Meets Specifications

Dear student

Congratulations on completing the project! You should be very proud of your accomplishments in building an awesome ETL pipeline with Airflow! You have now the knowledge of how to build dynamic and reusable ETL pipelines and how to make sure the data quality meets specifications! 😊 👏

General

DAG can be browsed without issues in the Airflow UI

The DAG looks just like we expected it to be and the task dependencies follow the required data flow. Great job! To make the DAG even more compact, you could try to use the SubDag operator with the dimension loads and hide the repetitive parts behind that.

The dag follows the data flow provided in the instructions, all the tasks have a dependency and DAG begins with a start_execution task and ends with a end_execution task.

Dag configuration

DAG contains default_args dict, with the following keys:

- **Owner**
- **Depends_on_past**
- **Start_date**
- **Retries**
- **Retry_delay**
- **Catchup**

The DAG object has default args set

The DAG configurations look great, the default params prevent now unnecessary backfills when the DAG is started and retry interval is set to 5 minutes.

Especially the depends_on_past and catchup params are the ones that are great to understand to avoid unnecessary backfills when adding new DAGs. Nice job on configuring those args!

The DAG should be scheduled to run once an hour

Awesome job!

Staging the data

There is a task that to stages data from S3 to Redshift. (Runs a Redshift copy statement)

Great job building the staging operator and just having single operator to stage both. The copy statement is very similar on both cases and nice usage of Python string operator to keep the code clean and readable.

Also very good usage of parameters in the operators and including all the copy statements parts as task params to allow the usage on several csv files types with different header and separator setting.

Instead of running a static SQL statement to stage the data, the task uses params to generate the copy statement dynamically

The operator contains logging in different steps of the execution

The SQL statements are executed by using a Airflow hook

Loading dimensions and facts

Dimensions are loaded with on the LoadDimension operator

Great work on the dimension loads and adding the possibility to choose between insert modes.

It is related to source datasets but quite often the dimension tables can be small in size and need to represent only the latest values. In these cases it is good to use delete-insert on the dimension loading.

Facts are loaded with on the LoadFact operator

Great work on this, fact tables in large scale environments are usually very large and use append only since the merge/upsert operations are simply too costly when you have billions of rows of data.

This could be even simplified so that the ‘INSERT INTO [Table]’ statement would be included in the SQL statement, but it is also ok to keep those separate.

Instead of running a static SQL statement to stage the data, the task uses params to generate the copy statement dynamically

The DAG allows to switch between append-only and delete-load functionality

Data Quality Checks

Data quality check is done with correct operator

The data quality operator looks awesome. It is simple and in few lines of codes, but still allows you to do import checks on the data and catch the possible data quality issues as soon as possible.

The DAG either fails or retries n times

Operator uses params to get the tests and the results, tests are not hard coded to the operator