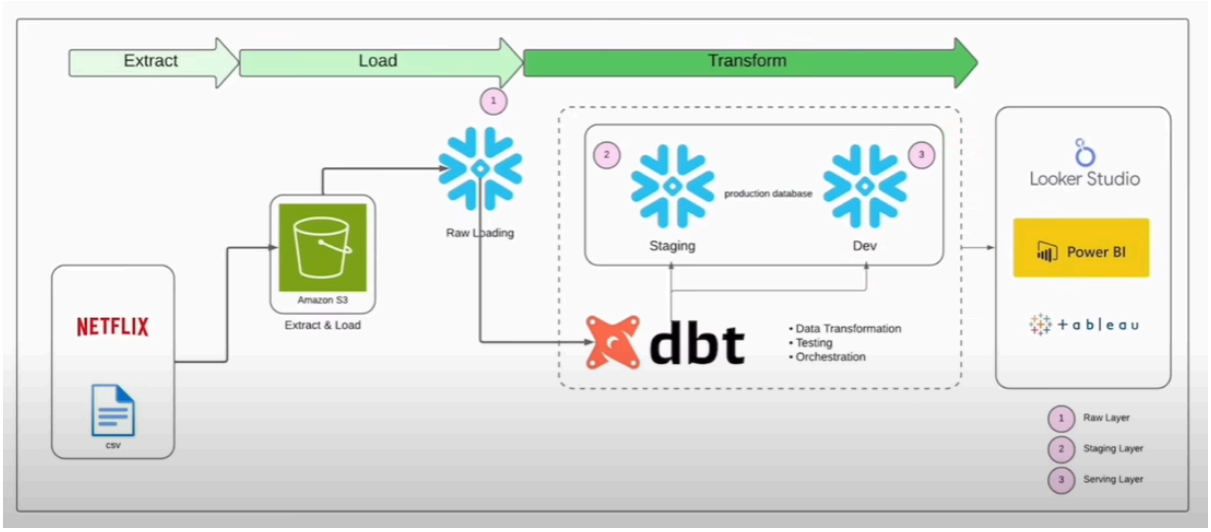
NETFLIX ELT DATA PIPELINE

Architecture:

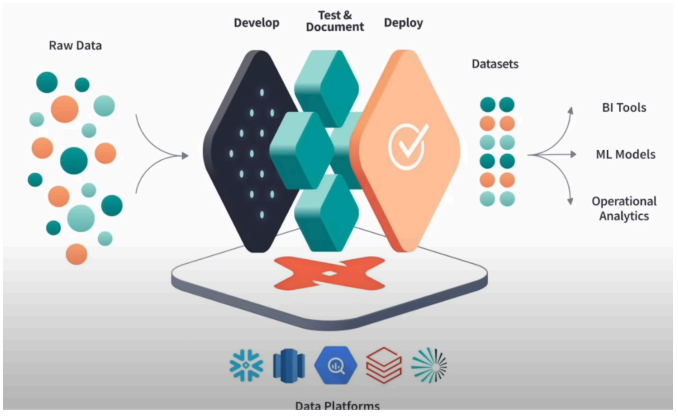


Tech-Stack Used:

1. AWS S3 - As Data Lake
2. Snowflake - Warehouse
3. dbt (Data Build Tool)- This is a transformation tool that allows us to write modular SQL queries to transform raw data into analytics-ready data models. It focuses mainly on the transformation step in the ELT process.

