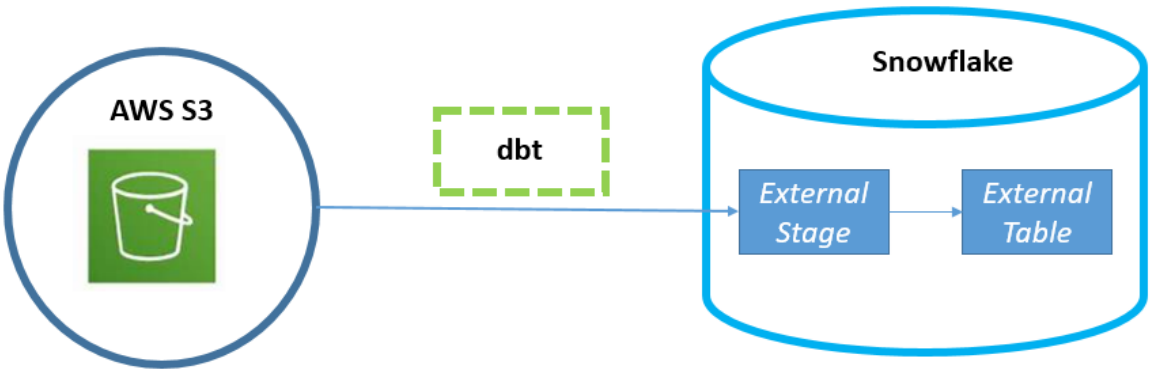
Snowflake with dbt(Data Build Tool): ETL Transformations (Installed via CLI)

*I designed and implemented an end-to-end ETL pipeline leveraging **AWS S3, Snowflake, and dbt (through CLI)** to build a scalable, structured data workflow. The project included orchestrating data ingestion from S3 into Snowflake, performing advanced common table expressions(**CTE's**) of **SQL** transformations using dbt, and managing clean, version-controlled development through GitHub. This hands-on build provided valuable practical insights into cloud-based data engineering and modern ELT practices, resulting in a robust pipeline aligned with industry standards.*



"How I built an end-to-end Snowflake + dbt pipeline from scratch 🚀"

1️⃣ Downloaded MovieLens CSV files locally.

2️⃣ Uploaded the CSVs to an S3 bucket in my AWS workspace.

3️⃣ Created an IAM role with access keys for Snowflake to securely pull data.

4️⃣ Set up Snowflake: created users, roles, warehouse, database, schema.

5️⃣ Pulled CSV data from S3 into Snowflake raw tables using CREATE STAGE and COPY INTO commands.

6️⃣ Installed dbt CLI (instead of dbt Cloud) and initialized a structured dbt project in VS Code.

7️⃣ Created **three folders:** staging, dim, fct for organized Medallion architecture.

