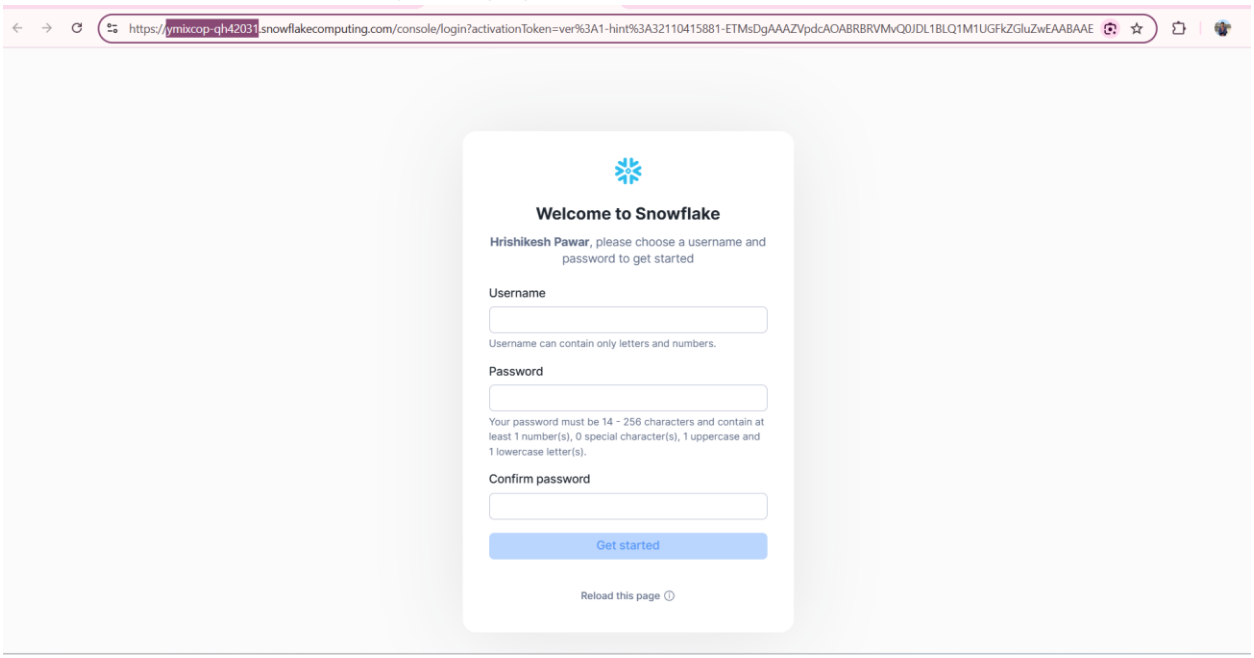
Activated snowflake account  - ymixcop-qh42031




Used this streamlit app to import Airbnb data - https://dbt-data-importer.streamlit.app/

email).

***This is not your Snowflake username**, but the first part of the snowflake url you received in your snowflake registration email:*

ymixcop-qh42031

Snowflake username (change this is you didn't set it to `admin` at registration):

dbtlearner

Snowflake Password:

••••••••••••••••

Start Setup

✓ Connecting to Snowflake

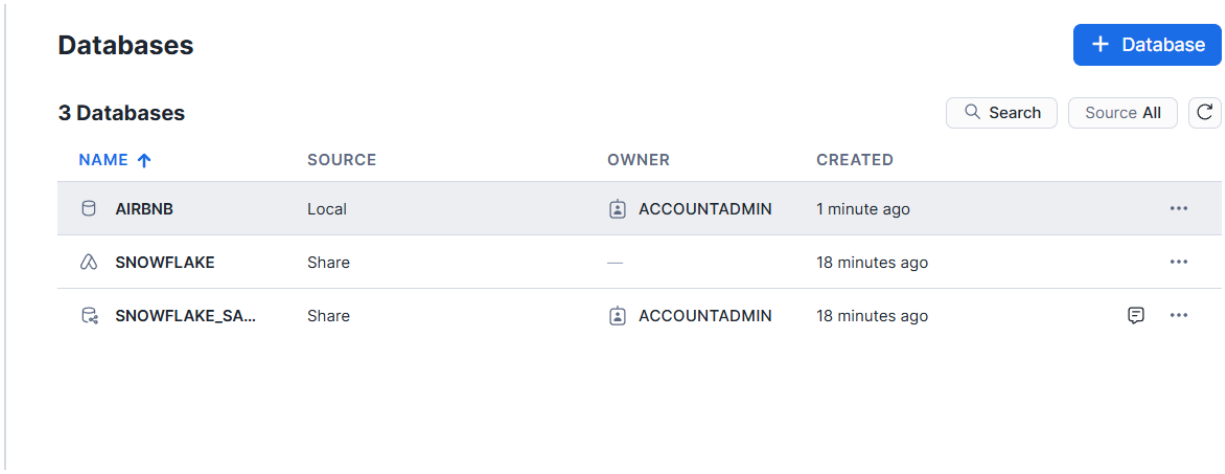
Connected to Snowflake successfully!

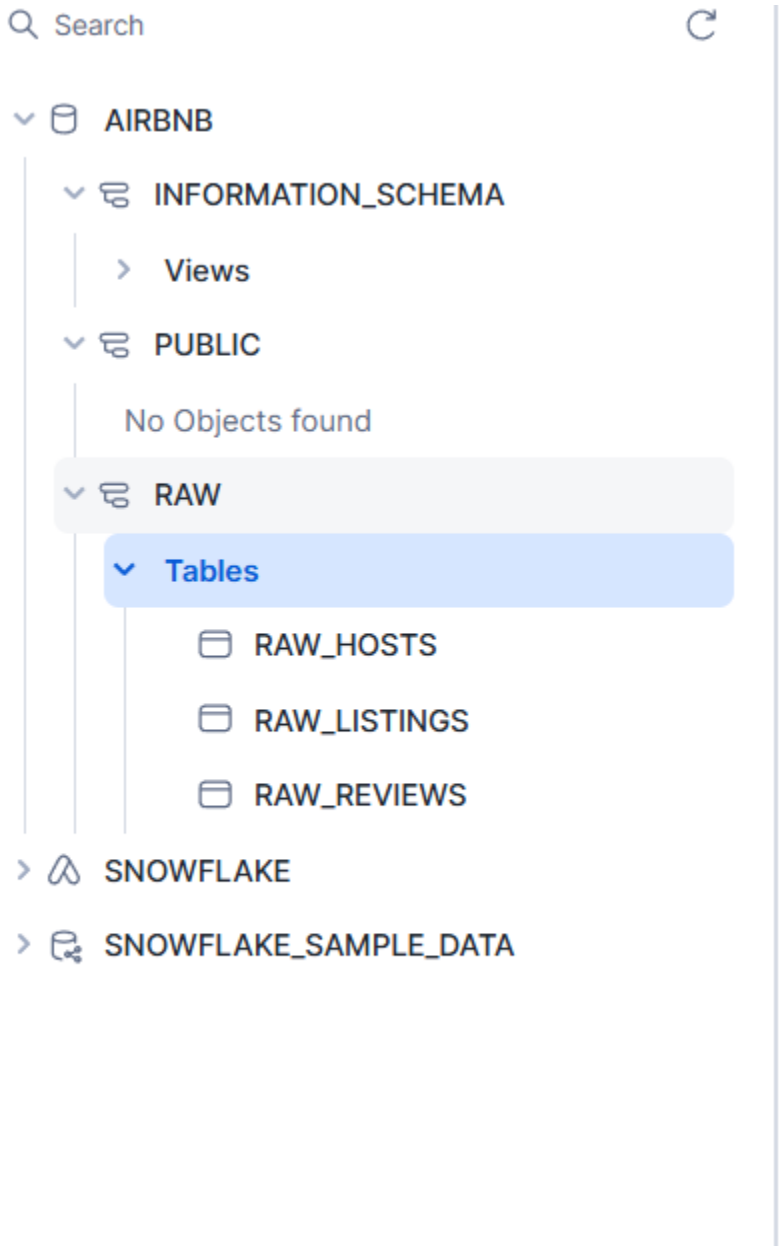
✓ Setting up the dbt User an Roles

✓ Importing Raw Tables

✓ Creating Reporter Role

Got Airbnb as database -






If not use this import tool then you can setup the database manually in snowflake by this - also these commands are executed in sql worksheet in snowflake

## Snowflake user creation

Copy these SQL statements into a Snowflake Worksheet, select all and execute them (i.e. pressing the play button).

If you see a *Grant partially executed: privileges [REFERENCE_USAGE] not granted.* message when you execute `GRANT ALL ON DATABASE AIRBNB to ROLE transform`, that's just an info message and you can ignore it.

```
-- Use an admin role
USE ROLE ACCOUNTADMIN;

-- Create the `transform` role
CREATE ROLE IF NOT EXISTS TRANSFORM;
GRANT ROLE TRANSFORM TO ROLE ACCOUNTADMIN;

-- Create the default warehouse if necessary
CREATE WAREHOUSE IF NOT EXISTS COMPUTE_WH;
GRANT OPERATE ON WAREHOUSE COMPUTE_WH TO ROLE TRANSFORM;

-- Create the `dbt` user and assign to role
CREATE USER IF NOT EXISTS dbt
  PASSWORD='dbtPassword123'
  LOGIN_NAME='dbt'
  MUST_CHANGE_PASSWORD=FALSE
  DEFAULT_WAREHOUSE='COMPUTE_WH'
  DEFAULT_ROLE=TRANSFORM
  DEFAULT_NAMESPACE='AIRBNB.RAW'
  COMMENT='DBT user used for data transformation';
GRANT ROLE TRANSFORM to USER dbt;

-- Create our database and schemas
CREATE DATABASE IF NOT EXISTS AIRBNB;
CREATE SCHEMA IF NOT EXISTS AIRBNB.RAW;

-- Set up permissions to role `transform`
GRANT ALL ON WAREHOUSE COMPUTE_WH TO ROLE TRANSFORM;
GRANT ALL ON DATABASE AIRBNB to ROLE TRANSFORM;
GRANT ALL ON ALL SCHEMAS IN DATABASE AIRBNB to ROLE TRANSFORM;
GRANT ALL ON FUTURE SCHEMAS IN DATABASE AIRBNB to ROLE TRANSFORM;
GRANT ALL ON ALL TABLES IN SCHEMA AIRBNB.RAW to ROLE TRANSFORM;
GRANT ALL ON FUTURE TABLES IN SCHEMA AIRBNB.RAW to ROLE TRANSFORM;
```

---

and then import the data manually in snowflake through s3 -

## Snowflake data import

Copy these SQL statements into a Snowflake Worksheet, select all and execute them (i.e. pressing the play button).

```
-- Set up the defaults
USE WAREHOUSE COMPUTE_WH;
USE DATABASE airbnb;
USE SCHEMA RAW;

-- Create our three tables and import the data from S3
CREATE OR REPLACE TABLE raw_listings
                    (id integer,
                     listing_url string,
                     name string,
                     room_type string,
                     minimum_nights integer,
                     host_id integer,
                     price string,
                     created_at datetime,
                     updated_at datetime);

COPY INTO raw_listings (id,
                        listing_url,
                        name,
                        room_type,
                        minimum_nights,
                        host_id,
                        price,
                        created_at,
                        updated_at)
                    from 's3://dbtlearn/listings.csv'
                     FILE_FORMAT = (type = 'CSV' skip_header = 1
                     FIELD_OPTIONALLY_ENCLOSED_BY = '"');


CREATE OR REPLACE TABLE raw_reviews
                    (listing_id integer,
                     date datetime,
                     reviewer_name string,
                     comments string,
                     sentiment string);

COPY INTO raw_reviews (listing_id, date, reviewer_name, comments, sentiment)
                    from 's3://dbtlearn/reviews.csv'
                     FILE_FORMAT = (type = 'CSV' skip_header = 1
                     FIELD_OPTIONALLY_ENCLOSED_BY = '"');


CREATE OR REPLACE TABLE raw_hosts
                    (id integer,
                     name string,
                     is_superhost string,
                     created_at datetime,
                     updated_at datetime);

COPY INTO raw_hosts (id, name, is_superhost, created_at, updated_at)
                    from 's3://dbtlearn/hosts.csv'
                     FILE_FORMAT = (type = 'CSV' skip_header = 1
                     FIELD_OPTIONALLY_ENCLOSED_BY = '"');
```

Setting up the virtual env -

```
Setting up python3.11-venv (3.11.0~rc1-1~22.04) ...
hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy$ python3.11 -m venv dbt_venv
hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy$ source dbt_venv/bin/activate
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy$
```

```
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy$ python --version
Python 3.11.0rc1
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy$ pip install dbt-snowflake==1.9.0
```

Now created a dbt profile – dbtlearn and this profile will store connection credentials to snowflake.

```
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy$ dbt init dbtlearn
04:50:24  Running with dbt=1.9.3
04:50:25  [ConfigFolderDirectory]: Unable to parse logging event dictionary. Failed to parse dir field: expected string
or bytes-like object, got 'PosixPath'.. Dictionary: {'dir': PosixPath('/home/hrishi/.dbt')}
04:50:25  Creating dbt configuration folder at
04:50:25
Your new dbt project "dbtlearn" was created!

For more information on how to configure the profiles.yml file,
please consult the dbt documentation here:

  https://docs.getdbt.com/docs/configure-your-profile

One more thing:

Need help? Don't hesitate to reach out to us via GitHub issues or on Slack:

  https://community.getdbt.com/

Happy modeling!

04:50:25  Setting up your profile.
Which database would you like to use?
[1] snowflake

(Don't see the one you want? https://docs.getdbt.com/docs/available-adapters)

Enter a number: 1
```

Now I'll have to setup this profile as below -

```
Your new dbt project "dbtlearn" was created!

For more information on how to configure the profiles.yml file,
please consult the dbt documentation here:

  https://docs.getdbt.com/docs/configure-your-profile

One more thing:

Need help? Don't hesitate to reach out to us via GitHub issues or on Slack:

  https://community.getdbt.com/

Happy modeling!

04:50:25  Setting up your profile.
Which database would you like to use?
[1] snowflake

(Don't see the one you want? https://docs.getdbt.com/docs/available-adapters)

Enter a number: 1
account (https://<this_value>.snowflakecomputing.com): ymixcop-qh42031
user (dev username): dbtlearner
[1] password
[2] keypair
[3] sso
Desired authentication type option (enter a number): 1
password (dev password):
role (dev role): transform
warehouse (warehouse name): COMPUTE_WH
database (default database that dbt will build objects in): AIRNBN
schema (default schema that dbt will build objects in): DEV
threads (1 or more) [1]: 1
04:57:06  Profile dbtlearn written to /home/hrishi/.dbt/profiles.yml using target's profile_template.yml and your supplied values. Run 'dbt debug' to valida
te the connection.
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy$
```

Now my dbt profile has dbtlearner as user (dev username) and password – Hrishikesh$261999

Checking If the connection is done using the command - dbt_debug

```
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$ dbt debug
04:59:12  Running with dbt=1.9.3
04:59:12  dbt version: 1.9.3
04:59:12  python version: 3.11.0rc1
04:59:12  python path: /mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbt_venv/bin/python3.11
04:59:12  os info: Linux-5.15.153.1-microsoft-standard-WSL2-x86_64-with-glibc2.35
04:59:16  Using profiles dir at /home/hrishi/.dbt
04:59:16  Using profiles.yml file at /home/hrishi/.dbt/profiles.yml
04:59:16  Using dbt_project.yml file at /mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn/dbt_project.yml
04:59:16  adapter type: snowflake
04:59:16  adapter version: 1.9.0
04:59:16  Configuration:
04:59:16    profiles.yml file [OK found and valid]
04:59:16    dbt_project.yml file [OK found and valid]
04:59:16  Required dependencies:
04:59:16   - git [OK found]

04:59:16  Connection:
04:59:16    account: ymixcop-qh42031
04:59:16    user: dbtlearner
04:59:16    database: AIRNBN
04:59:16    warehouse: COMPUTE_WH
04:59:16    role: transform
04:59:16    schema: DEV
04:59:16    authenticator: None
04:59:16    oauth_client_id: None
04:59:16    query_tag: None
04:59:16    client_session_keep_alive: False
04:59:16    host: None
04:59:16    port: None
04:59:16    proxy_host: None
04:59:16    proxy_port: None
04:59:16    protocol: None
04:59:16    connect_retries: 1
04:59:16    connect_timeout: None
04:59:16    retry_on_database_errors: False
04:59:16    retry_all: False
04:59:16    insecure_mode: False
04:59:16    reuse_connections: True
04:59:16  Registered adapter: snowflake=1.9.0
04:59:31    Connection test: [OK connection ok]
```