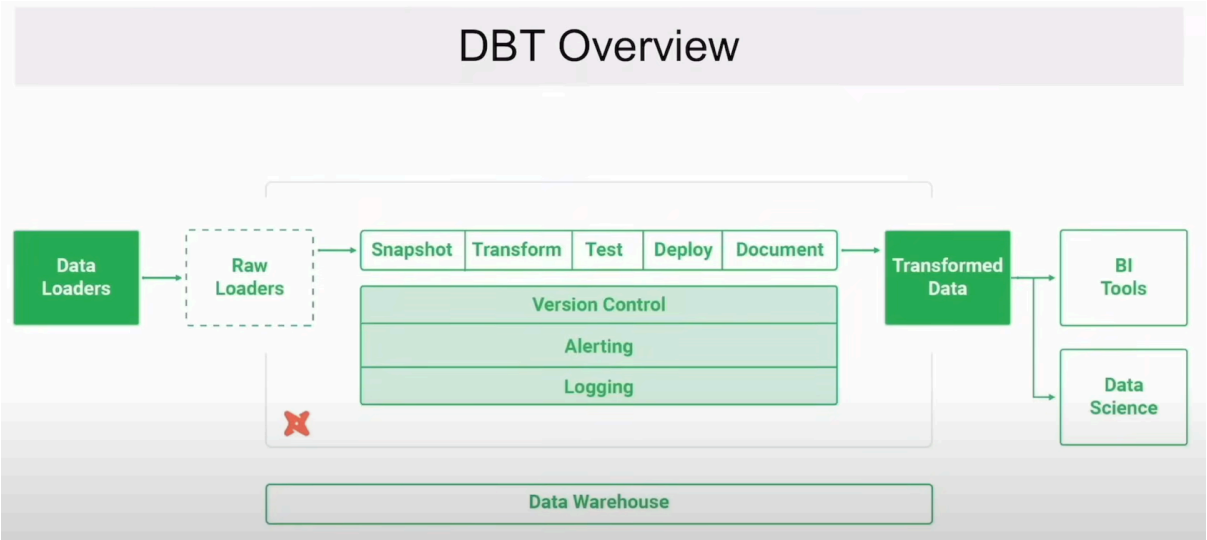
Layers Description:
Raw Landing Layer
Staging Layer - Copy of Raw (created in case we make any changes or loose data here we always have raw version available)
Dev Layer - We apply all transformation on staging data using dbt and push data in this layer

Some dbt concepts before actually starting implementing our pipeline:
High level overview of dbt:

