fails for certain records it will show a ‘Fail’ status in the Job\_Status\_Table.
20. IF ‘Fail’, then the Load job is run again which overrides the previous Fail status until it shows ‘Success’
21. The failed records will be reflected in Daily\_Error\_Table
22. IF ‘Success’ then the process is completed.
23. If the user wants to load the table which has shown ‘Success’ previously (second pass), it will override this record in the Job\_Status\_Table.

**Consequences**

This pattern will allow different users to access data to perform further analytics .