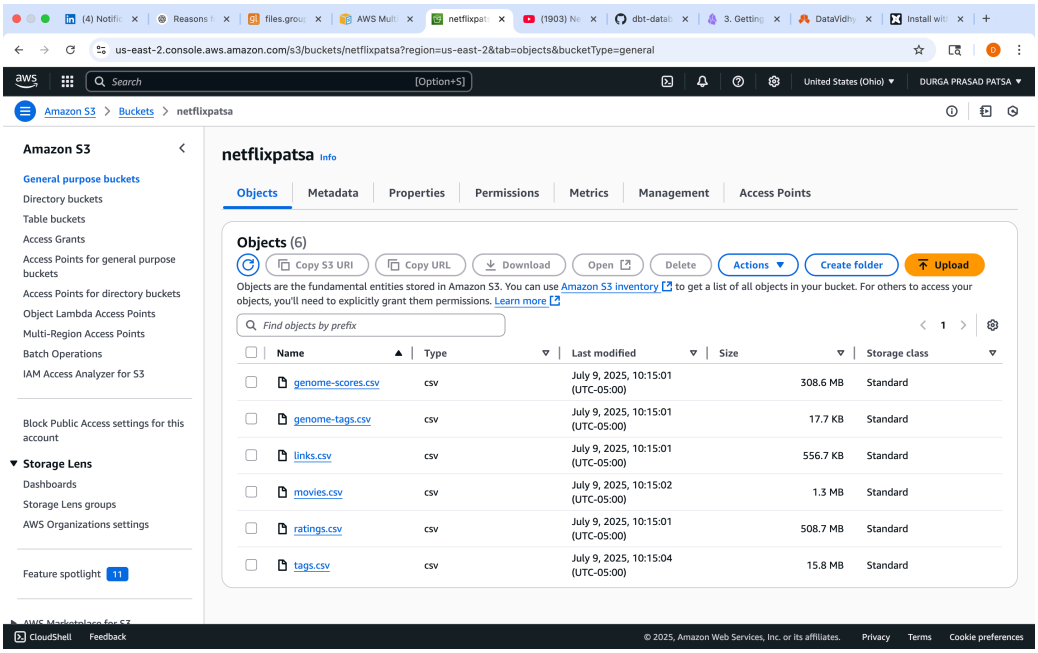
8️⃣ Built src_* models in staging as **views** from raw Snowflake tables.

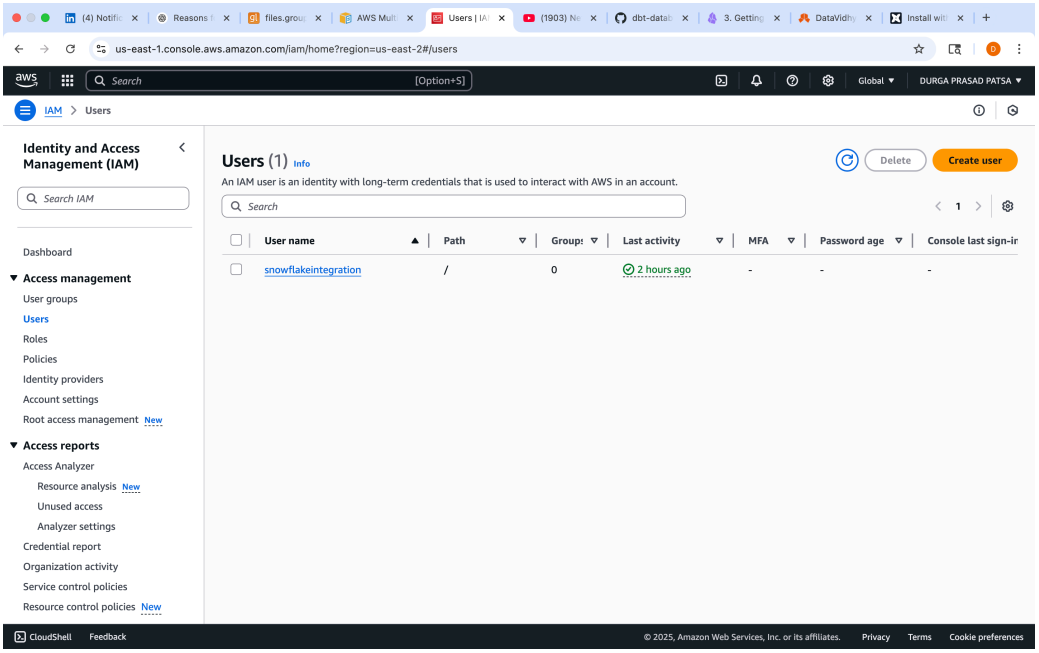
9️⃣ Transformed src_* views to dim_* tables with cleaning and structuring logic.

🔟 Further transformed dim_* tables to fct_* fact tables to create a clean, analytics-ready final table.

✅ Validated the pipeline with dbt run, dbt test, and explored the lineage in dbt docs.

S3 Work: Upload CSV files into AWS S3 and setup the IAM role and access.