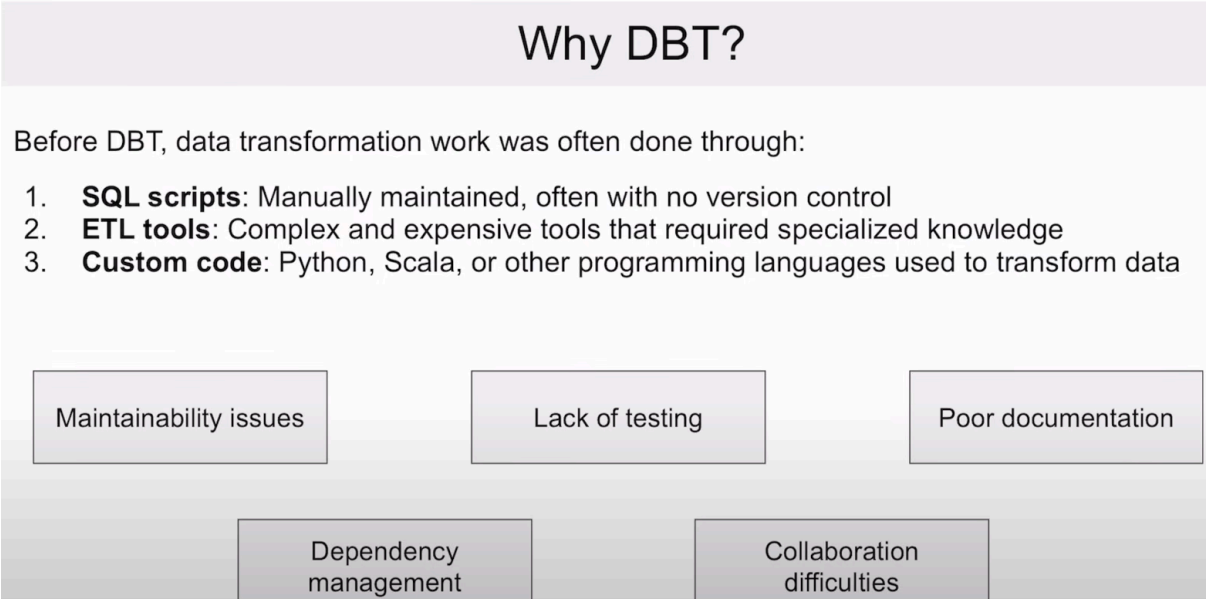


dbt sits on top of the data platforms on the serving side i.e. it will connect to these platforms and will perform the transformations on the data available. It enables us to develop, test, document, and deploy scalable data transformation pipelines.

Detailed Overview of individual components:



But Why DBT?



1. **Modularity**: Break down complex transformations into manageable pieces
2. **Reusability**: Create and reuse common SQL patterns across your project
3. **Testing framework:** Built-in testing capabilities to ensure data quality
4. **Documentation generation:** Automatically generate documentation for your data models
5. **Dependency management:** Automatically handle the order of model execution
6. **Development workflow:** Support for development environments separate from production

Development:

Useful Links:

- Dataset Source
- Dataset README

We will store the data first in AWS S3 bucket(in our project we will be using AWS localstack)

1. Install localstack , awscli, awscli-local
2. Run localstack using: localstack start (Ensure Docker is up & running)
3. Configure dummy credentials: (Run: aws configure)
   AWS Access Key ID [None]: test
   AWS Secret Access Key [None]: test
   Default region name [None]: us-east-1
   Default output format [None]: csv
4. Check for s3 buckets: aws --endpoint-url=http://localhost:4566 s3 ls
   (Initially no bucket). Port can be different for you.
5. Create a AWS S3 bucket: aws --endpoint-url=http://localhost:4566 s3 mb s3://netflixdataset
6. Create upload_to_s3.py file for uploading csv files from data source folder & run the same.

```
PS E:\Data_Engineering_Projects\Netflix_Data_Pipeline> python  upload_to_s3.py
File uploaded successfully: genome-scores.csv
File uploaded successfully: genome-tags.csv
File uploaded successfully: links.csv
File uploaded successfully: movies.csv
File uploaded successfully: ratings.csv
File uploaded successfully: tags.csv
```

7. Check if files uploaded onto s3 bucket using: aws --endpoint-url=http://localhost:4566 s3 ls s3://netflixdataset

```
PS E:\Data_Engineering_Projects\Netflix_Data_Pipeline> aws --endpoint-url=http://localhost:4566 s3 ls s3://
netflixdataset
2025-12-27 09:29:36  323544381 genome-scores.csv
2025-12-27 09:29:41      18103 genome-tags.csv
2025-12-27 09:29:41     570090 links.csv
2025-12-27 09:29:41    1397542 movies.csv
2025-12-27 09:29:41  533444411 ratings.csv
2025-12-27 09:29:48   16603996 tags.csv
```

8. Create Snowflake Objects - Warehouse, user, database, schema and grant the necessary permissions (Refer netflix.sql file)
9. In this project, since I don't have an AWS account, I will be loading data in snowflake stage from local rather than AWS S3, but if you have one you can load data directly from S3.
   /*create stage & load data from S3*/
   – We will need to create an IAM User before creating stage to have AWS Access Key & Secret Key

