Week 3 - Implementation: Staging layer

The objective of the third week is to create the staging layer with all data sources including the google sheet that contain the list of companies and repositories that will be analyzed in depth.

General

Data sources

- Stack Overflow (BigQuery)
- GitHub (BigQuery)
- Google sheet (local file)

Best practices for dbt modeling

- dbt style guide
- Create a primary key in every model, called _*pk*.
- Make sure that there is a timestamp column which indicates the creation or load datetime of each record.
- Make sure that every datetime is in UTC format.
- Add _*datetime_utc* postfix to every datetime column name.
- Add *is_* prefix to every boolean type (‘Yes’/’No’, 1/0) column name and convert it to boolean (true/false).

dbt packages

- dbt codegen package
- dbt expectations package (optional)

Development principles

- During the development try to work with smaller subsets, don’t run the queries on full tables if it is not necessary and the smaller subset has the same result.
- Estimated process cost can give you hints in BigQuery.
- Use the Preview dataset option when possible (no table scan there).
- In some cases, instead of dbt run use dbt compile before, you can see the query before it actually runs and make changes if it is needed.

GitHub dataset

The GitHub dataset is located in the BigQuery environment and curated and updated by Google.

More information about the BigQuery public datasets:

Big Query public datasets

The GitHub data is available in three levels of aggregation: day, month, and year. In this project, the daily data source will be used.

Location

GitHub Activity Data dataset

Objective

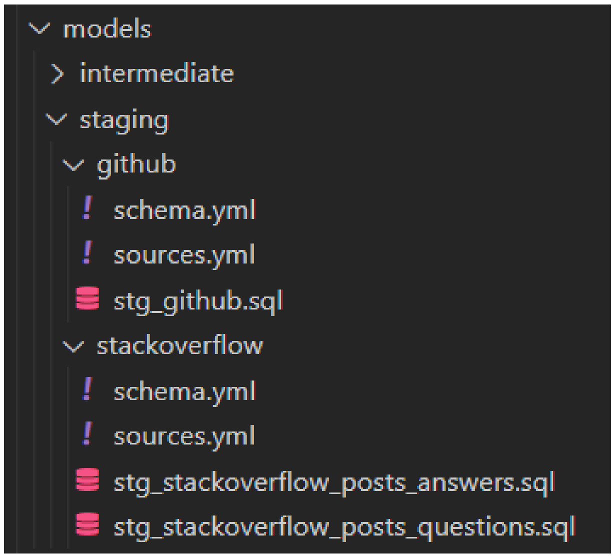
Create a staging layer for the GitHub data in dbt according to the dbt principles and best practices.

Steps

- Add the dbt codegen package to the project.
- Add folder lever materialization configurations in the dbt_project.yml file.
    - Make sure that the staging models all materialized as views.
- Create the folder structure inside the dbt project.
- Create the sources.yml file that contains the GitHub data using dbt codegen package.
    - Use BigQuery's wildcard function to union all the daily tables, but make sure that you use a limited dataset for development.
    - Ensure that the dbt sources.yml file can handle the wildcard.
    - Check if you have troubles adding wildcard to sources.
- Create the staging model materialized as a view using 2022 data only.
    - Optionally, the dbt codegen package can be used for that.
- Add best practices to the model.
- Make sure that the RECORD type fields are readable in the model the same way as it is in the source tables.
- Make sure that there is a unique field in the table.
- Create the schema.yml using dbt codegen package
- Add uniqueness and not null tests for the primary key field.
- Add proper documentation to the project.
    - Add a few descriptions to columns which are not intuitive enough to understand without context.