1️⃣ Environment Setup in Snowflake

Create Warehouse, Database, Schema

```
USE ROLE ACCOUNTADMIN;
CREATE ROLE IF NOT EXISTS TRANSFORM;
GRANT ROLE TRANSFORM TO ROLE ACCOUNTADMIN;

CREATE WAREHOUSE IF NOT EXISTS COMPUTE_WH;
GRANT OPERATE ON WAREHOUSE COMPUTE_WH TO ROLE TRANSFORM;

CREATE DATABASE IF NOT EXISTS MOVIELENS;
CREATE SCHEMA IF NOT EXISTS MOVIELENS.RAW;

GRANT ALL ON WAREHOUSE COMPUTE_WH TO ROLE TRANSFORM;
GRANT ALL ON DATABASE MOVIELENS TO ROLE TRANSFORM;
GRANT ALL ON ALL SCHEMAS IN DATABASE MOVIELENS TO ROLE TRANSFORM;
```

Create User for dbt

```
CREATE USER IF NOT EXISTS dbt PASSWORD = '****'
DEFAULT_WAREHOUSE = 'COMPUTE_WH'
DEFAULT_ROLE = TRANSFORM
DEFAULT_NAMESPACE = 'MOVIELENS.RAW';

GRANT ROLE TRANSFORM TO USER dbt;
```

Create External Stage for S3 Ingestion

```
CREATE STAGE netflixstage
URL = 's3://netflixpatsa'
CREDENTIALS = (AWS_KEY_ID='***' AWS_SECRET_KEY='***');
```

2️⃣ Ingesting CSV Data from S3 into Snowflake Staging Tables

Create staging tables and load data:

#Creating raw movies table and copying data from S3 location

```
CREATE OR REPLACE TABLE raw_movies (

    movieId INTEGER,
    title STRING,
    genres STRING
);
COPY INTO raw_movies FROM '@netflixstage/movies.csv' FILE_FORMAT = (TYPE =
'CSV' SKIP_HEADER = 1 FIELD_OPTIONALLY_ENCLOSED_BY = '"');
```

Creating raw ratings table and copying data from S3 location

```
CREATE OR REPLACE TABLE raw_ratings (
    userId INTEGER,
    movieId INTEGER,
    rating FLOAT,
    timestamp BIGINT
);
COPY INTO raw_ratings FROM '@netflixstage/ratings.csv' FILE_FORMAT = (TYPE =
'CSV' SKIP_HEADER = 1 FIELD_OPTIONALLY_ENCLOSED_BY = '"');
```

Creating raw movies tags and copying data from S3 location

```
CREATE OR REPLACE TABLE raw_tags (
    userId INTEGER,
    movieId INTEGER,
    tag STRING,
    timestamp BIGINT
);
COPY INTO raw_tags FROM '@netflixstage/tags.csv' FILE_FORMAT = (TYPE = 'CSV'
SKIP_HEADER = 1 FIELD_OPTIONALLY_ENCLOSED_BY = '"') ON_ERROR = 'CONTINUE';
```

Creating raw genome table and copying data from S3 location

```
CREATE OR REPLACE TABLE raw_genome_tags (
    tagId INTEGER,
    tag STRING
);
COPY INTO raw_genome_tags FROM '@netflixstage/genome-tags.csv' FILE_FORMAT =
(TYPE = 'CSV' SKIP_HEADER = 1 FIELD_OPTIONALLY_ENCLOSED_BY = '"');
```

Creating raw genome scores table and copying data from S3 location

```
CREATE OR REPLACE TABLE raw_genome_scores (
    movieId INTEGER,
    tagId INTEGER,
    relevance FLOAT
);
COPY INTO raw_genome_scores FROM '@netflixstage/genome-scores.csv'
FILE_FORMAT = (TYPE = 'CSV' SKIP_HEADER = 1 FIELD_OPTIONALLY_ENCLOSED_BY =
'"');
```

Creating raw links table and copying data from S3 location

```
CREATE OR REPLACE TABLE raw_links (
    movieId INTEGER,
    imdbId INTEGER,
    tmdbId INTEGER
);
COPY INTO raw_links FROM '@netflixstage/links.csv' FILE_FORMAT = (TYPE =
'CSV' SKIP_HEADER = 1 FIELD_OPTIONALLY_ENCLOSED_BY = '"');
```

✅ Validate row counts for each table to confirm ingestion.