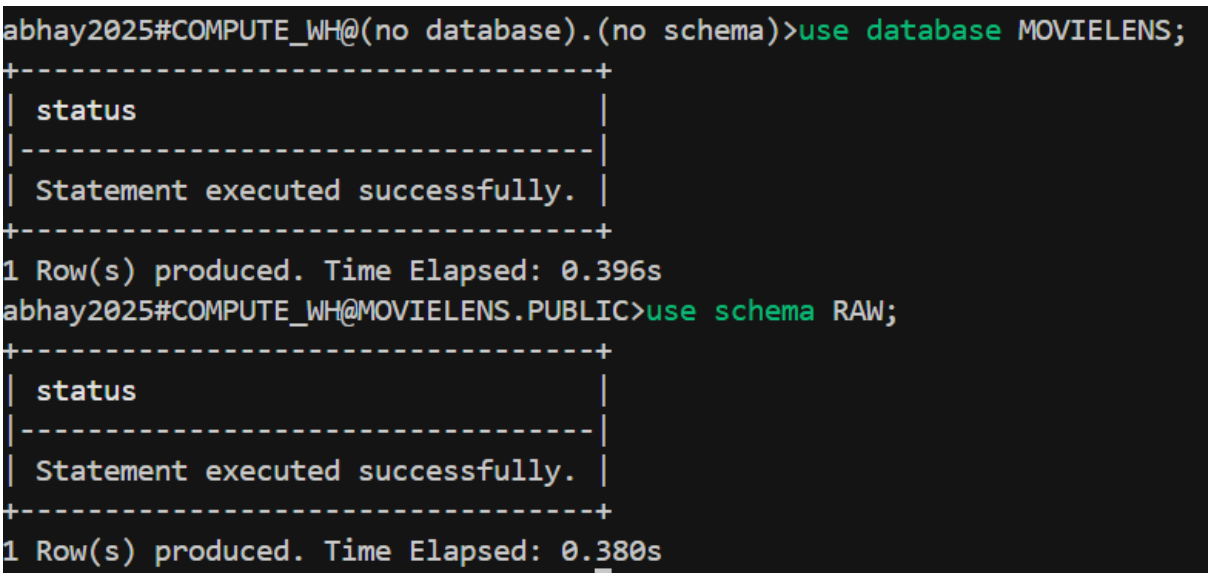
CREATE STAGE NETFLIX_STG
URL='s3://netflixdataset'
CREDENTIALS=(AWS_KEY_ID='<*your_access_key*>'
AWS_SECRET_KEY='<*your_secret_key*>')

#----------#
/*Loading data from local to snowflake stage*/
Install SnowSQL
From VS Code terminal:
snowsql -a <ACCOUNT> -u <USER>

– put all the 'csv' files from data folder
put file://E:/Data_Engineering_Projects/Netflix_Data_Pipeline/data/*.csv
@netflix_stage;

```
abhay2025#COMPUTE_WH@MOVIELENS.RAW>put file://E:/Data_Engineering_Projects/Netflix_Data_Pipeline/data/*.cs
                                  v @netflix_stage;
+------------------+--------------------+-------------+-------------+--------------------+--------------------+--
--------+---------+
| source           | target             | source_size | target_size | source_compression | target_compression | s
tatus   | message |
|------------------+--------------------+-------------+-------------+--------------------+--------------------+--
--------+---------|
| genome-scores.csv | genome-scores.csv.gz |   323544381 |    60441424 | NONE               | GZIP
     | UPLOADED |         |
| genome-tags.csv   | genome-tags.csv.gz   |       18103 |        8400 | NONE               | GZIP
     | UPLOADED |         |
| links.csv         | links.csv.gz         |      570090 |      246256 | NONE               | GZIP
     | UPLOADED |         |
| movies.csv        | movies.csv.gz        |     1397542 |      498848 | NONE               | GZIP
     | UPLOADED |         |
| ratings.csv       | ratings.csv.gz       |   533444411 |   132656096 | NONE               | GZIP               | U
PLOADED |         |
| tags.csv          | tags.csv.gz          |    16603996 |     4787920 | NONE               | GZIP               | U
PLOADED |         |
+------------------+--------------------+-------------+-------------+--------------------+--------------------+--
```

```
* SnowSQL * v1.4.5
Type SQL statements or !help
abhay2025#COMPUTE_WH@(no database).(no schema)>use role ACCOUNTADMIN;
+----------------------------------+
| status                           |
|----------------------------------|
| Statement executed successfully. |
+----------------------------------+
1 Row(s) produced. Time Elapsed: 0.355s
```