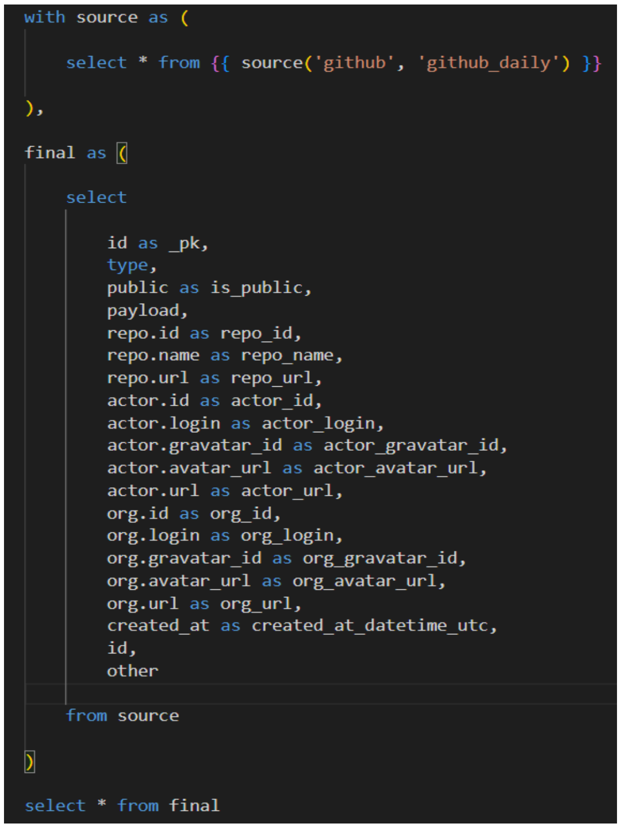
Examples

Possible folder structure



```
models
  intermediate
  staging
    github
      schema.yml
      sources.yml
      stg_github.sql
    stackoverflow
      schema.yml
      sources.yml
      stg_stackoverflow_posts_answers.sql
      stg_stackoverflow_posts_questions.sql
```

Possible staging model output



```
with source as (

    select * from {{ source('github', 'github_daily') }}

),

final as (

    select

        id as _pk,
        type,
        public as is_public,
        payload,
        repo.id as repo_id,
        repo.name as repo_name,
        repo.url as repo_url,
        actor.id as actor_id,
        actor.login as actor_login,
        actor.gravatar_id as actor_gravatar_id,
        actor.avatar_url as actor_avatar_url,
        actor.url as actor_url,
        org.id as org_id,
        org.login as org_login,
        org.gravatar_id as org_gravatar_id,
        org.avatar_url as org_avatar_url,
        org.url as org_url,
        created_at as created_at_datetime_utc,
        id,
        other

    from source

)

select * from final
```

Stack Overflow dataset

The Stack Overflow dataset is technically located in the same place as the GitHub dataset.

Location

Stack Overflow dataset

Objective

Create a staging layer for Stack Overflow data in dbt according to the dbt principles and best practices.

List of tables

Stack Overflow has a different structure for storing the data.

In this part of the project, two tables will be used:

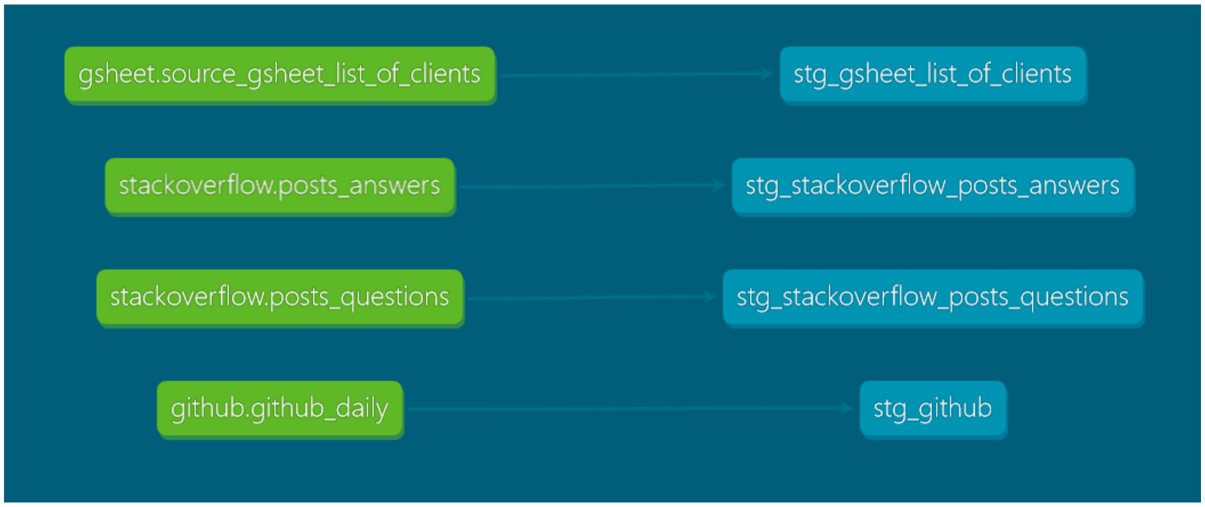
- posts_questions
- posts_answers

Steps

- The same steps as it is for GitHub data.
- Different methods:
    - The granularity of the Stack Overflow data is different because one table contains all the records.
    - Because of this, the time period should be filtered out inside the staging model.
    - Find the column that will function as a key for that purpose.
        - Explore the dataset by matching the rows with actual Stack Overflow questions to determine which column can be the period filter column.
        - The objective is to have every question and the related answers from 2022.

Examples

Possible outcome of the dbt DAG


gsheet.source_gsheet_list_of_clients
stg_gsheet_list_of_clients
stackoverflow.posts_answers
stg_stackoverflow_posts_answers
stackoverflow.posts_questions
stg_stackoverflow_posts_questions
github.github_daily
stg_github

Google sheet

The objective is to create an ingestion pipeline that ingest the google sheet data into a BigQuery table. Then create a staging view in dbt based on that ingested BigQuery table. For the first task, Airbyte will be used.

Location

Each student has their own Company Details google sheet in their own google environment

This spreadsheet is being used and updated by other consumers, therefore the design does not match the expected database look. The goal is to create a table inside the project database based on that spreadsheet and have the same set of data inside that table.

Steps

- Setup a source (Google Sheets) and a destination (BigQuery) following the Airbyte documentation.
    - Google Sheet as source
    - BigQuery as destination
- Create a new dataset in the project database called *raw_airbyte*. Inside the dataset create a table that can be the destination of the google spreadsheet. Make sure that the column names are in a database standard format (spaces, lowercases, etc.).
- Create a connection that ingests the source data into the BigQuery table.
- Apply one-time ingestion of the source.
    - The scheduling of the Airbyte ingestion will be configured in a later phase of the project.

- Create a staging model in dbt as a view based on the already created and ingested google sheet table.
- Apply the same principles and best practices as for the previous data sources. Notice that there is no datetime or unique key and it should be created manually. It can be created with Airbyte or inside the dbt model.

Additional tasks

Testing

- dbt has a strong belief in the importance of testing and is dedicated to support proper testing methods throughout the whole project.
- In the staging layer uniqueness and not null tests are defined, but it is possible to add other tests if it is necessary.
- Tests in dbt are basically queries that also have process cost. It is a double-edged sword to find the balance in the number of tests applied.
- After exploring the source data, dbt built-in tests or other tests can be added to the staging models considering process costs.
- Tests can be added in a later phase of the project when the business requirements are clearer.
- Few possible examples:
    - Key fields always have values.
    - Referential integrity between related tables (TBD in the intermediate layer).
    - Minimum or maximum date of the records matches the expected content of the table.

Using variables

- It is more sophisticated to add project-level variables to handle and manage time periods and other dimensions.
- With this solution, the time period can be managed from one source.
    - The time period can be controlled by changing only this one project-level variable.
    - It is possible to use only one variable and then manipulate that variable depending on which data source is being used.
    - It should be considered that it can have more maintenance overhead at the end then using more variables for each dataset. But prefer using only on year variable for the project.

Branching strategy in the GitHub repository

It is common to use feature branches for development cycles. In this case, everyone has their own branch inside the repository. In that phase of the project, there is no need for feature branches as one individual working on their own branch only. Later, when the project is more complete and it is needed to add a new source or test some function outside the main branch, a feature branch should be created.