3️⃣ Setting Up dbt Locally

Install dbt using pipx:

```
pipx install dbt-snowflake
pipx ensurepath
```

Add ~/.local/bin to your PATH in ~/.zshrc:

```
export PATH="$PATH:/Users/your_username/.local/bin"
source ~/.zshrc
```

Initialize the dbt Project

```
dbt init my_dbt_project
```

Configure ~/.dbt/profiles.yml:

```
my_dbt_project:
  target: dev
  outputs:
    dev:
      type: snowflake
```

```
      account: your_account
      user: dbt
      password: your_password
      role: TRANSFORM
      database: MOVIELENS
      warehouse: COMPUTE_WH
      schema: RAW
      threads: 1
```

✅ Validate with:

```
dbt debug
```

4️⃣ Create dbt_project.yml

```yaml
# Name your project! Project names should contain only lowercase characters
# and underscores. A good package name should reflect your organization's
# name or the intended use of these models
name: 'my_dbt_project'
version: '1.0.0'

# This setting configures which "profile" dbt uses for this project.
profile: 'my_dbt_project'

# These configurations specify where dbt should look for different types of files.
# The `model-paths` config, for example, states that models in this project can be
# found in the "models/" directory. You probably won't need to change these!
model-paths: ["models"]
analysis-paths: ["analyses"]
test-paths: ["tests"]
seed-paths: ["seeds"]
macro-paths: ["macros"]
snapshot-paths: ["snapshots"]

clean-targets:         # directories to be removed by `dbt clean`
  - "target"
  - "dbt_packages"


# Configuring models
# Full documentation: https://docs.getdbt.com/docs/configuring-models

# In this example config, we tell dbt to build all models in the example/
# directory as views. These settings can be overridden in the individual model
# files using the `{{ config(...) }}` macro.
models:
```

```
  my_dbt_project:
    +materialized: view
    dim:
      +materialized: table
    fct:
      +materialized: table
```

5️⃣ Building the dbt Pipeline

Create and populate models/staging/ with src_ models (views):

- src_movies.sql, src_ratings.sql, src_tags.sql, src_genome_tags.sql, src_genome_scores.sql, src_links.sql referencing MOVIELENS.RAW tables.

```
{{ config(materialized='view') }}

WITH raw_movies AS (
    SELECT * FROM movielens.raw.raw_movies
)
SELECT
    movieId as movie_id,
    title,
    generes
FROM raw_moviesdbt
```

```
WITH raw_ratings AS (
    SELECT * FROM MOVIELENS.RAW.raw_ratings
)

SELECT
    userId AS user_id,
    movieId AS movie_id,
    rating,
    TO_TIMESTAMP_LTZ(timestamp) AS rating_timestamp
FROM raw_ratings
```

```
WITH raw_tags AS (
    SELECT * FROM MOVIELENS.RAW.raw_tags
)

SELECT
    userId AS user_id,
```

```
    movieId AS movie_id,
    tag,
    TO_TIMESTAMP_LTZ(timestamp) AS tag_timestamp
FROM raw_tags
```

```
WITH raw_links AS (
    SELECT * FROM MOVIELENS.RAW.raw_links
)

SELECT
    movieId AS movie_id,
    imdbId AS imdb_id,
    tmdbId AS tmdb_id
FROM raw_links
```

```
WITH raw_genome_tags AS (
    SELECT * FROM MOVIELENS.RAW.raw_genome_tags
)

SELECT
    tagId AS tag_id,
    tag
FROM raw_genome_tags
```

✅ Run:

```
dbt run
```

✅ Validate creation of SRC_* views.