– Load data into Tables [Refer netflix.sql]
– Below is one example

```
CREATE OR REPLACE TABLE RAW_LINKS(
    movieId INTEGER,
    imdbId INTEGER,
    tmdbId INTEGER
);

COPY INTO RAW_LINKS
FROM '@netflix_stage/links.csv'
FILE_FORMAT=NETFLIX_FF;
```

10. DBT Setup:
    Create a new venv: *python -m venv netflix*
    Activate new venv: *netflix\Scripts\activate.bat* [Reverify your venv is activated]
    Install dbt-snowflake & dbt-core: *pip install dbt-snowflake*
    pip install dbt-core

11. DBT Execution:
    Inititalize: dbt init netflixdbt
    Check account name, username from Snowflake UI & login using the user we created in netflix.sql file
    Note: You can check profiles.yml file for more details of the account you have used to log in.
12. Create models and run dbt run cmd, you will see a staging view is created in snowflake. But why view only, not table?, since we declared view in dbt_project.yml

```
netflixdbt > dbt_project.yml
21    clean-targets:         # directories to be removed by `dbt clean`
23      - "dbt_packages"
24
25
26    # Configuring models
27    # Full documentation: https://docs.getdbt.com/docs/configuring-models
28
29    # In this example config, we tell dbt to build all models in the example/
30    # directory as views. These settings can be overridden in the individual model
31    # files using the `{{ config(...) }}` macro.
32    models:
33      netflixdbt:
34        # Config indicated by + and applies to all files under models/example/
35        example:
36          +materialized: view
37
```

Materialization:
Table will give a much faster result as it will actually store the data but will cost in storage and will take a long time to rebuild specially for complex transformations, on the other hand view will store the definition and will fetch data from src data each time a model is built.

13. So now we have data till our staging layer starts from raw. Now we will create Dimension & Fact Tables.
    Now this is the core of dbt we can reference other models (***using ref jinja function***) in dimension or can join different models to have transformed data.
    We will create dim_movies model that fetch data from stg_movies
    Also we have created schema DEV for dimensions and staging models now, by changing schema to DEV in profiles.yml file under .dbt folder.
14. Develop fact and dimension tables , also update the materialization of fact and dim as table, since default we set as views.
15. Incremental Fact Table Development (fct_ratings.sql):
    For this we will update the materialization of the stg_ratings model as a table just to test the incremental functionality. It will move it from views to tables in DEV schema.
    Run individual model: *dbt run -- model <model_name>*
    Now one option is to change the raw data and add a new row to check the functionality, but here we will manually add a new row in the stg_ratings table [Refer netflix_scd.sql file]
    After insertion we can check the stg_ratings will be updated but fct_ratings is still not updated, so we will run the fct_ratings model again. We can now see the fct_ratings table is updated itself.
16. **Materialization: ephemeral**
    We will create a dim_movies_with_tags model, when we run this model it will not create a table or view or incremental but will get executed in the backend, any other model can reference this (Refer- ep_movie_with_tags.sql fact model). You will not see ephemeral dim in snowflake but you will see ep_movie_with_tags fact table.
17. **Seeds**: Seeds are csv files that we can load in our DWH.Useful for static reference data like country codes, zip code or product categories.

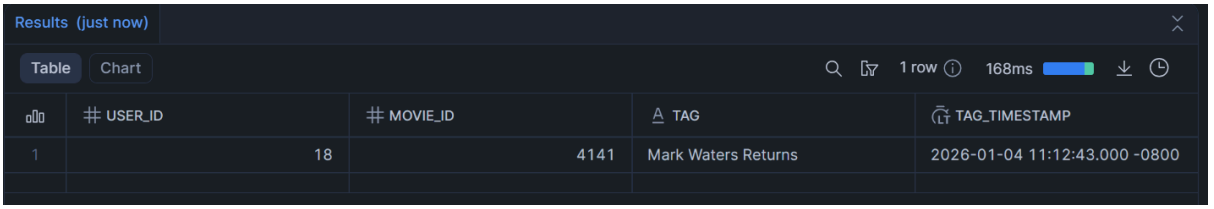
Now create a new csv file under /seeds folder as seed_movie_release_dates.csv(or download it from my repo)
Run cmd:> dbt seed
This will create a new table in snowflake with the same name as the csv file. We can reference this seed in other models.
We have created a /mart/mart_movie_releases.sql model that uses the above seed, it will create a new table in snowflake.
Sources: Sources in DBT represent the raw data loaded into your data warehouse. Sources are defined in YAML files. Typically, you'd create a file like models/sources.yml. It can be further used in models as used in stg_movies model.