With all checks passed

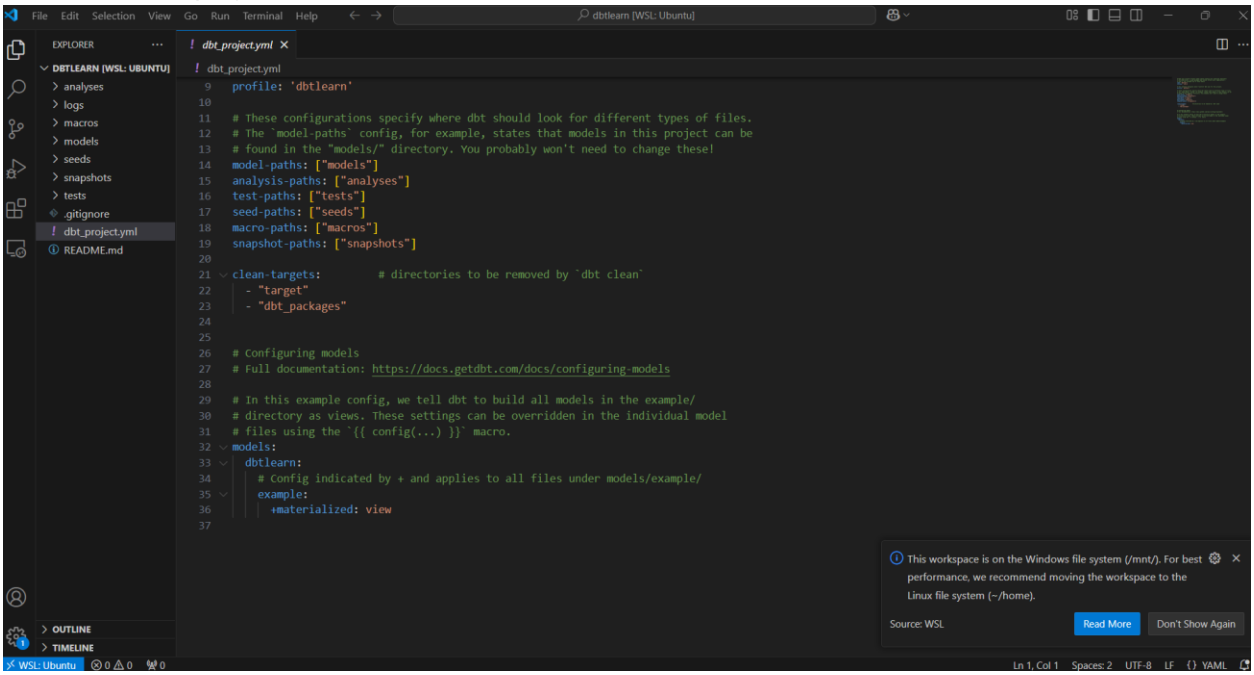
Connection details -

```
04:59:16  Connection:
04:59:16    account: ymixcop-qh42031
04:59:16    user: dbtlearner
04:59:16    database: AIRNBN
04:59:16    warehouse: COMPUTE_WH
04:59:16    role: transform
04:59:16    schema: DEV
04:59:16    authenticator: None
04:59:16    oauth_client_id: None
04:59:16    query_tag: None
04:59:16    client_session_keep_alive: False
04:59:16    host: None
04:59:16    port: None
04:59:16    proxy_host: None
04:59:16    proxy_port: None
04:59:16    protocol: None
04:59:16    connect_retries: 1
04:59:16    connect_timeout: None
04:59:16    retry_on_database_errors: False
04:59:16    retry_all: False
04:59:16    insecure_mode: False
04:59:16    reuse_connections: True
04:59:16  Registered adapter: snowflake=1.9.0
04:59:31    Connection test: [OK connection ok]
```

Opening the project in vscode -

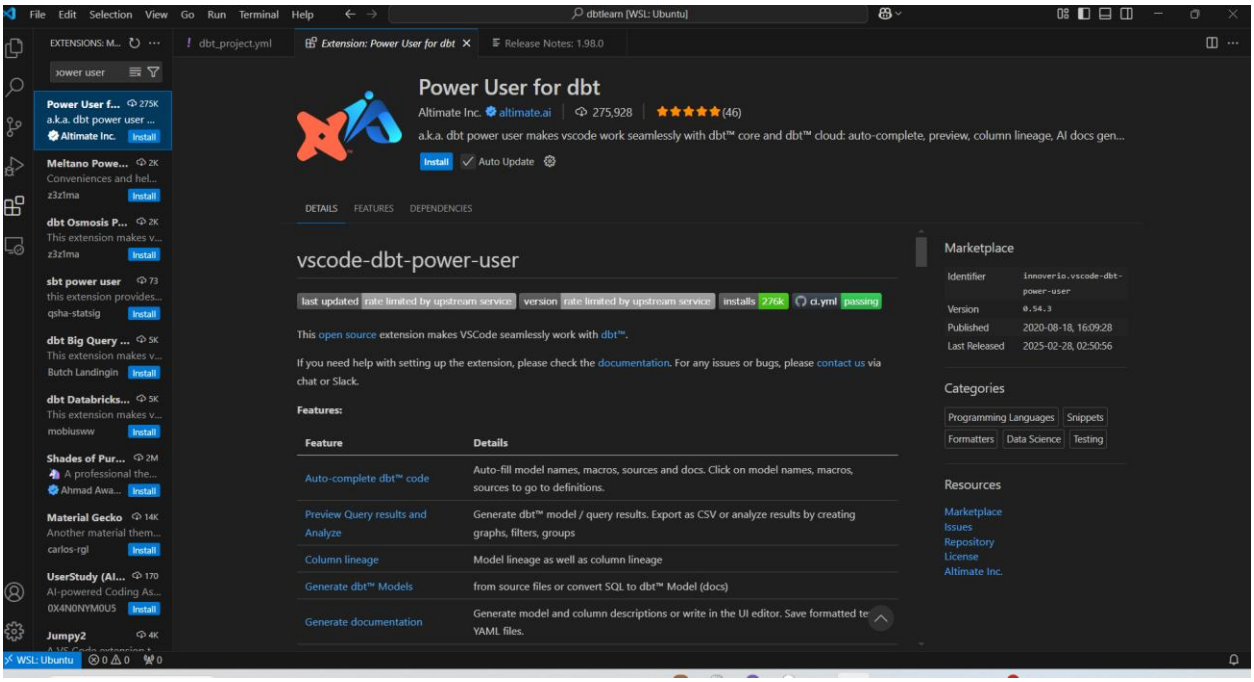
```
04:59:31  All checks passed!
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$ code .
Updating VS Code Server to version e54c774e0add60467559eb0d1e229c6452cf8447
Removing previous installation...
Installing VS Code Server for Linux x64 (e54c774e0add60467559eb0d1e229c6452cf8447)
Downloading: 100%
Unpacking: 100%
Unpacked 2064 files and folders to /home/hrishi/.vscode-server/bin/e54c774e0add60467559eb0d1e229c6452cf8447.
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$
```

This is how the project looks.

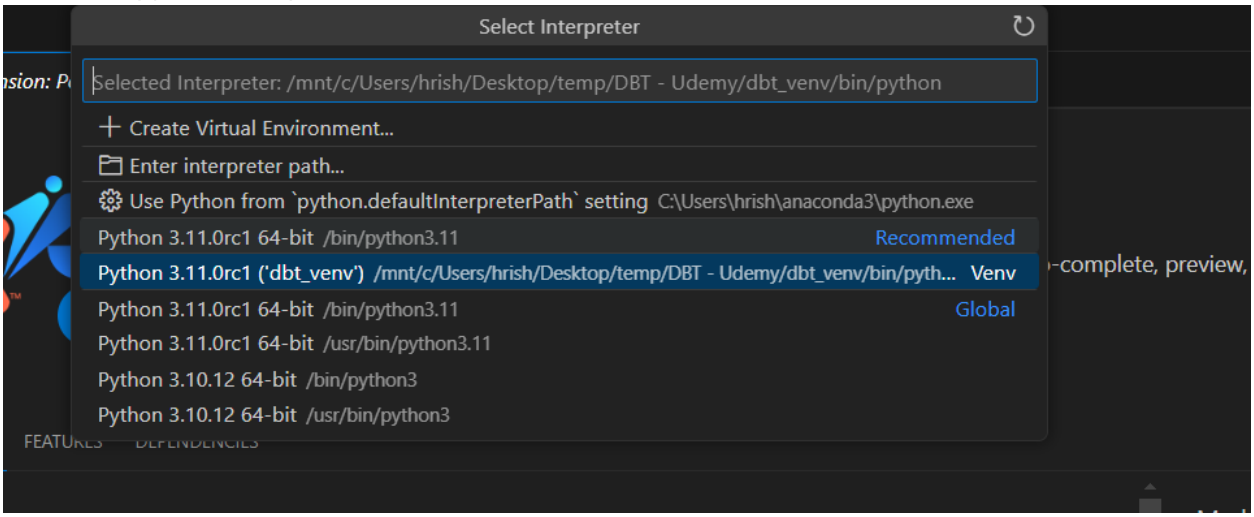



now, here delete line – 35-36 as they are just for examples which were initially set up

Install dbt-power user extension for better assistance while writing dbt code -



Select the python interpreter -

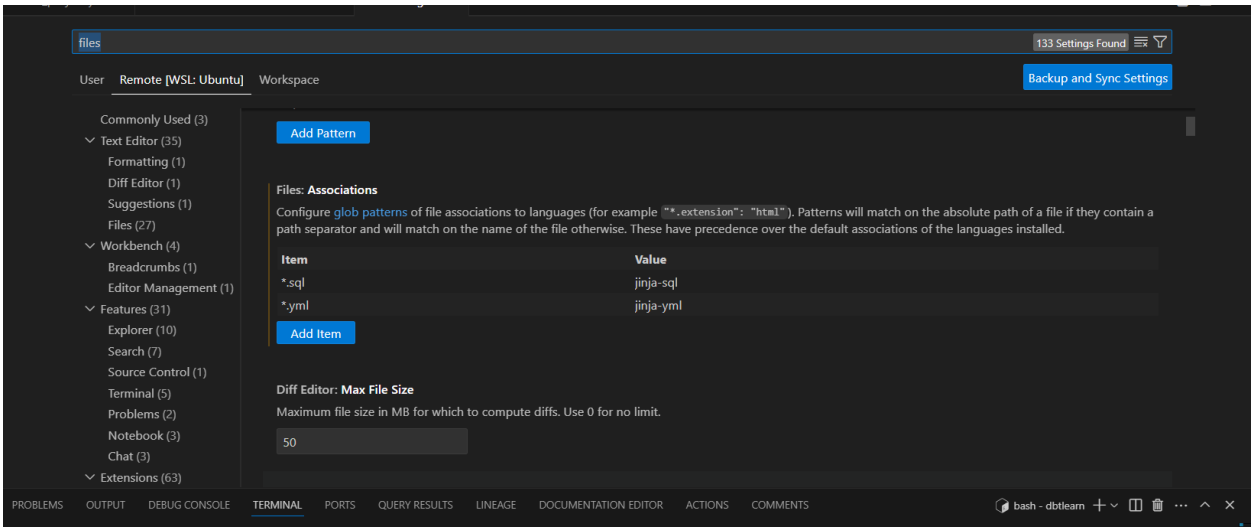



Or enter the file path manually (the venv is located in project folder)

After this, You also need to setup dbt file associations with jinja – Jinja is basically a template engine that helps you write dynamic code by mixing regular text (like SQL or HTML) with placeholders for variables and simple logic (like if statements or for loops).

Placeholders: You can write something like {{ my_variable }} and Jinja will replace it with a real value at runtime.

Logic: You can add simple if conditions or loops to generate code automatically.

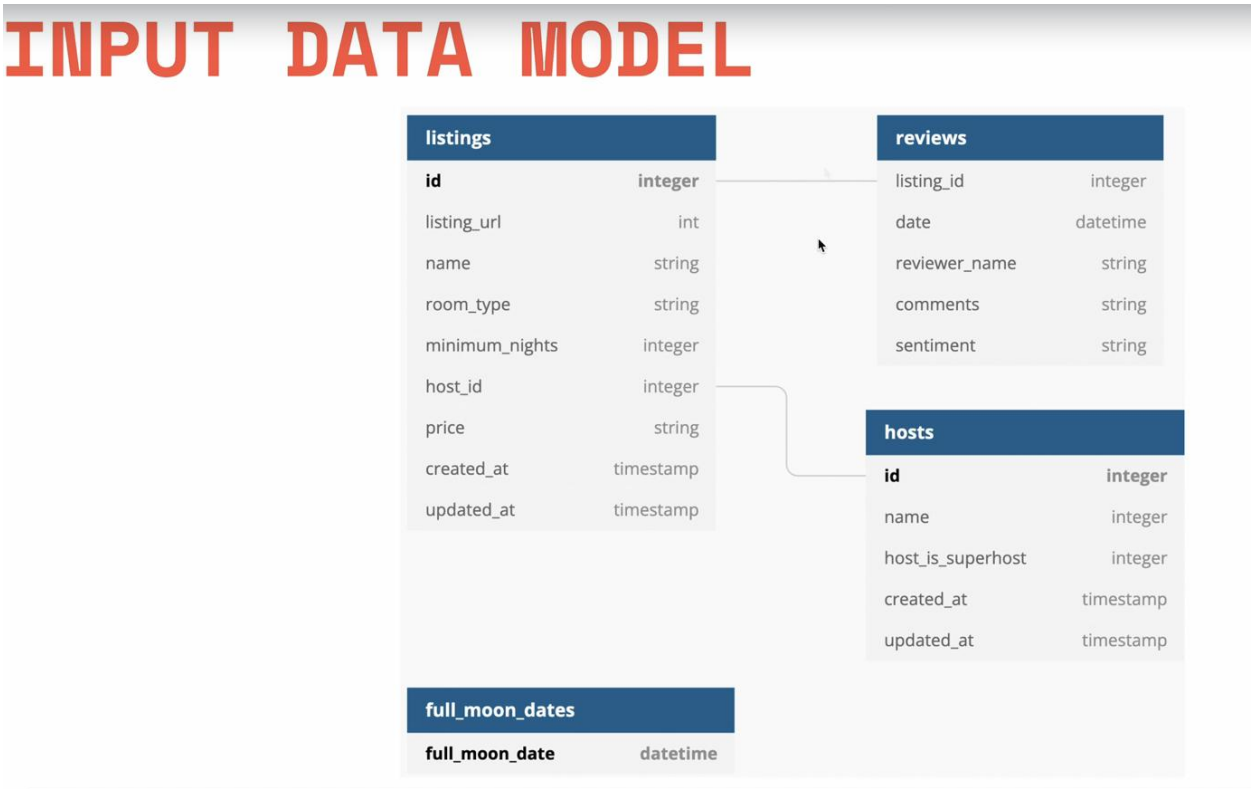


And then verify dbt setup – dbt debug

```
17:33:15  Connection:
17:33:15    account: ymixcop-qh42031
17:33:15    user: dbtlearner
17:33:15    database: AIRNBN
17:33:15    warehouse: COMPUTE_WH
17:33:15    role: transform
17:33:15    schema: DEV
17:33:15    authenticator: None
17:33:15    oauth_client_id: None
17:33:15    query_tag: None
17:33:15    client_session_keep_alive: False
17:33:15    host: None
17:33:15    port: None
17:33:15    proxy_host: None
17:33:15    proxy_port: None
17:33:15    protocol: None
17:33:15    connect_retries: 1
17:33:15    connect_timeout: None
17:33:15    retry_on_database_errors: False
17:33:15    retry_all: False
17:33:15    insecure_mode: False
17:33:15    reuse_connections: True
17:33:15  Registered adapter: snowflake=1.9.0
17:33:38    Connection test: [OK connection ok]

17:33:38  All checks passed!
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$
```

dbt core

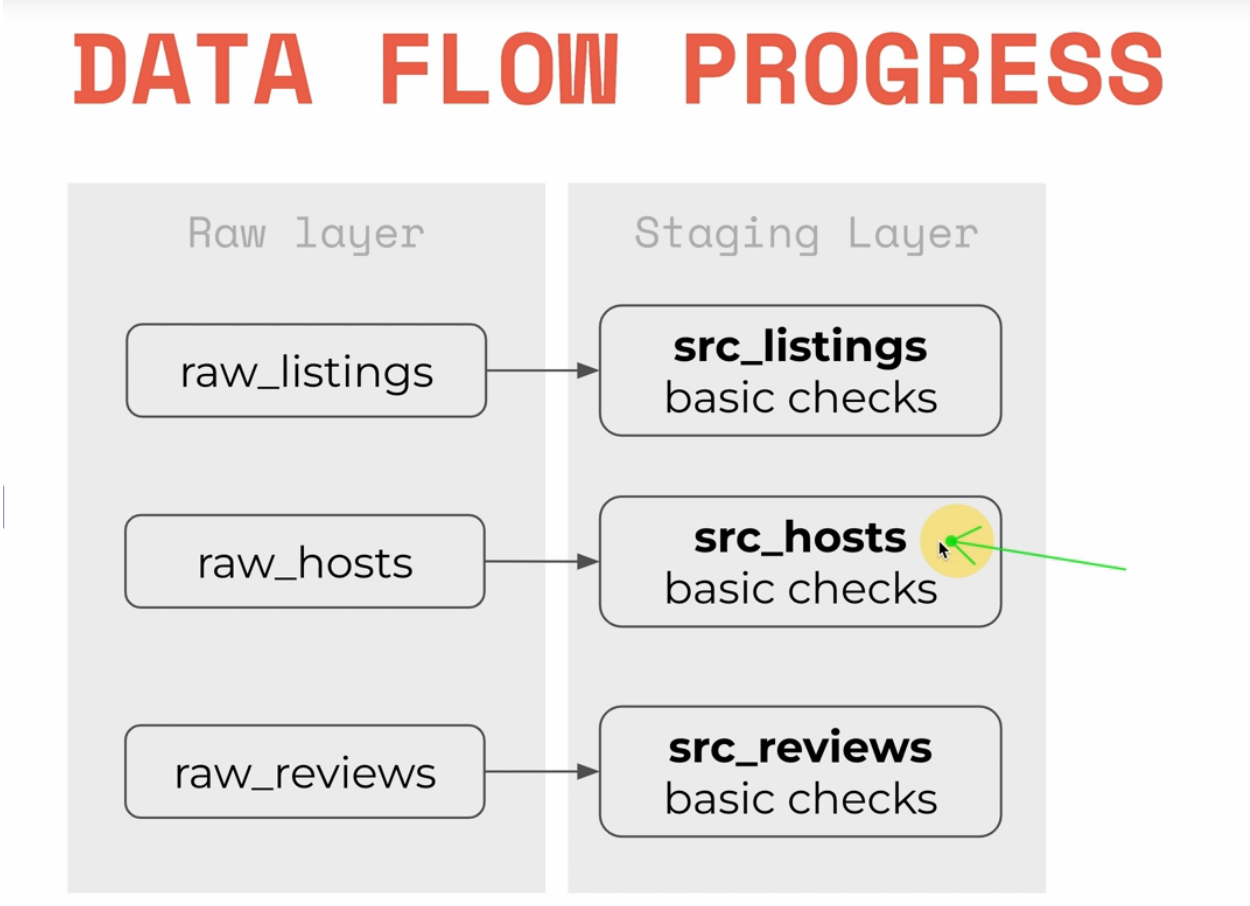
Now getting to understand data model of Airbnb and the data flow overview which we are going to build.





now that we are known of the data model that we are going to produce. We'll start with the data modelling process. Remember we have a raw layer from our database namely – RAW_HOSTS, RAW_LISTING, RAW_REVIEWS.