Create and populate models/dim/ with dim_ models (tables):

Example dim_movies.sql:

```
{{ config(materialized='table') }}

WITH src_movies AS (
```

```
    SELECT * FROM {{ ref('src_movies') }}
)
SELECT
    movie_id,
    INITCAP(TRIM(title)) AS movie_title,
    SPLIT(genres, '|') AS genre_array,
    genres
FROM src_movies
```

✅ Repeat similarly
for dim_ratings.sql, dim_tags.sql, dim_links.sql, dim_genome_tags.sql, dim_genome_scores.sql using {{ ref('src_*') }}.

✅ Run:

```
dbt run
```

✅ Validate creation of DIM_* tables.

Create and populate models/fct/ with fct_ models (fact tables):

Example fct_user_activity.sql:

```
{{ config(materialized='table') }}

WITH ratings AS (
    SELECT DISTINCT user_id FROM {{ ref('dim_ratings') }}
),
tags AS (
    SELECT DISTINCT user_id FROM {{ ref('dim_tags') }}
)
SELECT DISTINCT user_id
FROM (
    SELECT * FROM ratings
    UNION
    SELECT * FROM tags
)
)
```

✅ Run:

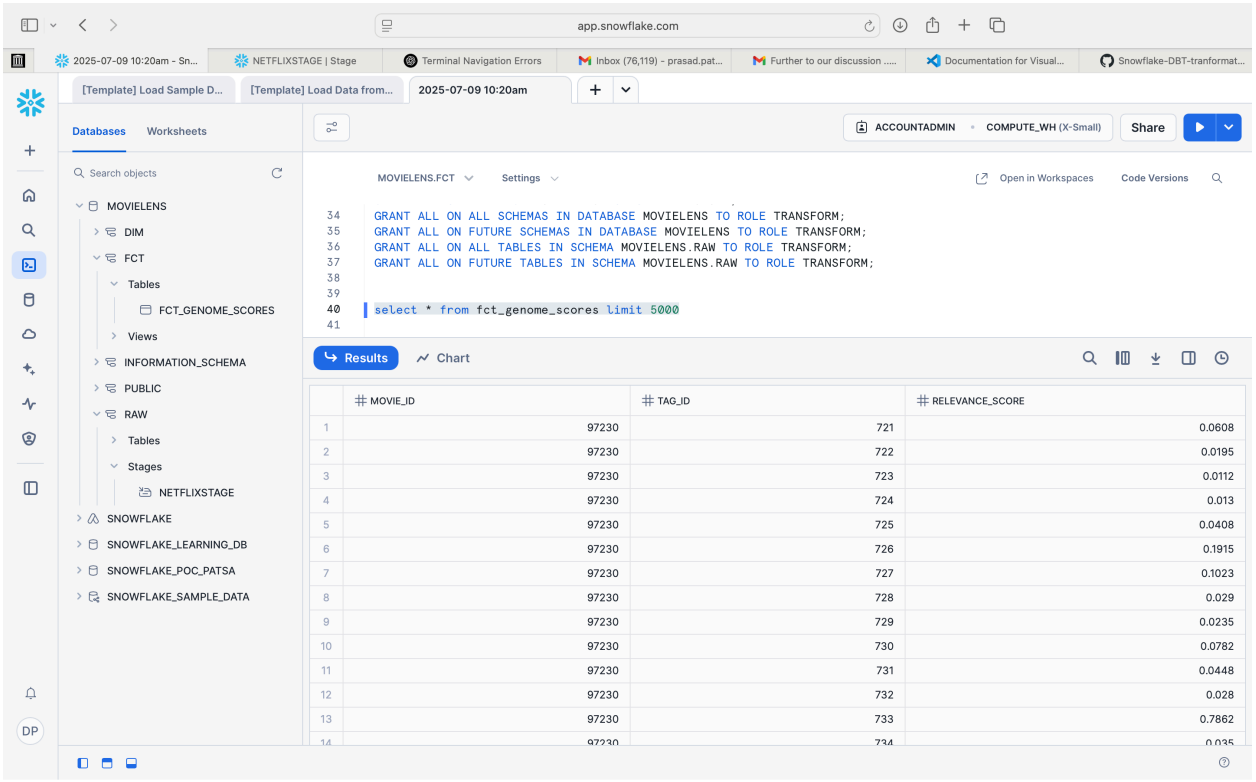
```
dbt run
```

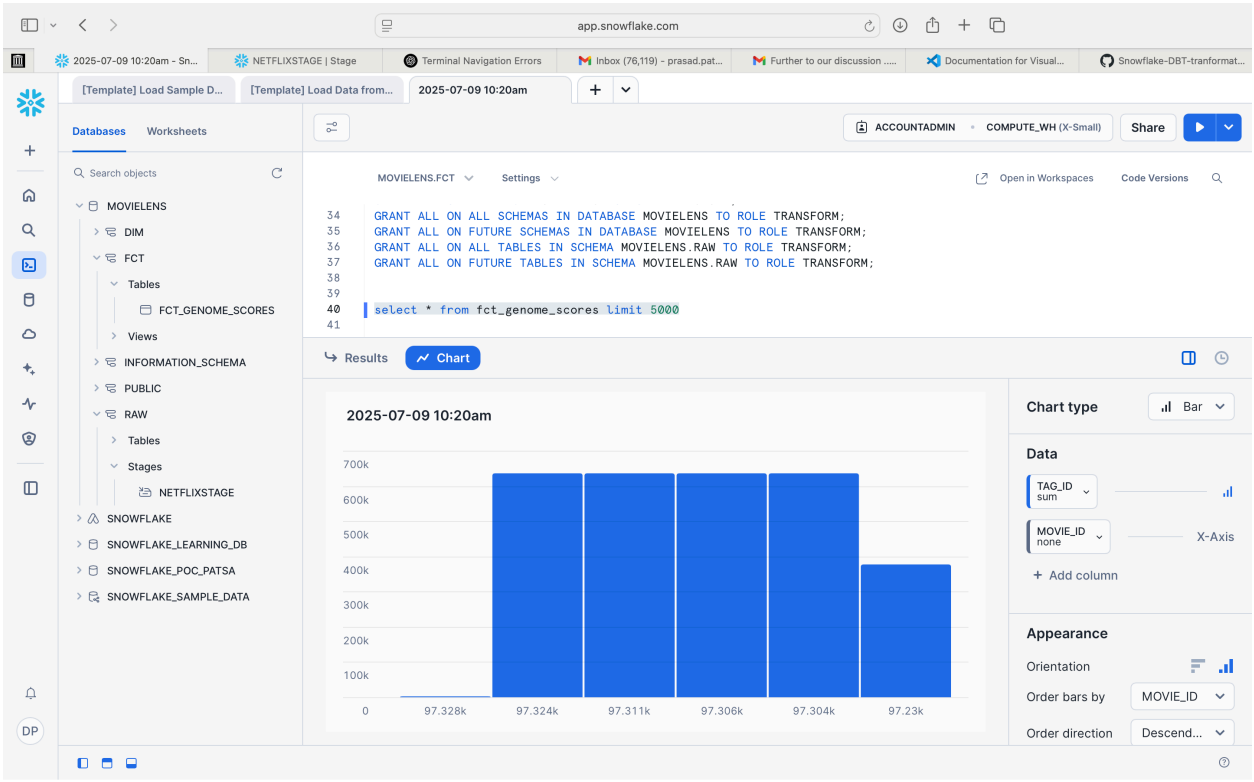
✅ Validate creation of FCT_* fact tables.