18. **Snapshots**: Snapshots in DBT are a built-in implementation of Type 2 Slowly Changing Dimensions. They record the state of a mutable table over time by:

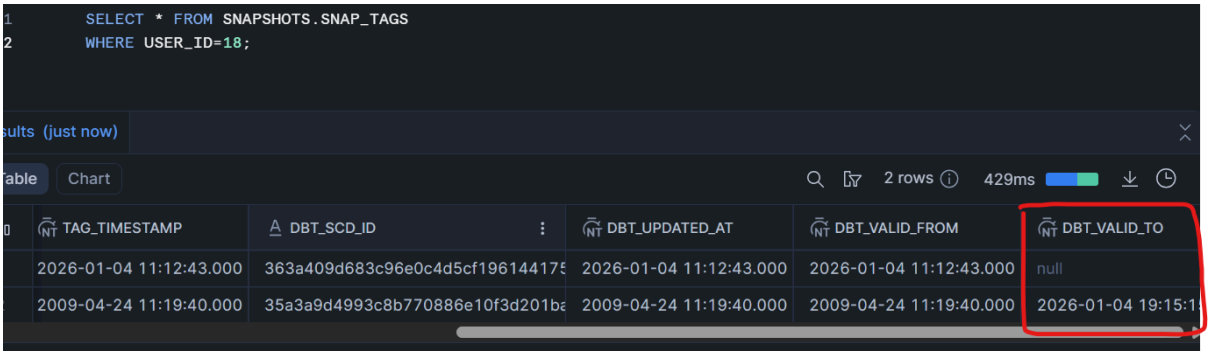
- Tracking changes to specified columns
- Adding effective dates (`dbt_valid_from` and `dbt_valid_to`)
- Maintaining a complete history of changes

We have created snp_tags snapshot with a timestamp strategy. We have also created a surrogate_key to uniquely identify rows.For that we used the dbt_utils package and created a packages.yml file.
Run cmd> dbt deps → it will install the package dbt-labs/dbt_utils (since we only defined this in packages.yml file)
Then run the snapshot , we will have a new schema created in snowflake as SNAPSHOT
Run all snapshots: dbt snapshot

Now we will test SCD2 in snap_tags
We will update a record in stg_tags [Refer netflix_scd.sql], but before that we will change the config of stg_tags to table & run the same.



Then we will again run dbt snapshot



We can see the earlier record expired and a new one is inserted which is active now.

19. Testing: Types of Tests

DBT supports two main types of tests:

1. Generic tests: Reusable tests that can be applied to multiple models
2. Singular tests: Custom SQL queries that test specific conditions

Generic Tests : Generic tests are reusable assertions about your models.Generic tests are typically defined in YAML files alongside your models. DBT comes with four built-in generic tests:

1. Unique: Tests that values in a column are unique
2. Not Null: Tests that a column contains no null values
3. Relationships: Tests referential integrity between tables
4. Accepted Values: Tests that values in a column are from a defined set

Singular Tests : Singular tests are custom SQL queries that return rows that fail the test. An empty result set mea

Creating a Singular Test : Create a file in the `tests` directory, e.g., `tests/order_amount_is_positive.sql`. Any records returned by this query represent test failures.