We'll now create a sql worksheet where we'll transform data from this raw layer to a staging layer referred to as src_listings, src_hosts, src_reviews. Here, we'll be using CTE SQL statements to transform the data in our required staging form.



This is the raw_listings data that we have -

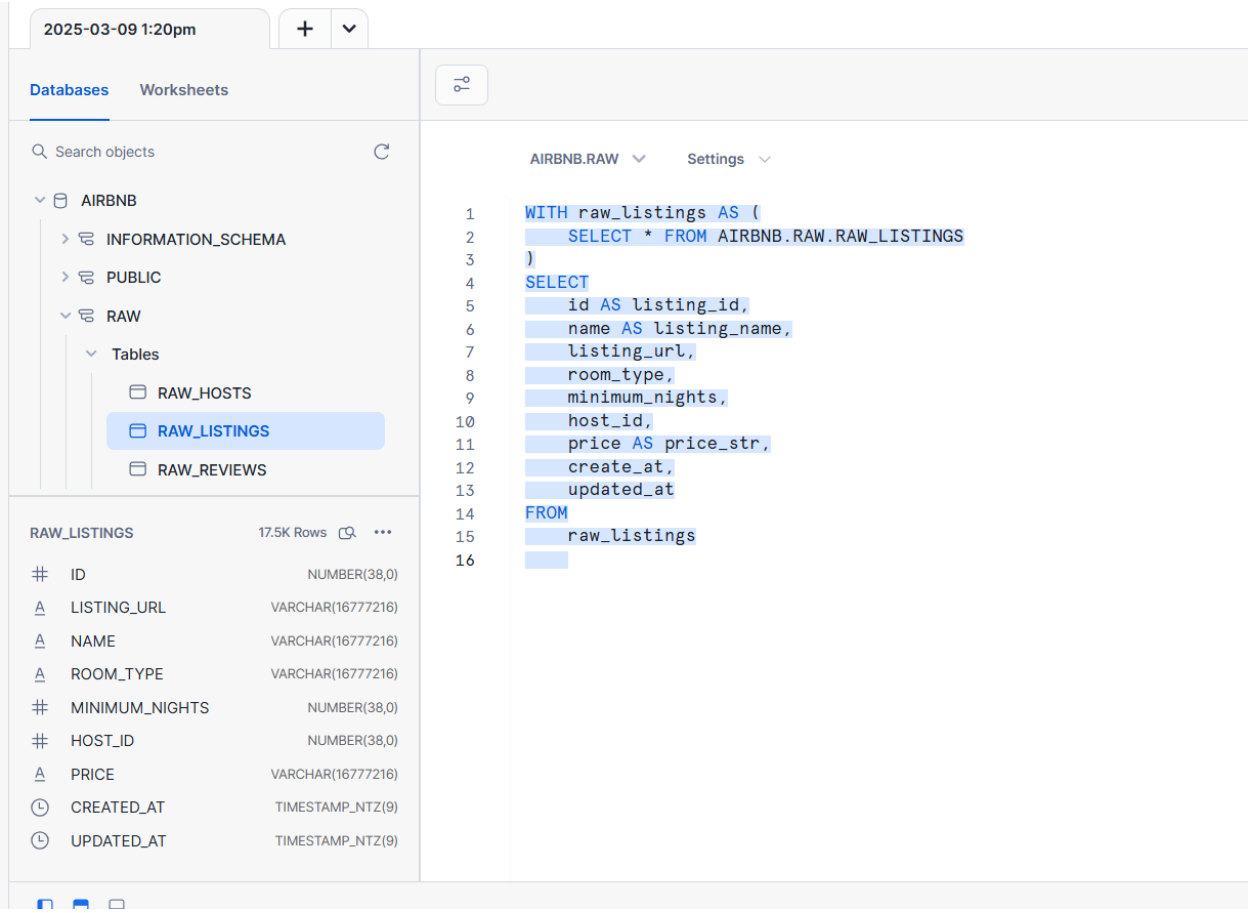
| | ID | LISTING_URL | NAME | ROOM_TYPE | MINIMUM_NIGHT |
|---|---|---|---|---|---|
| 1 | 3176 | https://www.airbnb.com/rooms/3176 | Fabulous Flat in great Location | Entire home/apt | 6 |
| 2 | 7071 | https://www.airbnb.com/rooms/7071 | BrightRoom with sunny greenview! | Private room | |
| 3 | 9991 | https://www.airbnb.com/rooms/9991 | Geourgeous flat - outstanding views | Entire home/apt | |
| 4 | 14325 | https://www.airbnb.com/rooms/14325 | Apartment in Prenzlauer Berg | Entire home/apt | 9 |
| 5 | 16644 | https://www.airbnb.com/rooms/16644 | In the Heart of Berlin - Kreuzberg | Entire home/apt | 6 |
| 6 | 17904 | https://www.airbnb.com/rooms/17904 | Beautiful Kreuzberg studio - 3 months minimum | Entire home/apt | 9 |
| 7 | 20858 | https://www.airbnb.com/rooms/20858 | Designer Loft in Berlin Mitte | Entire home/apt | |
| 8 | 21869 | https://www.airbnb.com/rooms/21869 | Studio in the Heart of Kreuzberg | Entire home/apt | 6 |
| 9 | 22438 | https://www.airbnb.com/rooms/22438 | WOHNUNG IN BERLIN ★ MITTE | Entire home/apt | 9 |
| 10 | 22677 | https://www.airbnb.com/rooms/22677 | Prenzel garden with leafy terrace (quiet Guests) | Entire home/apt | |
| 11 | 23834 | https://www.airbnb.com/rooms/23834 | Apartment in the heart of Berlin | Entire home/apt | 18 |

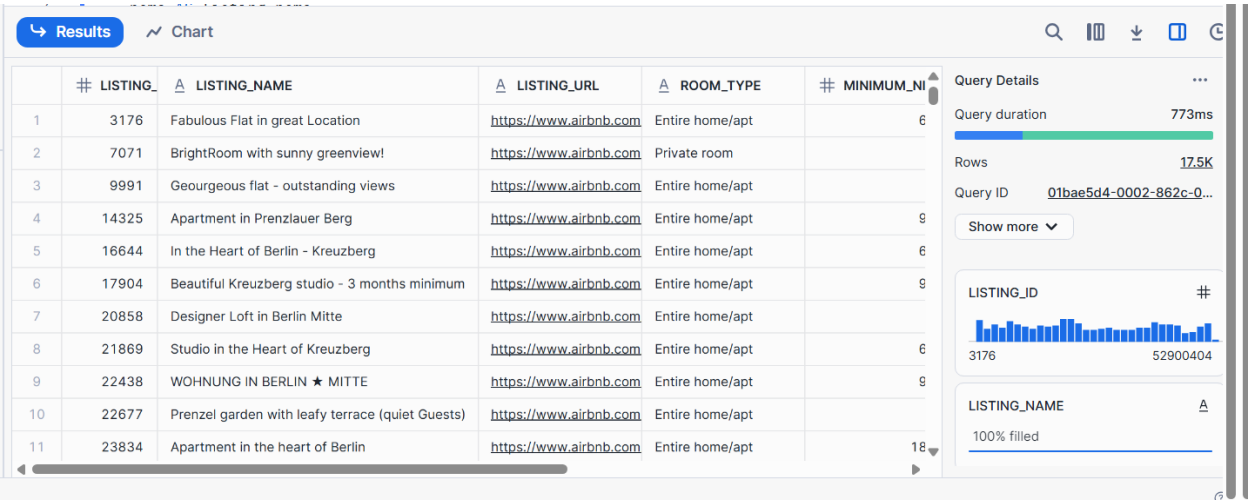
RAW_LISTINGS 17.5K Rows

| | |
|---|---|
| ID | NUMBER(38,0) |
| LISTING_URL | VARCHAR(16777216) |
| NAME | VARCHAR(16777216) |
| ROOM_TYPE | VARCHAR(16777216) |
| MINIMUM_NIGHTS | NUMBER(38,0) |
| HOST_ID | NUMBER(38,0) |
| PRICE | VARCHAR(16777216) |
| CREATED_AT | TIMESTAMP_NTZ(9) |
| UPDATED_AT | TIMESTAMP_NTZ(9) |

```
14  FROM
15      raw_listings
16
```

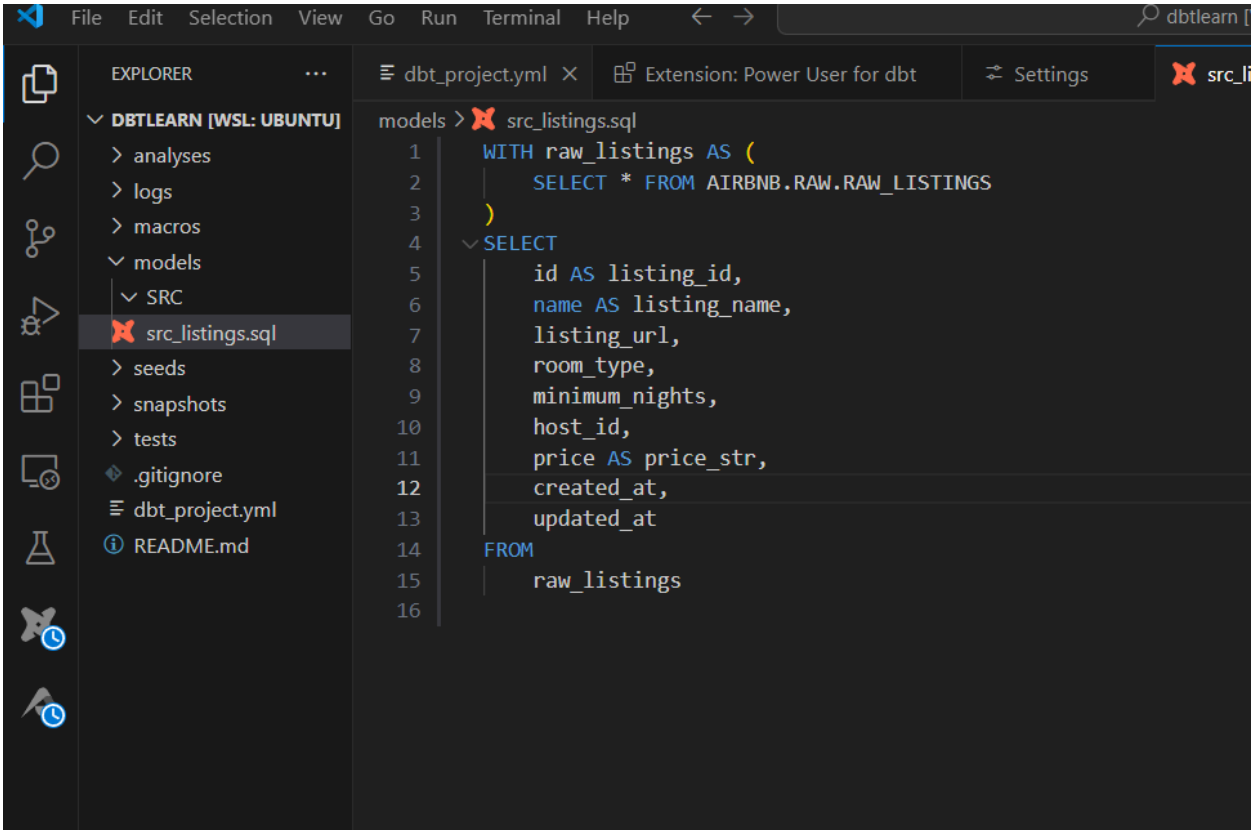
Writing cte -





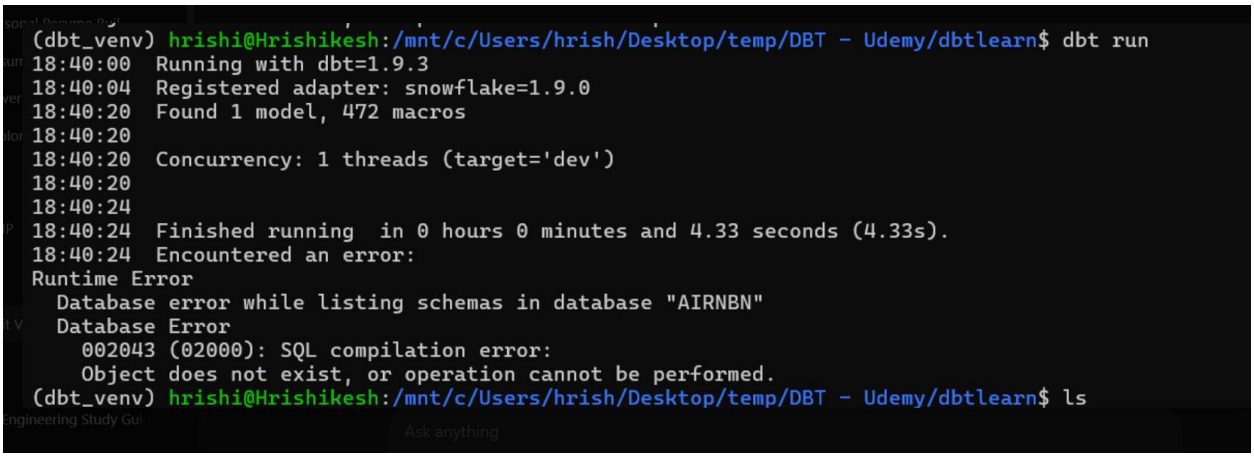

| | LISTING_ | LISTING_NAME | LISTING_URL | ROOM_TYPE | MINIMUM_NI |
|---|---|---|---|---|---|
| 1 | 3176 | Fabulous Flat in great Location | https://www.airbnb.com | Entire home/apt | 6 |
| 2 | 7071 | BrightRoom with sunny greenview! | https://www.airbnb.com | Private room | |
| 3 | 9991 | Geourgeous flat - outstanding views | https://www.airbnb.com | Entire home/apt | |
| 4 | 14325 | Apartment in Prenzlauer Berg | https://www.airbnb.com | Entire home/apt | 9 |
| 5 | 16644 | In the Heart of Berlin - Kreuzberg | https://www.airbnb.com | Entire home/apt | 6 |
| 6 | 17904 | Beautiful Kreuzberg studio - 3 months minimum | https://www.airbnb.com | Entire home/apt | 9 |
| 7 | 20858 | Designer Loft in Berlin Mitte | https://www.airbnb.com | Entire home/apt | |
| 8 | 21869 | Studio in the Heart of Kreuzberg | https://www.airbnb.com | Entire home/apt | 6 |
| 9 | 22438 | WOHNUNG IN BERLIN ★ MITTE | https://www.airbnb.com | Entire home/apt | 9 |
| 10 | 22677 | Prenzel garden with leafy terrace (quiet Guests) | https://www.airbnb.com | Entire home/apt | |
| 11 | 23834 | Apartment in the heart of Berlin | https://www.airbnb.com | Entire home/apt | 18 |

Now, we'll need to integrate this query in our dbt project




To run this, save and go to terminal and use the command – ***dbt run***

Error faced -




Encountered a mistake for incorrect database name by mistake – ‘AIRNBN’ instead of ‘AIRBNB’

Corrected it by locating the profiles.yml in correct folder- in wsl file system – home/hrishi/.dbt instead of users/hrishi which is a windows path and not the wsl path where I’m working. However, dbt was reporting two different files:

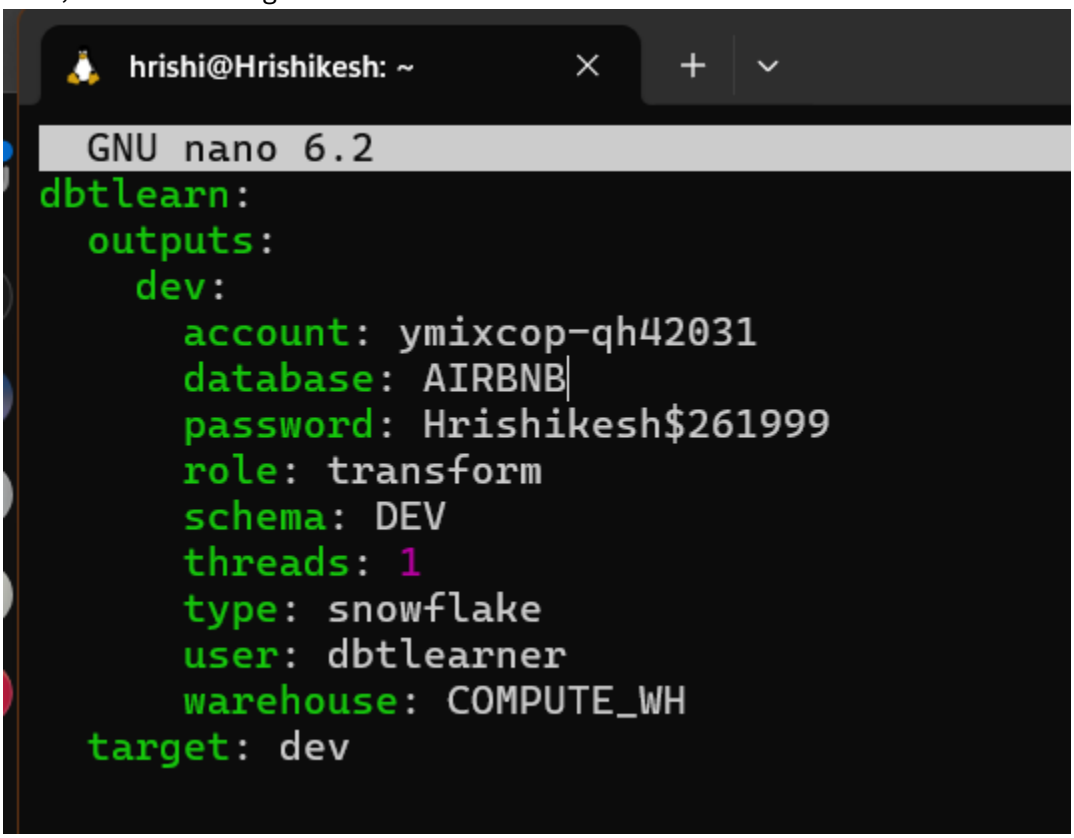
profiles.yml at ~/home/hrish/.dbt/profiles.yml
This file stores your connection details (Snowflake account, database, schema, role, etc.).
dbt_project.yml at /mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn/dbt_project.yml

This file defines the name of your dbt project, the folder structure for models, the target schema name, and other project-specific settings.
They serve two separate purposes, so it’s normal for dbt to find them in two different places:

profiles.yml is a user-level config file, typically located in ~/.dbt/ in your WSL home directory.

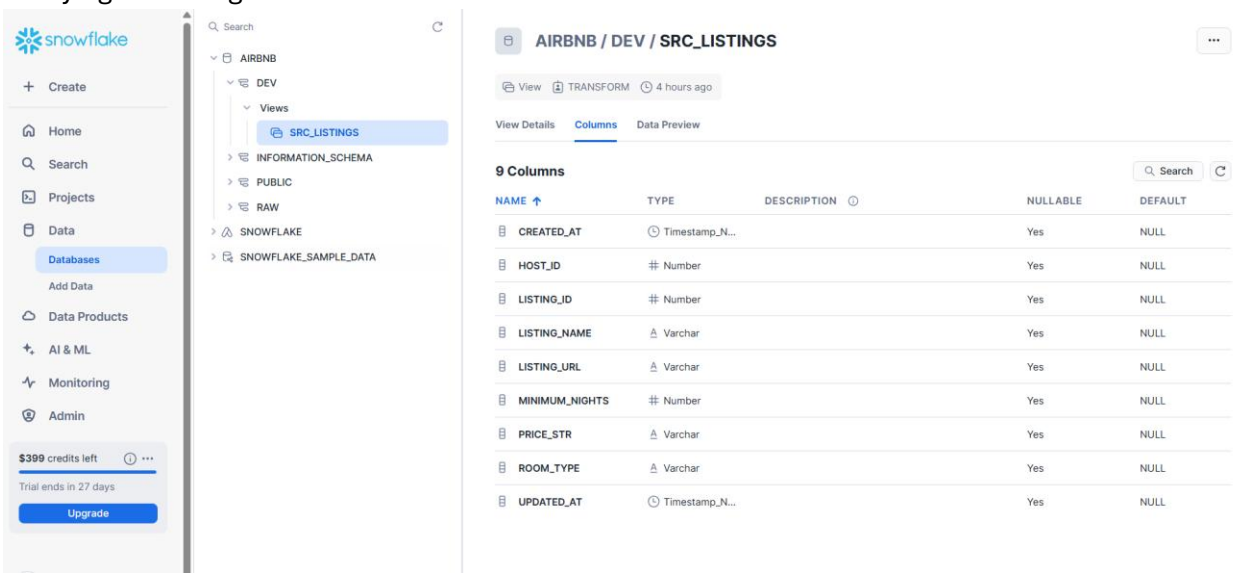
dbt_project.yml is a project-level config file, stored alongside your dbt project in your code folder.

Now, Made the changes -

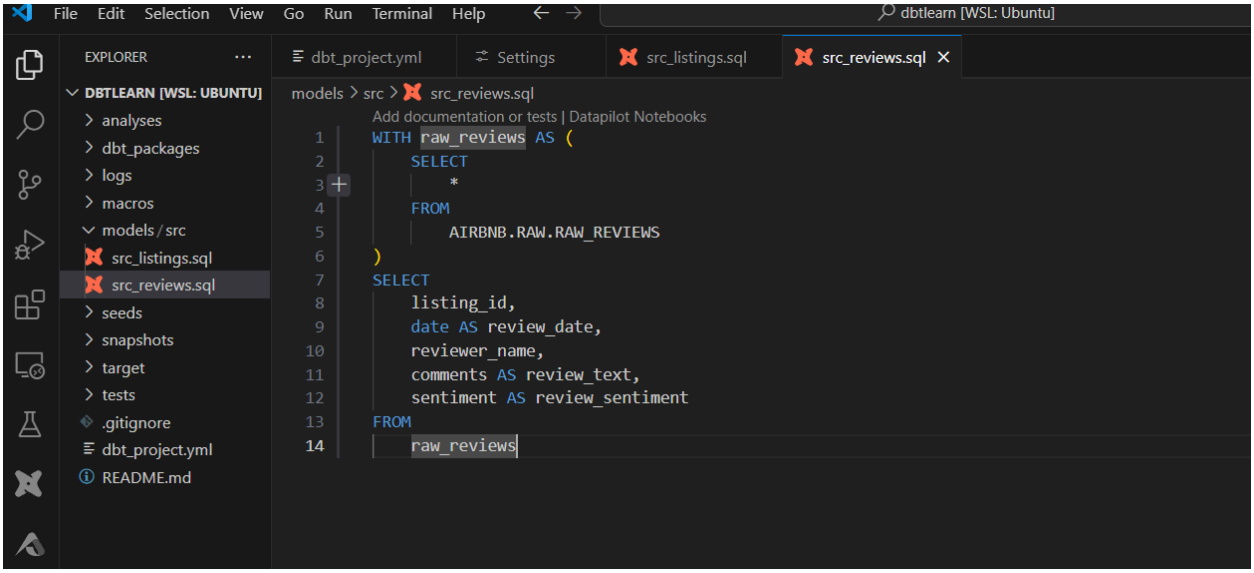



```
19:30:08  All checks passed!
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$ code .
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$ dbt run
19:31:04  Running with dbt=1.9.3
19:31:08  Registered adapter: snowflake=1.9.0
19:31:26  Unable to do partial parsing because profile has changed
19:31:32  Found 1 model, 472 macros
19:31:32
19:31:32  Concurrency: 1 threads (target='dev')
19:31:32
19:31:38  1 of 1 START sql view model DEV.src_listings ................................... [RUN]
19:31:39  1 of 1 OK created sql view model DEV.src_listings .............................. [SUCCESS 1 in 0.84s]
19:31:39
19:31:39  Finished running 1 view model in 0 hours 0 minutes and 6.81 seconds (6.81s).
19:31:39
19:31:39  Completed successfully
19:31:39
19:31:39  Done. PASS=1 WARN=0 ERROR=0 SKIP=0 TOTAL=1
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$
```

Verifying the changes in snowflake -



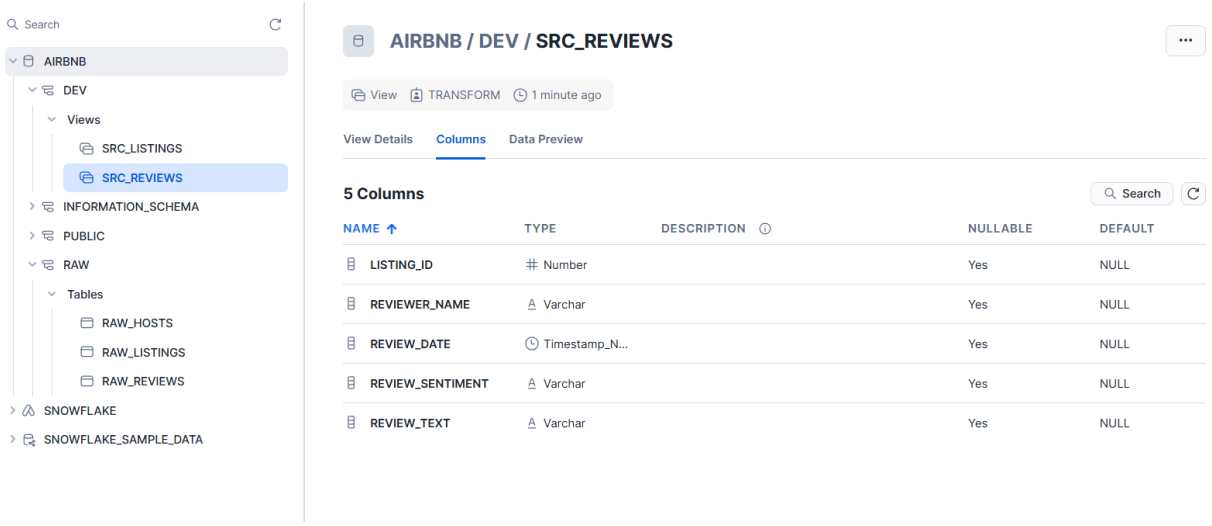

Applied the same step to create a *raw_reviews*




Dbt run –

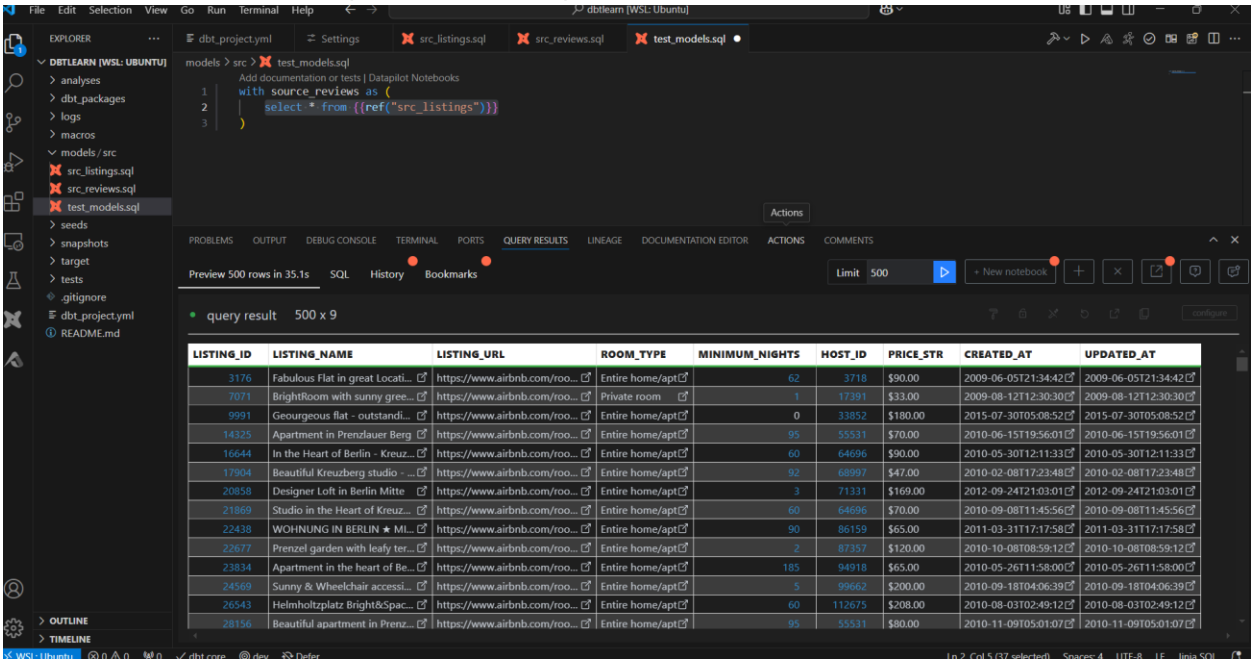
```
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$ dbt run
00:00:55  Running with dbt=1.9.3
00:00:59  Registered adapter: snowflake=1.9.0
00:01:14  Found 2 models, 472 macros
00:01:14
00:01:14  Concurrency: 1 threads (target='dev')
00:01:14
00:01:18  1 of 2 START sql view model DEV.src_listings ................................... [RUN]
00:01:19  1 of 2 OK created sql view model DEV.src_listings .............................. [SUCCESS 1 in 0.43s]
00:01:19  2 of 2 START sql view model DEV.src_reviews .................................... [RUN]
00:01:19  2 of 2 OK created sql view model DEV.src_reviews ............................... [SUCCESS 1 in 0.38s]
00:01:19
00:01:19  Finished running 2 view models in 0 hours 0 minutes and 5.46 seconds (5.46s).
00:01:19
00:01:19  Completed successfully
00:01:19
00:01:19  Done. PASS=2 WARN=0 ERROR=0 SKIP=0 TOTAL=2
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$
```

Verifying changes in snowflake-

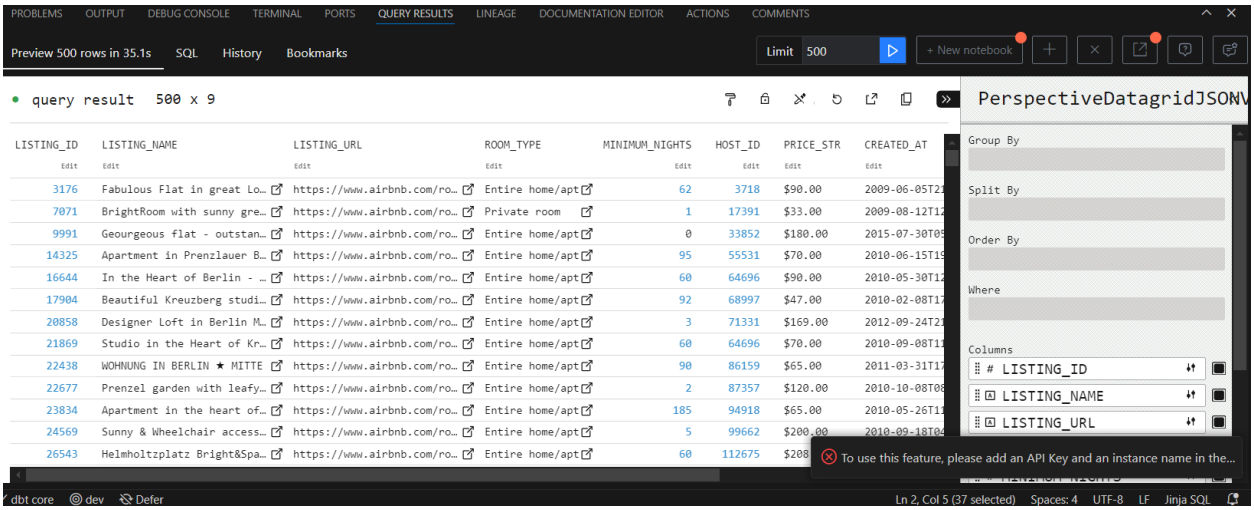



Created a test model to showcase how dbt allows to preview the data by using jinja references and autocompletes. What I needed to do was to select the sql statement which I want a preview for and

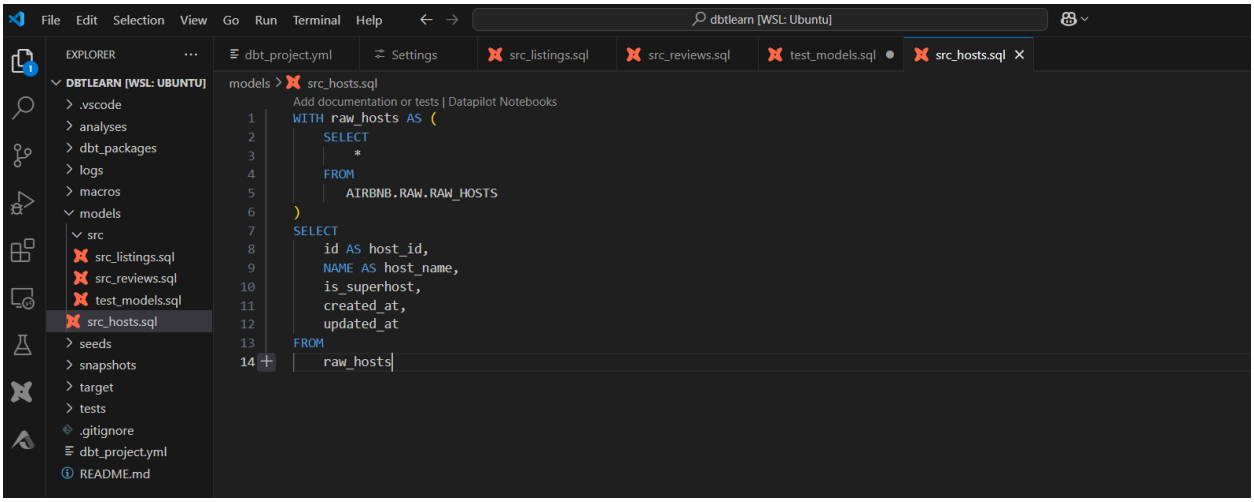
then hit run dbt button at top which shows a preview in terminal as a below screenshot -