6️⃣ Testing and Documentation

✅ Create tests in tests/ for not_null and unique constraints on primary keys.
✅ Run:

```
dbt test
```





✅ Generate documentation and explore lineage:

```
dbt docs generate
dbt docs serve

Catalog written to
/Users/durgapatsa/Documents/my_dbt_project/target/catalog.json
durgapatsa@MacBookPro my_dbt_project %
```


```
{"metadata": {"dbt_schema_version": "https://schemas.getdbt.com/dbt/catalog/v1.json",
"dbt_version": "1.10.3", "generated_at": "2025-07-09T22:32:57.109209Z",
"invocation_id": "64ac51d6-f813-4ef0-80af-20ef36aade1c", "invocation_started_at":
"2025-07-09T22:32:35.173983Z", "env": {}}, "nodes": {"model.my_dbt_project.src_tags":
{"metadata": {"type": "VIEW", "schema": "FCT", "name": "SRC_TAGS", "database":
"MOVIELENS", "comment": null, "owner": "TRANSFORM"}, "columns": {"USER_ID": {"type":
"NUMBER", "index": 1, "name": "USER_ID", "comment": null}, "MOVIE_ID": {"type":
"NUMBER", "index": 2, "name": "MOVIE_ID", "comment": null}, "TAG": {"type": "TEXT",
"index": 3, "name": "TAG", "comment": null}, "TAG_TIMESTAMP": {"type":
"TIMESTAMP_LTZ", "index": 4, "name": "TAG_TIMESTAMP", "comment": null}}, "stats":
{"has_stats": {"id": "has_stats", "label": "Has Stats?", "value": false, "include":
false, "description": "Indicates whether there are statistics for this table"}},
"unique_id": "model.my_dbt_project.src_tags"},
"model.my_dbt_project.fct_genome_scores": {"metadata": {"type": "BASE TABLE",
"schema": "FCT", "name": "FCT_GENOME_SCORES", "database": "MOVIELENS", "comment":
null, "owner": "TRANSFORM"}, "columns": {"MOVIE_ID": {"type": "NUMBER", "index": 1,
```

```
"name": "MOVIE_ID", "comment": null}, "TAG_ID": {"type": "NUMBER", "index": 2, "name":
"TAG_ID", "comment": null}, "RELEVANCE_SCORE": {"type": "FLOAT", "index": 3, "name":
"RELEVANCE_SCORE", "comment": null}}, "stats": {"last_modified": {"id":
"last_modified", "label": "Last Modified", "value": "2025-07-09 21:34UTC", "include":
true, "description": "The timestamp for last update/change"}, "bytes": {"id": "bytes",
"label": "Approximate Size", "value": 46086144, "include": true, "description":
"Approximate size of the table as reported by Snowflake"}, "row_count": {"id":
"row_count", "label": "Row Count", "value": 11709768, "include": true, "description":
"An approximate count of rows in this table"}, "has_stats": {"id": "has_stats",
"label": "Has Stats?", "value": true, "include": false, "description": "Indicates
whether there are statistics for this table"}}, "unique_id":
"model.my_dbt_project.fct_genome_scores"}, "model.my_dbt_project.src_genome_score":
{"metadata": {"type": "VIEW", "schema": "FCT", "name": "SRC_GENOME_SCORE", "database":
"MOVIELENS", "comment": null, "owner": "TRANSFORM"}, "columns": {"MOVIE_ID": {"type":
"NUMBER", "index": 1, "name": "MOVIE_ID", "comment": null}, "TAG_ID": {"type":
"NUMBER", "index": 2, "name": "TAG_ID", "comment": null}, "RELEVANCE": {"type":
"FLOAT", "index": 3, "name": "RELEVANCE", "comment": null}}, "stats": {"has_stats":
{"id": "has_stats", "label": "Has Stats?", "value": false, "include": false,
"description": "Indicates whether there are statistics for this table"}}, "unique_id":
"model.my_dbt_project.src_genome_score"}, "model.my_dbt_project.src_movies":
{"metadata": {"type": "VIEW", "schema": "FCT", "name": "SRC_MOVIES", "database":