We will create a new schema.yml file defining generic tests for each model. And run below command:

>> dbt test

```
19:02:41  9 of 12 START test not_null_fct_ratings_user_id ..............................[RUN]
19:02:42  9 of 12 PASS not_null_fct_ratings_user_id ...................................[PASS in 0.43s]
19:02:42  10 of 12 START test relationships_fct_ratings_movie_id__movieid__ref_dim_movies_  [RUN]
19:02:44  10 of 12 PASS relationships_fct_ratings_movie_id__movieid__ref_dim_movies_ .....[PASS in 2.51s]
19:02:44  11 of 12 START test unique_dim_genome_tags_tag_id ..............................[RUN]
19:02:45  11 of 12 PASS unique_dim_genome_tags_tag_id ....................................[PASS in 0.91s]
19:02:45  12 of 12 START test unique_dim_users_user_id ....................................[RUN]
19:02:47  12 of 12 PASS unique_dim_users_user_id ..........................................[PASS in 1.42s]
19:02:48
19:02:48  Finished running 12 data tests in 0 hours 0 minutes and 18.24 seconds (18.24s).
19:02:48
19:02:48  Completed successfully
19:02:48
19:02:48  Done. PASS=12 WARN=0 ERROR=0 SKIP=0 NO-OP=0 TOTAL=12
```

Singular Test: We will create a new file under tests/relevance_score_test.sql and run dbt test again to test.

20. **Documentation**: Run cmd >> dbt docs generate

Now run >> dbt docs serve , It will start a web server in your local pc, where we will be able to see everything about the project.

21. **Macros**:Macros in DBT are reusable pieces of SQL code, similar to functions in other programming languages. They use Jinja, a templating language, to generate SQL dynamically.

Benefits of macros:

- Reduce code duplication
- Encapsulate complex logic
- Make your code more maintainable

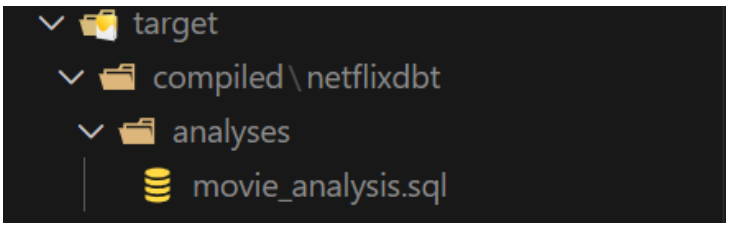
For example we wrote a Singular test to check relevance score , but what if we want to check this column in multiple tables. For that we will create a macro and pass the table info to the same & it will return the result.

We have created no_nulls_in_column macro. And will change the relevance score test file to call this macro for the fact table. And run>> dbt test

```
19:34:22  10 of 13 START test relationships_fct_ratings_movie_id__movieid__ref_dim_movies_  [RUN]
19:34:23  10 of 13 PASS relationships_fct_ratings_movie_id__movieid__ref_dim_movies_ ..... [PASS in 0.63s]
19:34:23  11 of 13 START test relevence_score_test ..................................... [RUN]
19:34:24  11 of 13 PASS relevence_score_test ........................................... [PASS in 1.62s]
19:34:24  12 of 13 START test unique_dim_genome_tags_tag_id ............................ [RUN]
19:34:25  12 of 13 PASS unique_dim_genome_tags_tag_id .................................. [PASS in 0.44s]
19:34:25  13 of 13 START test unique_dim_users_user_id .................................. [RUN]
19:34:26  13 of 13 PASS unique_dim_users_user_id ....................................... [PASS in 0.84s]
19:34:27
19:34:27  Finished running 13 data tests in 0 hours 0 minutes and 18.47 seconds (18.47s).
19:34:27
19:34:27  Completed successfully
19:34:27
19:34:27  Done. PASS=13 WARN=0 ERROR=0 SKIP=0 NO-OP=0 TOTAL=13
```

22. Analysis : We have created a sample analyses file (analyses/movie_analyses.sql) for adhoc-analyses. You can create few others if you want 🙂

Run cmd>> dbt compile



Above is the compiled version of the sql file. Execute this by copying this on snowflake and running the same you will get the data or you can create a table for the same.

Thanks for reading the documentation 🙂