Also can do quick operations use for data analysis as groupby, order by, limits etc -



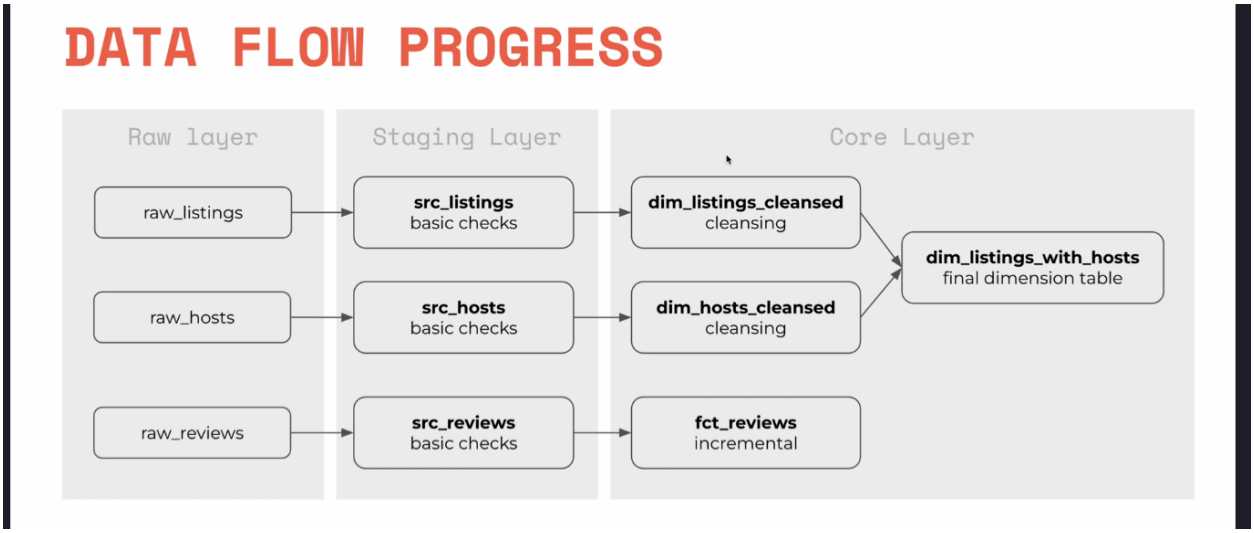

Creating another view called – src_hosts.sql –




```
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$ dbt run
00:50:41  Running with dbt=1.9.3
00:50:45  Registered adapter: snowflake=1.9.0
00:50:59  Found 4 models, 472 macros
00:50:59
00:50:59  Concurrency: 1 threads (target='dev')
00:50:59
00:51:03  1 of 3 START sql view model DEV.src_hosts ..................................... [RUN]
00:51:04  1 of 3 OK created sql view model DEV.src_hosts ................................ [SUCCESS 1 in 0.31s]
00:51:04  2 of 3 START sql view model DEV.src_listings .................................. [RUN]
00:51:04  2 of 3 OK created sql view model DEV.src_listings ............................. [SUCCESS 1 in 0.40s]
00:51:04  3 of 3 START sql view model DEV.src_reviews ................................... [RUN]
00:51:05  3 of 3 OK created sql view model DEV.src_reviews .............................. [SUCCESS 1 in 0.52s]
00:51:05
00:51:05  Finished running 3 view models in 0 hours 0 minutes and 5.40 seconds (5.40s).
00:51:05
00:51:05  Completed successfully
00:51:05
00:51:05  Done. PASS=3 WARN=0 ERROR=0 SKIP=0 TOTAL=3
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$
```

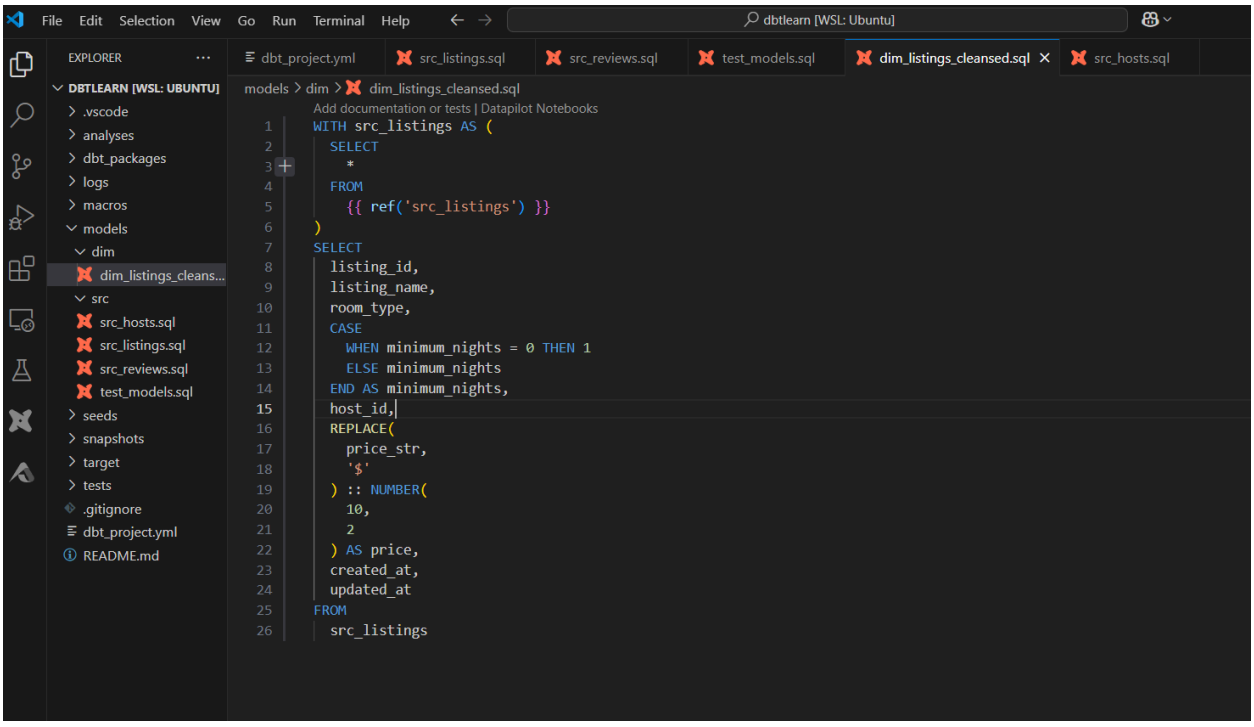
MATERIALIZATIONS

types – view, tables, incremental, ephemeral




As the above image, we are going to create different materializations for core layer (dimensions)

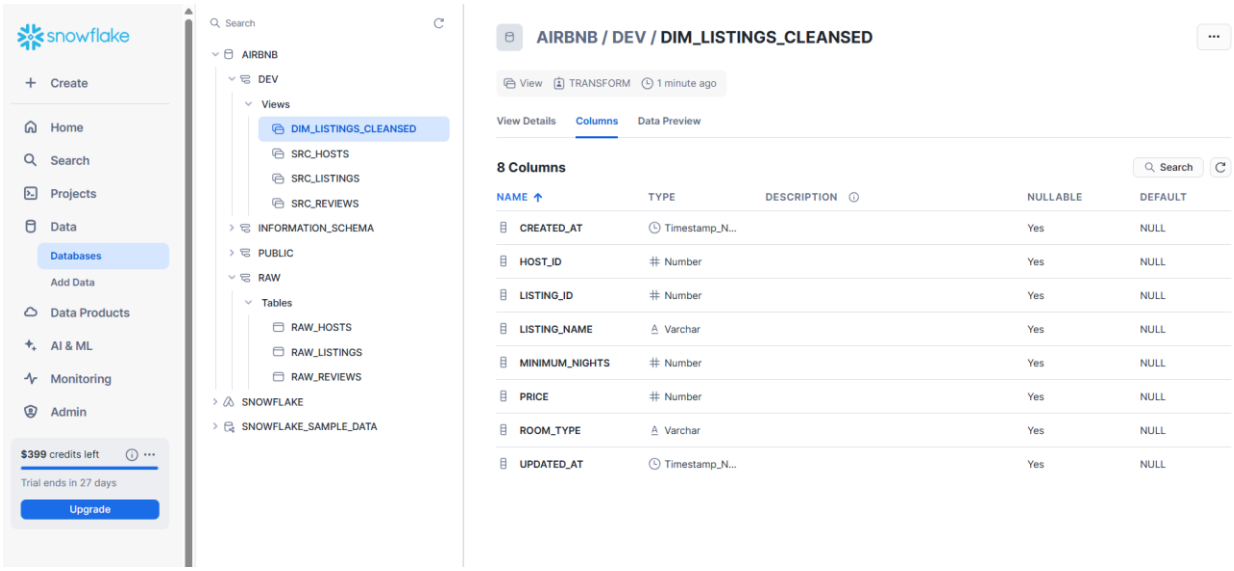
Now, we'll be creating dim_listings_cleansed on top of src_listings to do additional cleansing (i.e. Data cleaning) and similarly for src_reviews, src_hosts, etc.



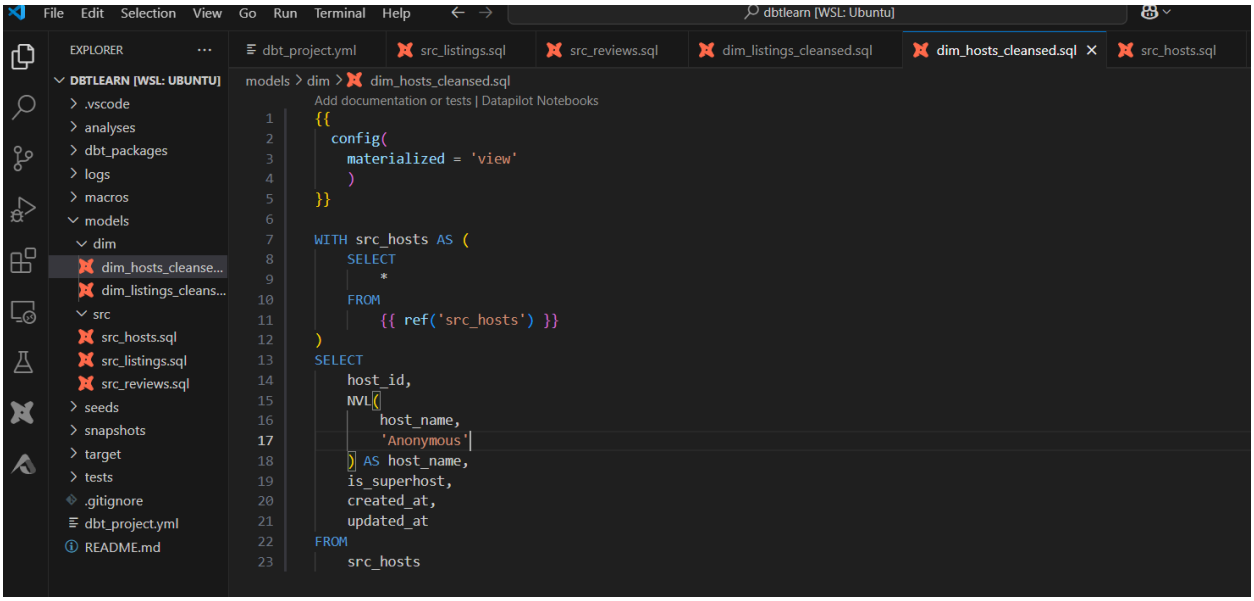

To run dbt model here successfully I deleted the test_models.sql as it was not needed it was just build for testing the referencing made by jinja.

```
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$ dbt run
01:28:25  Running with dbt=1.9.3
01:28:28  Registered adapter: snowflake=1.9.0
01:28:47  Found 4 models, 472 macros
01:28:47
01:28:47  Concurrency: 1 threads (target='dev')
01:28:47
01:28:52  1 of 4 START sql view model DEV.src_hosts ..................................... [RUN]
01:28:53  1 of 4 OK created sql view model DEV.src_hosts ................................ [SUCCESS 1 in 0.38s]
01:28:53  2 of 4 START sql view model DEV.src_listings .................................. [RUN]
01:28:53  2 of 4 OK created sql view model DEV.src_listings ............................. [SUCCESS 1 in 0.21s]
01:28:53  3 of 4 START sql view model DEV.src_reviews ................................... [RUN]
01:28:53  3 of 4 OK created sql view model DEV.src_reviews .............................. [SUCCESS 1 in 0.30s]
01:28:53  4 of 4 START sql view model DEV.dim_listings_cleansed ......................... [RUN]
01:28:54  4 of 4 OK created sql view model DEV.dim_listings_cleansed .................... [SUCCESS 1 in 0.36s]
01:28:54
01:28:54  Finished running 4 view models in 0 hours 0 minutes and 6.48 seconds (6.48s).
01:28:54
01:28:54  Completed successfully
01:28:54
01:28:54  Done. PASS=4 WARN=0 ERROR=0 SKIP=0 TOTAL=4
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$
```

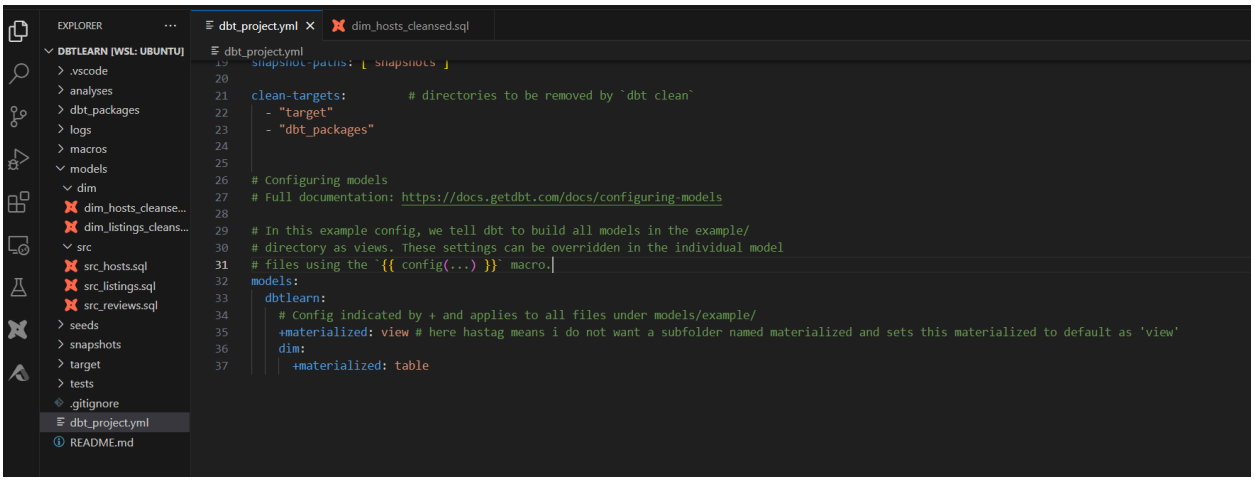
Verifying in snowflake –




Dim_hosts_cleansed.sql




Now, the source models should be set default as ‘views’ because they are not that offend used. SRC transformations are lightweight and not often used And the dim tables should be materialized as ‘table’ because they will be used commonly.