"MOVIELENS", "comment": null, "owner": "TRANSFORM"}, "columns": {"MOVIE_ID": {"type":
"NUMBER", "index": 1, "name": "MOVIE_ID", "comment": null}, "TITLE": {"type": "TEXT",
"index": 2, "name": "TITLE", "comment": null}, "GENERES": {"type": "TEXT", "index": 3,
"name": "GENERES", "comment": null}}, "stats": {"has_stats": {"id": "has_stats",
"label": "Has Stats?", "value": false, "include": false, "description": "Indicates
whether there are statistics for this table"}}, "unique_id":
"model.my_dbt_project.src_movies"}, "model.my_dbt_project.src_links": {"metadata":
{"type": "VIEW", "schema": "FCT", "name": "SRC_LINKS", "database": "MOVIELENS",
"comment": null, "owner": "TRANSFORM"}, "columns": {"MOVIE_ID": {"type": "NUMBER",
"index": 1, "name": "MOVIE_ID", "comment": null}, "IMDB_ID": {"type": "NUMBER",
"index": 2, "name": "IMDB_ID", "comment": null}, "TMDB_ID": {"type": "NUMBER",
"index": 3, "name": "TMDB_ID", "comment": null}}, "stats": {"has_stats": {"id":
"has_stats", "label": "Has Stats?", "value": false, "include": false, "description":
"Indicates whether there are statistics for this table"}}, "unique_id":
"model.my_dbt_project.src_links"}, "model.my_dbt_project.src_ratings": {"metadata":
{"type": "VIEW", "schema": "FCT", "name": "SRC_RATINGS", "database": "MOVIELENS",
"comment": null, "owner": "TRANSFORM"}, "columns": {"USER_ID": {"type": "NUMBER",
"index": 1, "name": "USER_ID", "comment": null}, "MOVIE_ID": {"type": "NUMBER",
"index": 2, "name": "MOVIE_ID", "comment": null}, "RATING": {"type": "FLOAT", "index":
3, "name": "RATING", "comment": null}, "RATING_TIMESTAMP": {"type": "TIMESTAMP_LTZ",
"index": 4, "name": "RATING_TIMESTAMP", "comment": null}}, "stats": {"has_stats":
{"id": "has_stats", "label": "Has Stats?", "value": false, "include": false,
"description": "Indicates whether there are statistics for this table"}}, "unique_id":
"model.my_dbt_project.src_ratings"}, "model.my_dbt_project.src_genome_tags":
{"metadata": {"type": "VIEW", "schema": "FCT", "name": "SRC_GENOME_TAGS", "database":
"MOVIELENS", "comment": null, "owner": "TRANSFORM"}, "columns": {"TAG_ID": {"type":
"NUMBER", "index": 1, "name": "TAG_ID", "comment": null}, "TAG": {"type": "TEXT",
```

```
"index": 2, "name": "TAG", "comment": null}}, "stats": {"has_stats": {"id":
"has_stats", "label": "Has Stats?", "value": false, "include": false, "description":
"Indicates whether there are statistics for this table"}}, "unique_id":
"model.my_dbt_project.src_genome_tags"}}, "sources": {}, "errors": null}
```

✅ Final Verification Checklist:

✅ Snowflake configured with raw data ingested from S3.
✅ dbt environment installed, debugged, and connected.
✅ staging models (SRC_*) created as views.
✅ dim models (DIM_*) created as tables.
✅ fct models (FCT_*) created as tables.
✅ Tests passing.
✅ Lineage documented and visualized.
✅ Ready for ML and dashboard integrations.

🚀 Next Steps

- Create Power BI / Streamlit dashboards using FCT_* tables.
- Integrate GitHub Actions for CI/CD on your dbt project.
- Extend with incremental models for large-scale data pipelines.

This structured end-to-end pipeline prepares you to **showcase your Snowflake + dbt skills confidently for your Data Architect or Senior Data Engineer roles** while reinforcing systematic architecture practices for your real-world learning.