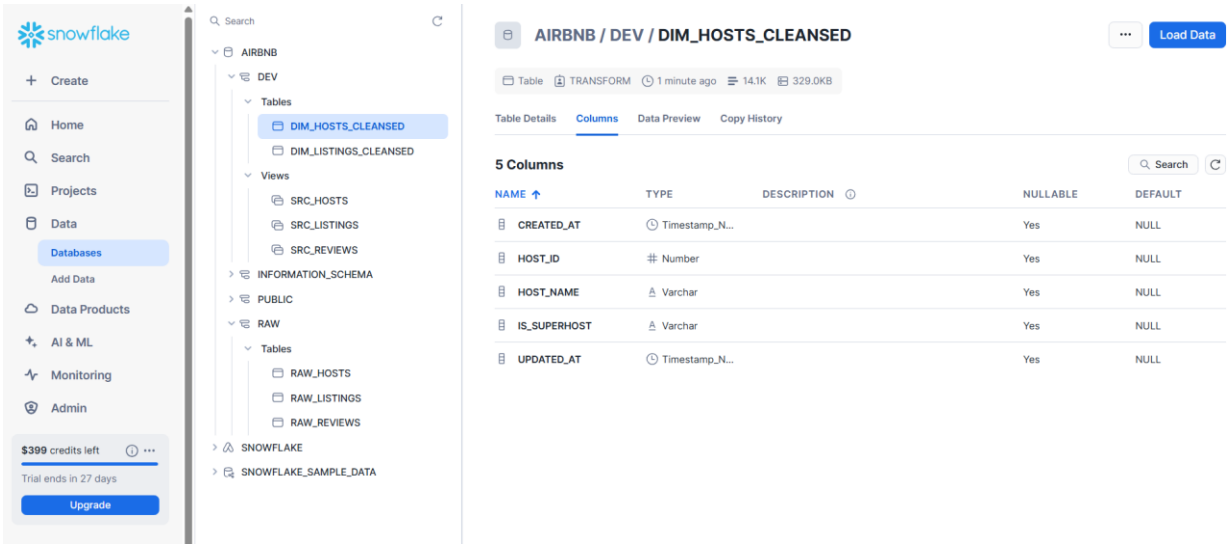



```
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$ dbt run
03:33:14  Running with dbt=1.9.3
03:33:19  Registered adapter: snowflake=1.9.0
03:33:37  Found 5 models, 472 macros
03:33:37
03:33:37  Concurrency: 1 threads (target='dev')
03:33:37
03:33:42  1 of 5 START sql view model DEV.src_hosts ..................................... [RUN]
03:33:42  1 of 5 OK created sql view model DEV.src_hosts ................................ [SUCCESS 1 in 0.57s]
03:33:42  2 of 5 START sql view model DEV.src_listings .................................. [RUN]
03:33:43  2 of 5 OK created sql view model DEV.src_listings ............................. [SUCCESS 1 in 0.81s]
03:33:43  3 of 5 START sql view model DEV.src_reviews ................................... [RUN]
03:33:43  3 of 5 OK created sql view model DEV.src_reviews .............................. [SUCCESS 1 in 0.26s]
03:33:43  4 of 5 START sql view model DEV.dim_hosts_cleansed ............................ [RUN]
03:33:44  4 of 5 OK created sql view model DEV.dim_hosts_cleansed ....................... [SUCCESS 1 in 0.37s]
03:33:44  5 of 5 START sql table model DEV.dim_listings_cleansed ........................ [RUN]
03:33:45  5 of 5 OK created sql table model DEV.dim_listings_cleansed ................... [SUCCESS 1 in 1.73s]
03:33:45
03:33:45  Finished running 1 table model, 4 view models in 0 hours 0 minutes and 8.87 seconds (8.87s).
03:33:46
03:33:46  Completed successfully
03:33:46
03:33:46  Done. PASS=5 WARN=0 ERROR=0 SKIP=0 TOTAL=5
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$
```

Point to remember –


Here earlier I set the type of materialization explicitly for dim_hosts_cleansed.sql as ‘view’.

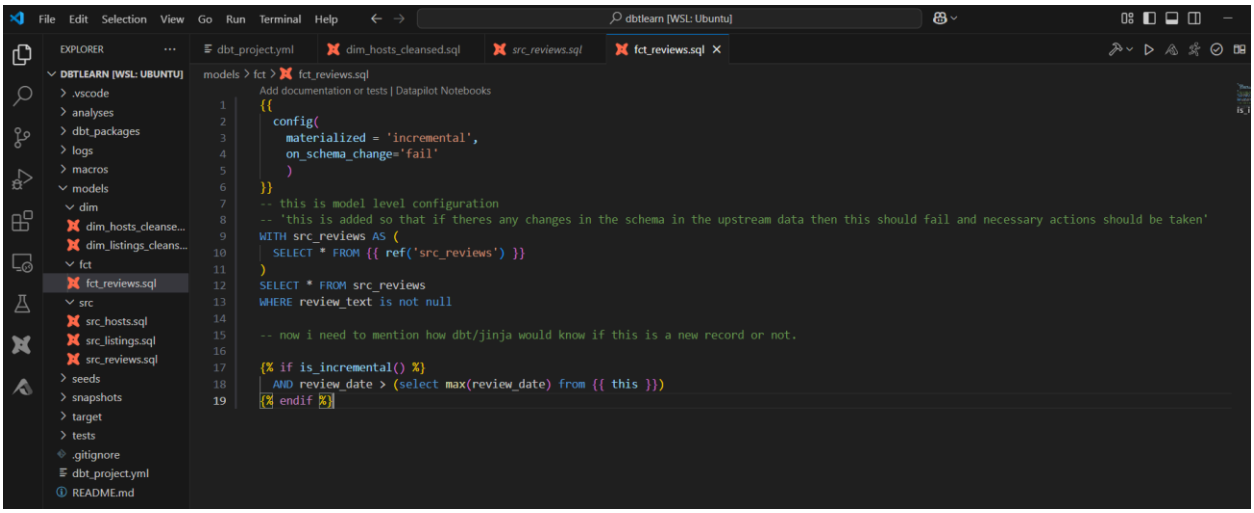
Due to which this dimension is not added to tables category in snowflake.
we have to make a change to ‘table’ so that it is treated as materialized for table type or just remove this explicit mentioning of materialized. Safe because we have mentioned materializing policy in dbt_project.yml.



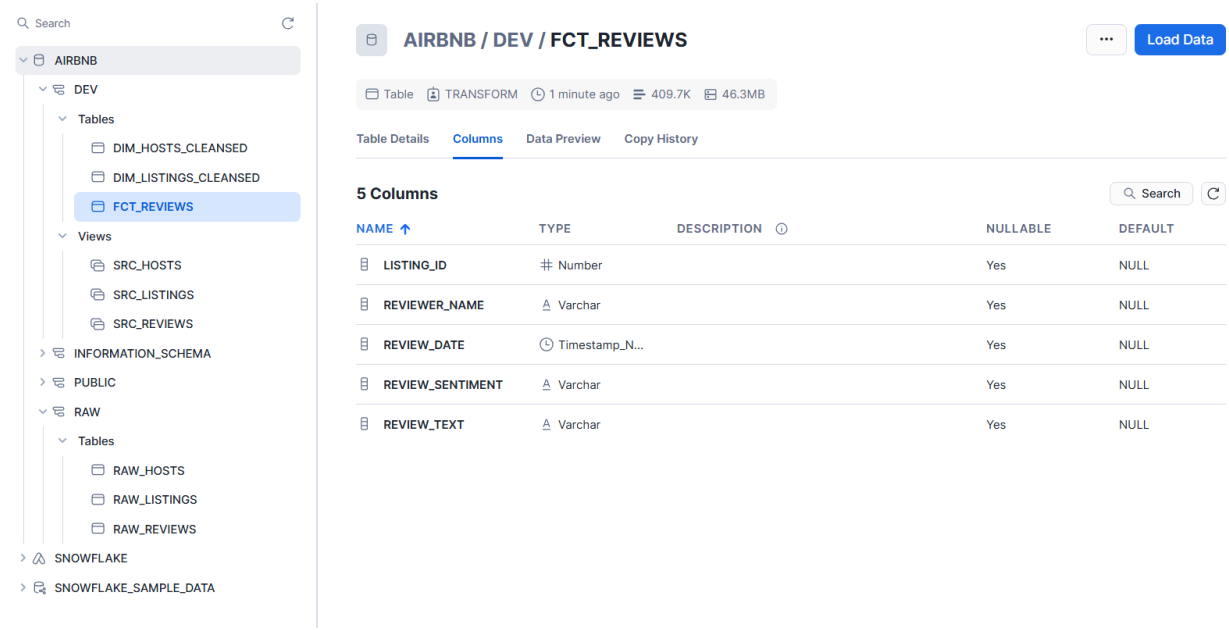

INCREMENTAL MATERIALIZATION

This type of materialization will be applied to raw_reviews because they will be appended as and when reviews are posted for the listings. And also to make this table updated incrementally at every dbt run and not recreated every time.

This is our fact table which has reviews and will be updated incrementally.

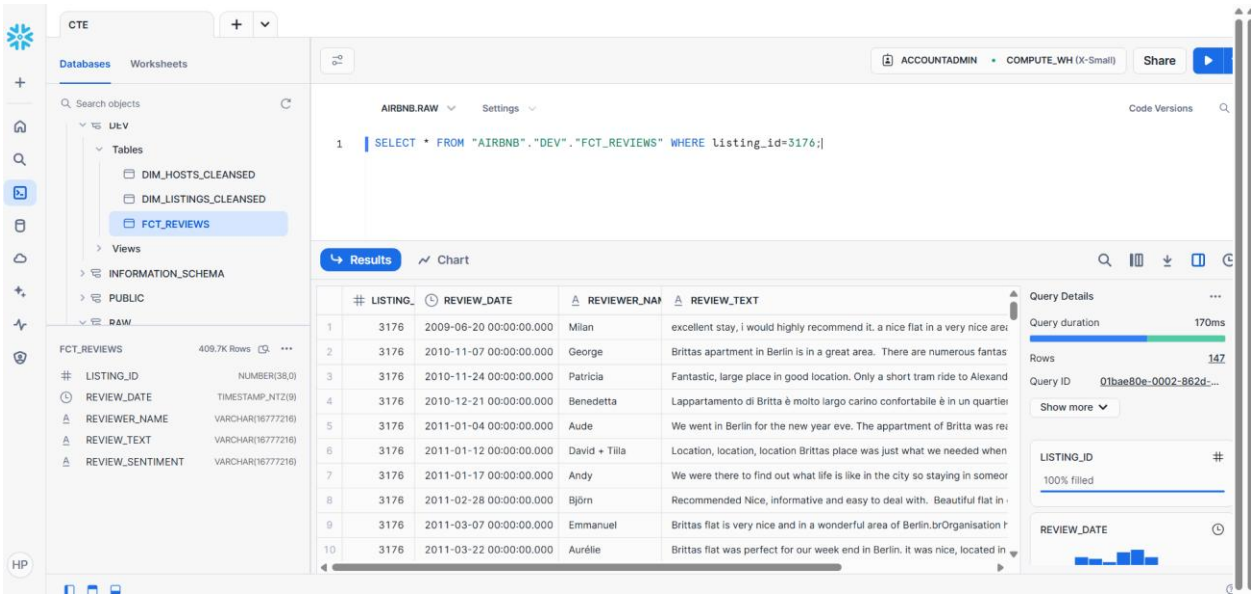


```
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$ dbt run
03:55:05  Running with dbt=1.9.3
03:55:09  Registered adapter: snowflake=1.9.0
03:55:30  Found 6 models, 472 macros
03:55:30
03:55:30  Concurrency: 1 threads (target='dev')
03:55:30
03:55:35  1 of 6 START sql view model DEV.src_hosts ..................................... [RUN]
03:55:35  1 of 6 OK created sql view model DEV.src_hosts ................................ [SUCCESS 1 in 0.52s]
03:55:35  2 of 6 START sql view model DEV.src_listings .................................. [RUN]
03:55:36  2 of 6 OK created sql view model DEV.src_listings ............................. [SUCCESS 1 in 0.38s]
03:55:36  3 of 6 START sql view model DEV.src_reviews ................................... [RUN]
03:55:36  3 of 6 OK created sql view model DEV.src_reviews .............................. [SUCCESS 1 in 0.28s]
03:55:36  4 of 6 START sql table model DEV.dim_hosts_cleansed ........................... [RUN]
03:55:37  4 of 6 OK created sql table model DEV.dim_hosts_cleansed ...................... [SUCCESS 1 in 1.16s]
03:55:37  5 of 6 START sql table model DEV.dim_listings_cleansed ........................ [RUN]
03:55:38  5 of 6 OK created sql table model DEV.dim_listings_cleansed ................... [SUCCESS 1 in 0.97s]
03:55:38  6 of 6 START sql incremental model DEV.fct_reviews ............................ [RUN]
03:55:41  6 of 6 OK created sql incremental model DEV.fct_reviews ....................... [SUCCESS 1 in 2.75s]
03:55:41
03:55:41  Finished running 1 incremental model, 2 table models, 3 view models in 0 hours 0 minutes and 10.70 seconds (10
.70s).
03:55:41
03:55:41  Completed successfully
03:55:41
03:55:41  Done. PASS=6 WARN=0 ERROR=0 SKIP=0 TOTAL=6
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$
```

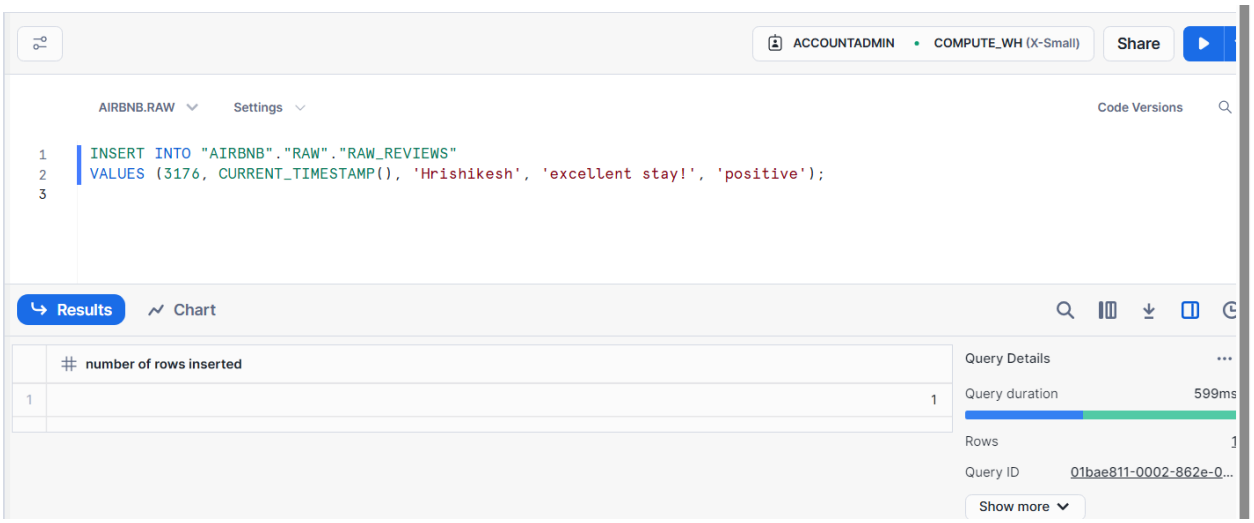



Now, we'll try to insert a review record to see if our dbt logic is working or not –

Retrieving data for a specific listing id = 3176




Now we'll, insert a data in it –




Even if we executed a insert statement, we wont be able to view it because we need to integrate with dbt by running ***dbt run*** command.

```
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$ dbt run
04:03:58  Running with dbt=1.9.3
04:04:02  Registered adapter: snowflake=1.9.0
04:04:21  Found 6 models, 472 macros
04:04:21
04:04:21  Concurrency: 1 threads (target='dev')
04:04:21
04:04:26  1 of 6 START sql view model DEV.src_hosts ..................................... [RUN]
04:04:26  1 of 6 OK created sql view model DEV.src_hosts ................................ [SUCCESS 1 in 0.43s]
04:04:26  2 of 6 START sql view model DEV.src_listings .................................. [RUN]
04:04:26  2 of 6 OK created sql view model DEV.src_listings .............................. [SUCCESS 1 in 0.30s]
04:04:26  3 of 6 START sql view model DEV.src_reviews .................................... [RUN]
04:04:27  3 of 6 OK created sql view model DEV.src_reviews .............................. [SUCCESS 1 in 0.32s]
04:04:27  4 of 6 START sql table model DEV.dim_hosts_cleansed ............................ [RUN]
04:04:28  4 of 6 OK created sql table model DEV.dim_hosts_cleansed ....................... [SUCCESS 1 in 1.07s]
04:04:28  5 of 6 START sql table model DEV.dim_listings_cleansed ......................... [RUN]
04:04:29  5 of 6 OK created sql table model DEV.dim_listings_cleansed .................... [SUCCESS 1 in 0.92s]
04:04:29  6 of 6 START sql incremental model DEV.fct_reviews ............................. [RUN]
04:04:31  6 of 6 OK created sql incremental model DEV.fct_reviews ........................ [SUCCESS 1 in 1.88s]
04:04:31
04:04:31  Finished running 1 incremental model, 2 table models, 3 view models in 0 hours 0 minutes and 10.10 seconds (10
.10s).
04:04:31
04:04:31  Completed successfully
04:04:31
04:04:31  Done. PASS=6 WARN=0 ERROR=0 SKIP=0 TOTAL=6
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$
```

Explanation for this step –
When you mark a dbt model as **incremental**, dbt **only processes new or updated rows** each time you run it, instead of rebuilding the entire table or view from scratch. Here's a simple breakdown:

1. **Initial Run (No Table Yet)**
    - If the table doesn't exist, dbt **creates** it and loads **all** the data from your source (e.g., src_reviews).
    - After this run, your fact (incremental) table has the complete dataset up to that point in time.
2. **Subsequent Runs (Table Exists)**
    - dbt checks whether the table already exists. If it does, dbt applies the logic inside the {% if is_incremental() %} block to **filter only new or changed records**.
    - For example, if your SQL says:

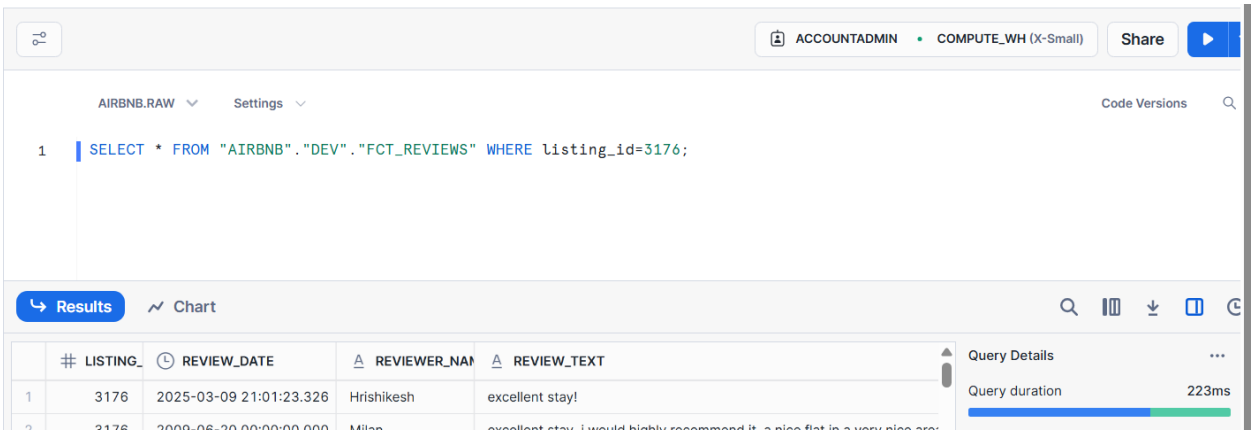
        *sql*

        *WHERE review_date > (SELECT MAX(review_date) FROM {{ this }})*

then dbt **only** pulls rows that have a review_date later than the latest date already in your incremental table.

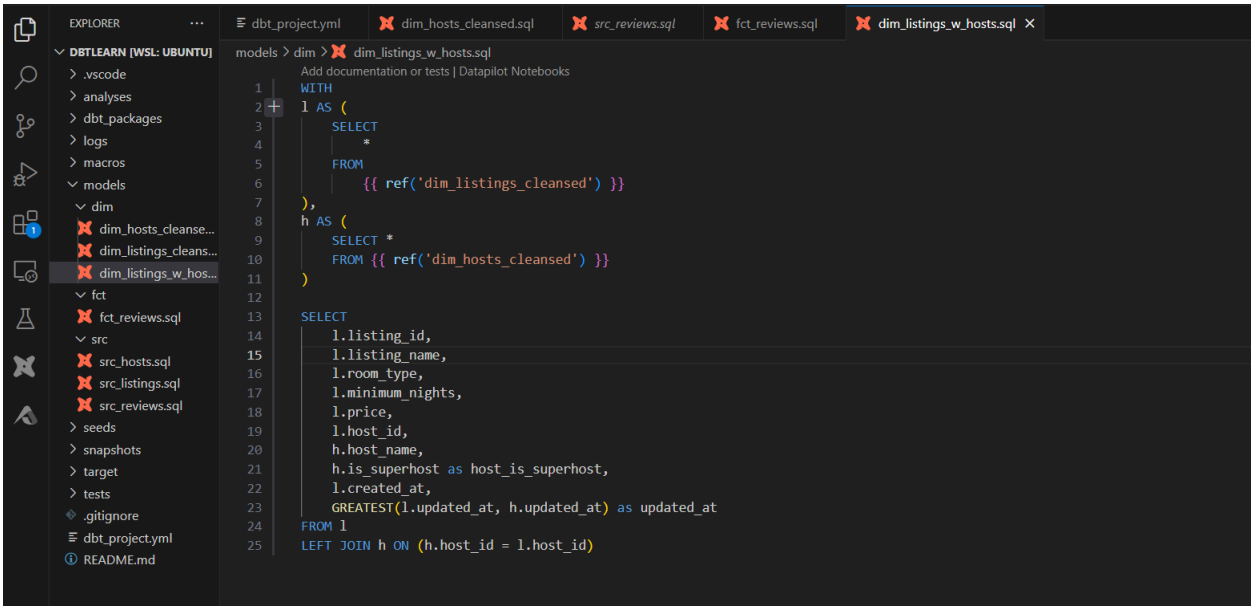
3. **Result**
    - Each time you run dbt run, your incremental table is **updated** with **just the new data** (like the row you inserted with your name "hrishikesh").
    - Old rows aren't reprocessed or duplicated. This makes the load faster and more efficient, especially as your dataset grows.

So, when you insert a new row into AIRBNB.RAW.RAW_REVIEWS and then run dbt run, dbt sees that there's a new row with a more recent review_date, and **just adds that row** to your incremental table/model. That's why you only see the newly inserted row get added, rather than reloading the entire table every time.

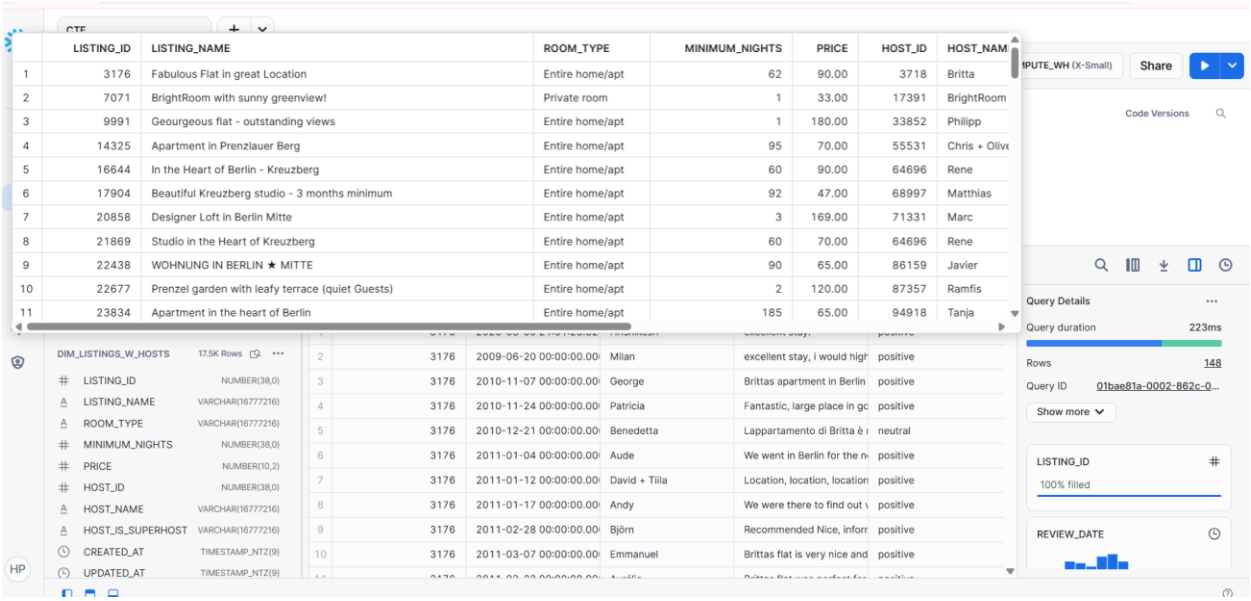



To refresh the entire build you can use – ***dbt run –full-refresh***

Creating dim_listings_w_hosts.sql which will contain hosts joined with their particular listings in listsings table. Also, we are now treating dim_listings_cleansed and dim_hosts_cleansed as view instead of tables. And src_hosts,src_listsings and src_reviews will be CTES as they wont be much accessed.

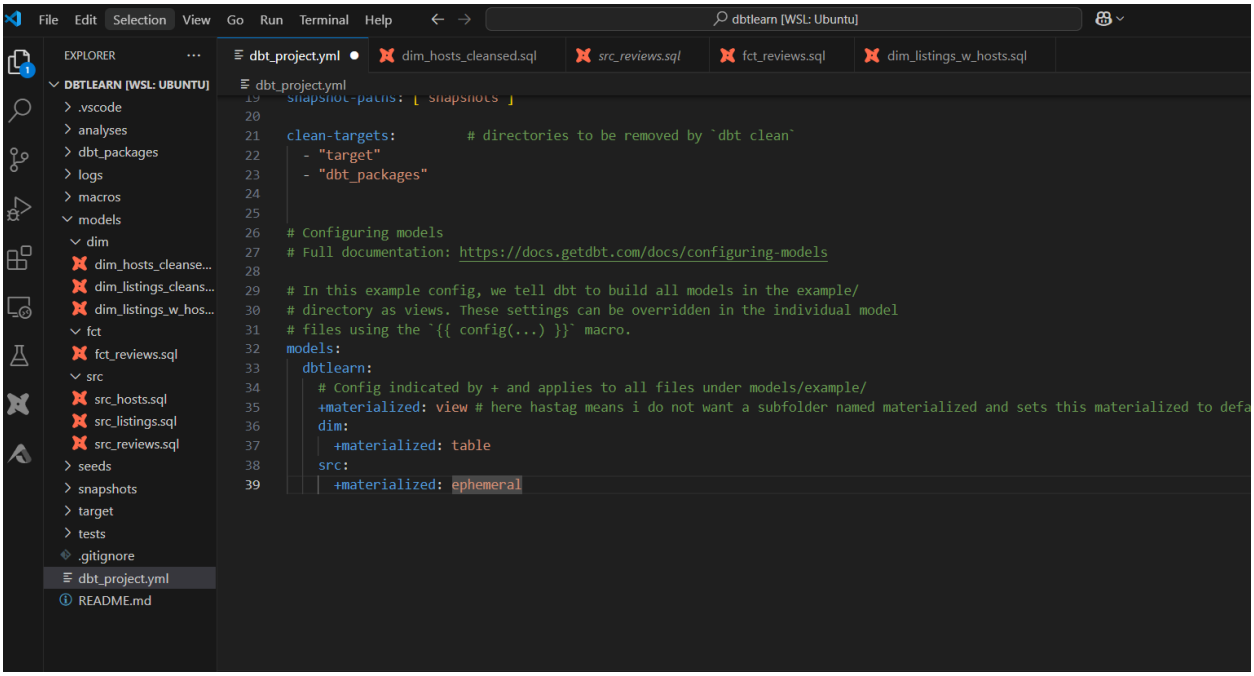


Ran ***dbt run*** command




Now, we'll need to clean up the materialization for all the dimensions which we did earlier -

1. In the source layer we don't need anything to be materialized. In dbt, an ephemeral model is a special type of model that does not create a table or view in your data warehouse. Instead, dbt inlines the ephemeral model's SQL into any downstream model that references it. Think of it like a subselect or CTE that's embedded directly into the final query rather than existing as its own object in the database.



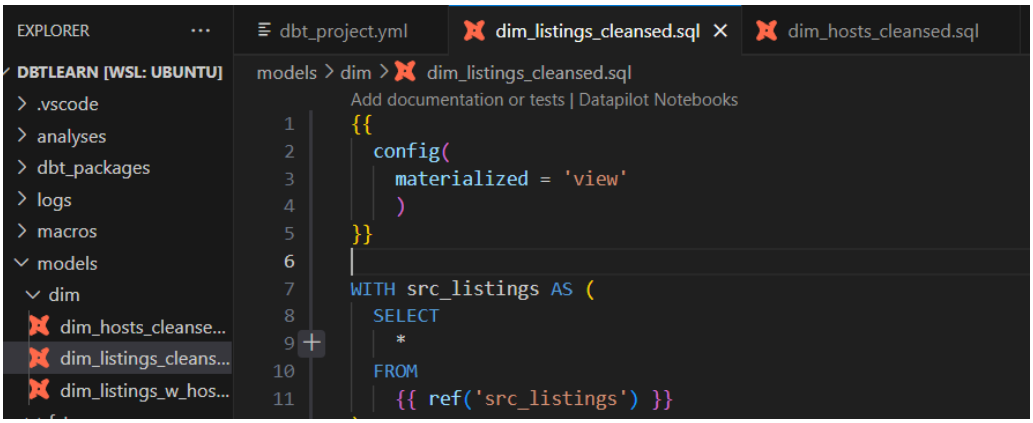

This will create every model in src as CTEs rather than views.
- DBT itself wont drop the views automatically so we need to drop them explicitly.

NOTE – if anything goes wrong then the dbt target folder is the place where you debug things which were runned.

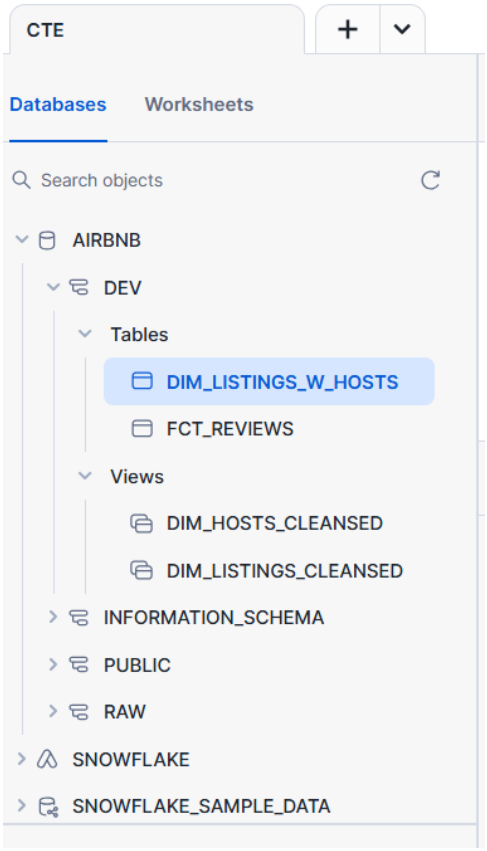
2. Turn dim_listings_cleansed.sql and dim_hosts_cleansed.sql to materialization = tables. What we did earlier was we mentioned in the project.yml file that the dim models will be materialized as tables. But, now we are changing the materialization explicitly to *'view'*.
Earlier -

```
# files using the `{{ config(...) }}` macro.
models:
  dbtlearn:
    # Config indicated by + and applies to all
    +materialized: view # here hastag means i
    dim:
      +materialized: table
    src:
      +materialized: ephemeral
```

now explicitly mentioning both the dim models-




```
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$ dbt run
03:50:48  Running with dbt=1.9.3
03:50:53  Registered adapter: snowflake=1.9.0
03:51:13  Found 7 models, 472 macros
03:51:13
03:51:13  Concurrency: 1 threads (target='dev')
03:51:13
03:51:19  1 of 4 START sql view model DEV.dim_hosts_cleansed ............................. [RUN]
03:51:19  1 of 4 OK created sql view model DEV.dim_hosts_cleansed ........................ [SUCCESS 1 in 0.58s]
03:51:19  2 of 4 START sql view model DEV.dim_listings_cleansed .......................... [RUN]
03:51:20  2 of 4 OK created sql view model DEV.dim_listings_cleansed ..................... [SUCCESS 1 in 0.63s]
03:51:20  3 of 4 START sql incremental model DEV.fct_reviews ............................. [RUN]
03:51:22  3 of 4 OK created sql incremental model DEV.fct_reviews ........................ [SUCCESS 0 in 1.89s]
03:51:22  4 of 4 START sql table model DEV.dim_listings_w_hosts .......................... [RUN]
03:51:23  4 of 4 OK created sql table model DEV.dim_listings_w_hosts ..................... [SUCCESS 1 in 1.64s]
03:51:24
03:51:24  Finished running 1 incremental model, 1 table model, 2 view models in 0 hours 0 minutes and 10.97 seconds (10.
97s).
03:51:24
03:51:24  Completed successfully
03:51:24
03:51:24  Done. PASS=4 WARN=0 ERROR=0 SKIP=0 TOTAL=4
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$
```


CTE
Databases
Worksheets
Search objects
AIRBNB
DEV
Tables
DIM_LISTINGS_W_HOSTS
FCT_REVIEWS
Views
DIM_HOSTS_CLEANSED
DIM_LISTINGS_CLEANSED
INFORMATION_SCHEMA
PUBLIC
RAW
SNOWFLAKE
SNOWFLAKE_SAMPLE_DATA

**SEEDS AND SOURCES** –

In dbt, **sources** and **seeds** are two ways to bring data into your project:

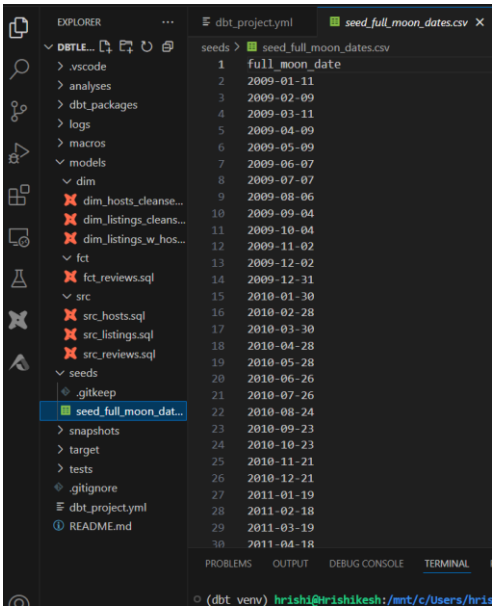
- **Sources** are your raw, existing data that's already in your data warehouse. Think of them as the starting ingredients that you want to clean up or combine.
- **Seeds** are CSV files you include in your dbt project. They're like small, ready-made datasets you commit to version control, and dbt can load them into your warehouse as tables for you to work with.

In short, sources are the external raw data you reference, and seeds are the CSV files you ship with your project to use as data.

Now, we'll import a dataset of Full Moon to draw a relation between bookings and the full moon night. This will be imported as seeds.

```
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$ curl https://dbtlearn.s3.us-east-2.am
azonaws.com/seed_full_moon_dates.csv -o seeds/seed_full_moon_dates.csv
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  3007  100  3007    0     0  14927      0 --:--:-- --:--:-- --:--:-- 14960
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$
```

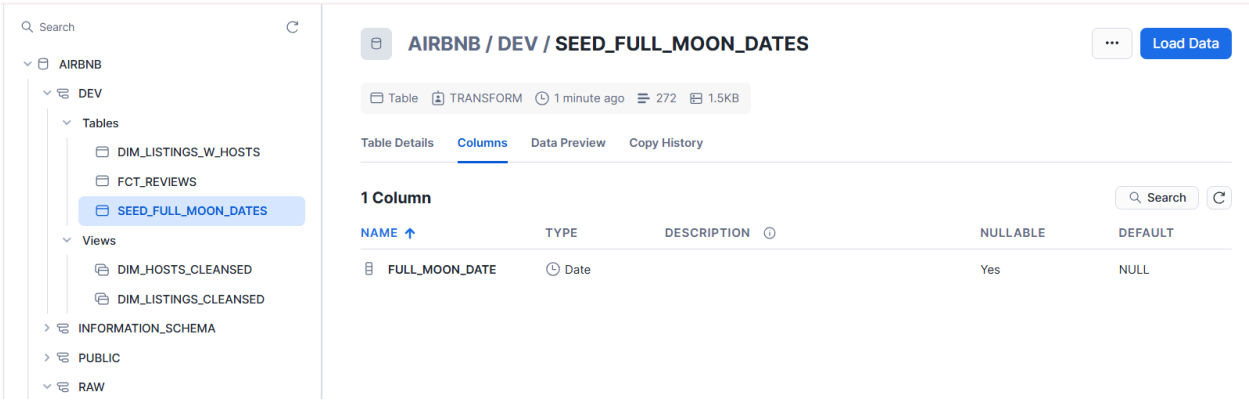
This command will directly copy the dataset into the seeds folder in my project directory.




Now, to populate the seeds with the snowflake use the below command –

```
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$ dbt seed
20:52:18  Running with dbt=1.9.3
20:52:21  Registered adapter: snowflake=1.9.0
20:52:37  Found 7 models, 1 seed, 472 macros
20:52:37
20:52:37  Concurrency: 1 threads (target='dev')
20:52:37
20:52:42  1 of 1 START seed file DEV.seed_full_moon_dates ............................... [RUN]
20:52:43  1 of 1 OK loaded seed file DEV.seed_full_moon_dates ........................... [INSERT 272 in 1.90s]
20:52:44
20:52:44  Finished running 1 seed in 0 hours 0 minutes and 6.86 seconds (6.86s).
20:52:44
20:52:44  Completed successfully
20:52:44
20:52:44  Done. PASS=1 WARN=0 ERROR=0 SKIP=0 TOTAL=1
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$
```

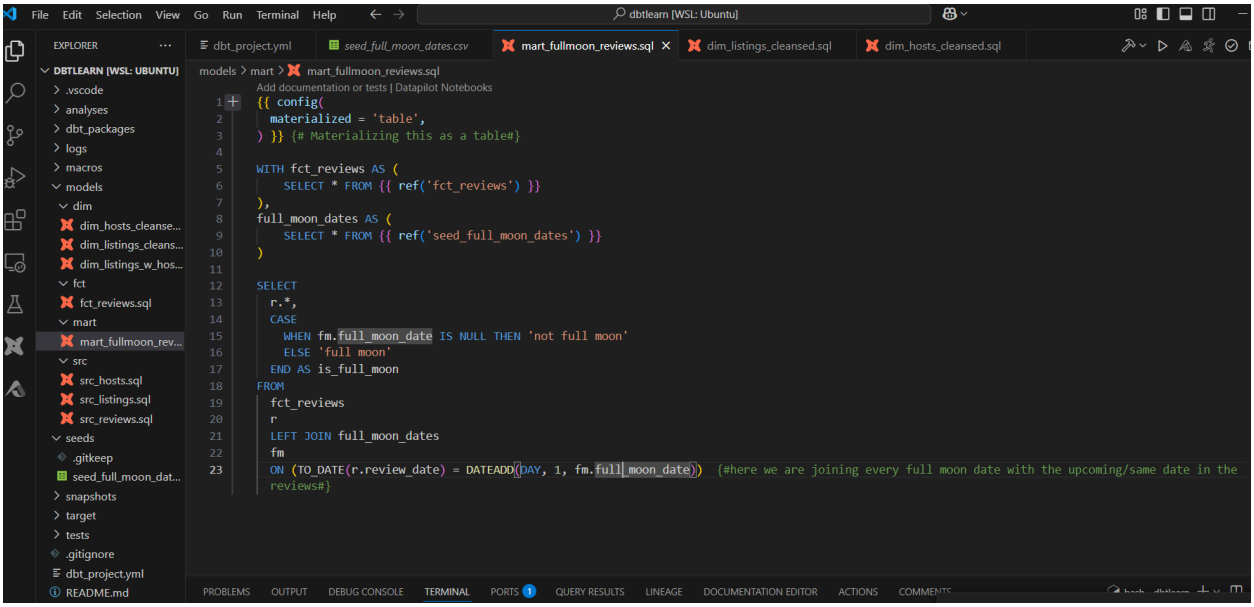
With dbt seed command it also gets integrated with the snowflake just like dbt run




The seed full moon dates has been added as a tables and the column has been automatically recognized as date time.

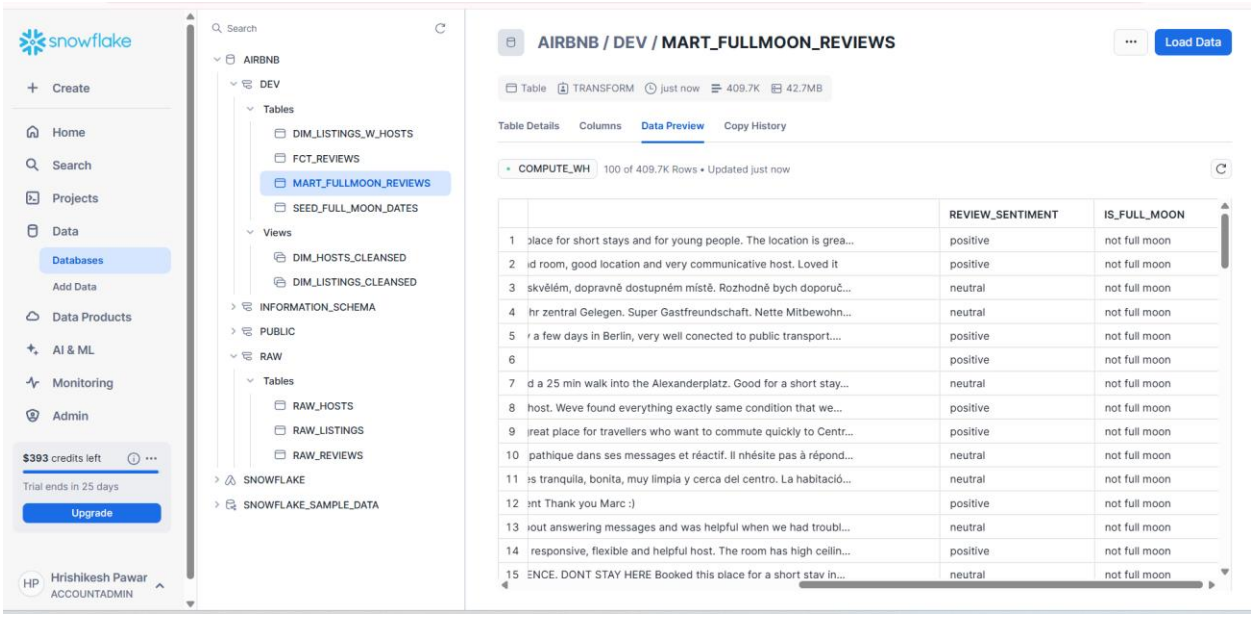
Now, we'll create the mart layer which will be accessible to our bi tools

- **NOTE** - We are taking full moon in consideration because we are proving a hypothesis that full moon has bad moods on people in night and that may affect the bookings.



```
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$ dbt run
21:04:10  Running with dbt=1.9.3
21:04:14  Registered adapter: snowflake=1.9.0
21:04:29  Found 8 models, 1 seed, 472 macros
21:04:29
21:04:29  Concurrency: 1 threads (target='dev')
21:04:29
21:04:34  1 of 5 START sql view model DEV.dim_hosts_cleansed ............................. [RUN]
21:04:35  1 of 5 OK created sql view model DEV.dim_hosts_cleansed ........................ [SUCCESS 1 in 0.44s]
21:04:35  2 of 5 START sql view model DEV.dim_listings_cleansed .......................... [RUN]
21:04:35  2 of 5 OK created sql view model DEV.dim_listings_cleansed ..................... [SUCCESS 1 in 0.56s]
21:04:35  3 of 5 START sql incremental model DEV.fct_reviews ............................. [RUN]
21:04:38  3 of 5 OK created sql incremental model DEV.fct_reviews ........................ [SUCCESS 0 in 2.50s]
21:04:38  4 of 5 START sql table model DEV.dim_listings_w_hosts .......................... [RUN]
21:04:40  4 of 5 OK created sql table model DEV.dim_listings_w_hosts ..................... [SUCCESS 1 in 1.92s]
21:04:40  5 of 5 START sql table model DEV.mart_fullmoon_reviews ......................... [RUN]
21:04:43  5 of 5 OK created sql table model DEV.mart_fullmoon_reviews .................... [SUCCESS 1 in 3.87s]
21:04:44
21:04:44  Finished running 1 incremental model, 2 table models, 2 view models in 0 hours 0 minutes and 14.04 seconds (14
.04s).
21:04:44
21:04:44  Completed successfully
21:04:44
21:04:44  Done. PASS=5 WARN=0 ERROR=0 SKIP=0 TOTAL=5
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$
```

This model is configured to be materialized as a table. It starts by defining two common table expressions: one that pulls all records from the existing "fct_reviews" table and another that retrieves dates from the "seed_full_moon_dates" CSV file. The main query then performs a left join between these two datasets, matching records where the review date (after converting to a date type) is equal to the full moon date plus one day. Finally, a new column named "is_full_moon" is added. This column is set to "full moon" when there is a matching full moon date, and "not full moon" when there is no match.



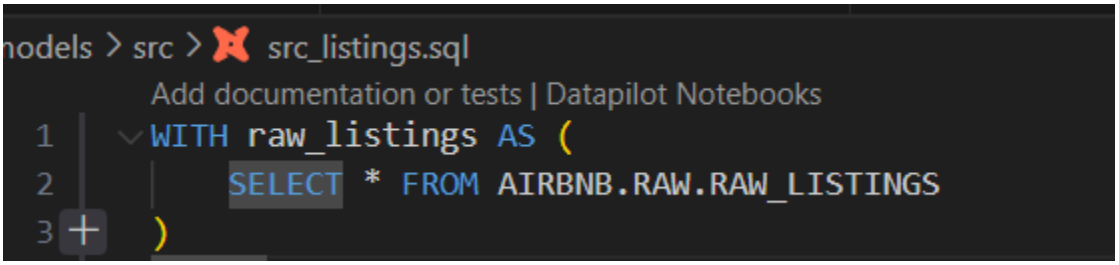

SOURCES

Now, We'll convert our raw tables into sources. Sources are abstractions on top of our input data. This will make our projects more structured.

Including sources in your dbt project offers several key advantages:

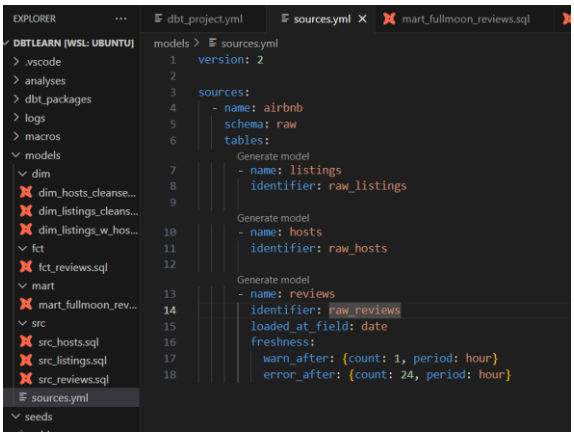
1. **Improved Transparency:**
   By defining sources, you document exactly where your raw data comes from. This clarity helps team members and stakeholders understand the origins of the data.
2. **Enhanced Data Quality:**
   Sources allow you to set up tests on raw data, ensuring that it meets quality standards before it's transformed. This early validation helps catch issues sooner.
3. **Easier Maintenance:**
   When you have a well-defined source, it becomes simpler to track changes in upstream data. Any changes in the raw data structure can be managed more effectively in your dbt models.
4. **Better Data Lineage:**
   Sources provide a clear map of data flow—from raw data to final transformed outputs. This lineage makes it easier to trace errors or understand how specific data points are derived.
5. **Facilitated Collaboration:**
   With sources clearly defined, teams have a common reference point for data. This shared understanding promotes consistency and reduces ambiguity across the project.

In summary, including sources in your data project enhances clarity, quality, maintainability, and collaboration by providing a well-documented starting point for your data transformations.




Now the above cte statement had direct reference to the raw layer of data.

With sources –



You can give reference to your source layer. This is done so that the direct reference of AIRBNB.RAW.RAW_LISTINGS is changed in future or if the schema changes then the query will throw an error. To avoid this we can use source-reference so that the code will be reading the source.yml and it will then get the identifier and refer the correct table needed for the query.

models > src > src_listings.sql

Add documentation or tests | Datapilot Notebooks

```sql
-- import raw listings
WITH raw_listings AS (
    SELECT * FROM {{ source('airbnb', 'listings') }}
)
SELECT
    id AS listing_id
```

Similarly, this can be done with hosts and reviews

If dbt compiled successfully without showing any error messages or warnings, then your project is syntactically valid and can be compiled. However, “dbt compile” doesn’t necessarily confirm that everything is fully correct or functional—only that dbt can parse your models and create the compiled SQL.

```
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$ dbt compile
22:05:15  Running with dbt=1.9.3
22:05:19  Registered adapter: snowflake=1.9.0
22:05:35  Found 8 models, 1 seed, 3 sources, 472 macros
22:05:35
22:05:35  Concurrency: 1 threads (target='dev')
22:05:35
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$
```

**NOTE** - In the ***source.yml*** code file, we also had something called as source freshness. This source definition specifies freshness thresholds for the “reviews” source. It uses the “loaded_at_field” called “date” to check when data was last updated. If the data is more than one hour old, dbt will issue a warning. If the data goes beyond 24 hours without being updated, dbt will raise an error.
we’ll use the ***dbt source freshness*** command to check for the source based on our policy

```
hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$ dbt source freshness
Running with dbt=1.9.3
Registered adapter: snowflake=1.9.0
Found 8 models, 1 seed, 3 sources, 472 macros

Concurrency: 1 threads (target='dev')

Pulling freshness from warehouse metadata tables for 0 sources
1 of 1 START freshness of airbnb.reviews ..................................... [RUN]
1 of 1 ERROR STALE freshness of airbnb.reviews ................................ [ERROR STALE in 0.58s]

Finished running 1 source in 0 hours 0 minutes and 4.88 seconds (4.88s).
Done.
hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$
```

We got an error because it’s more than 24hrs since our source was updated.

**SNAPSHOTS** - type 2 SCD

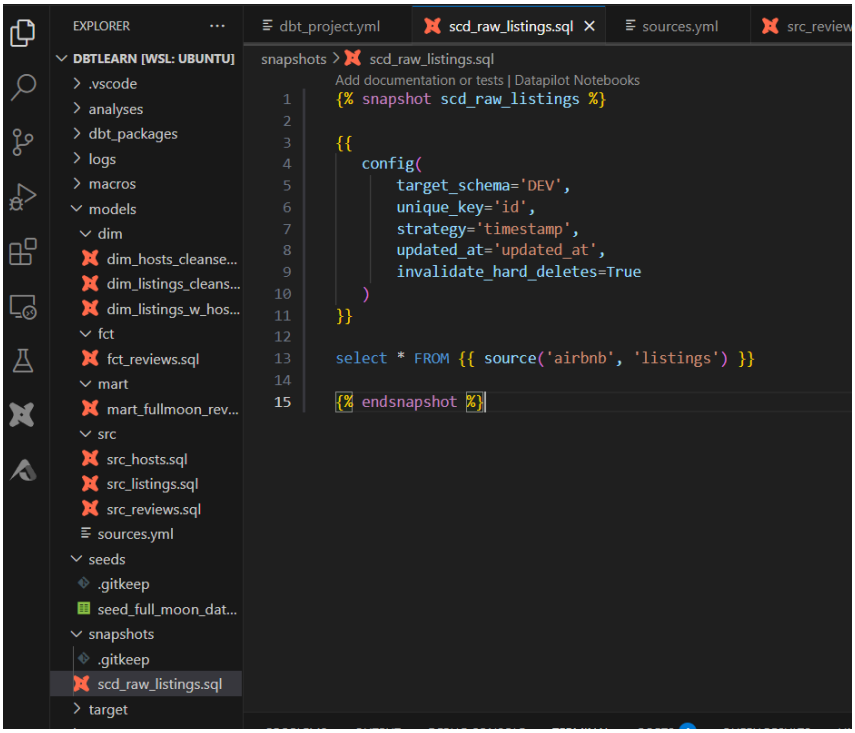
**CONFIGURATION AND STRATEGIES**

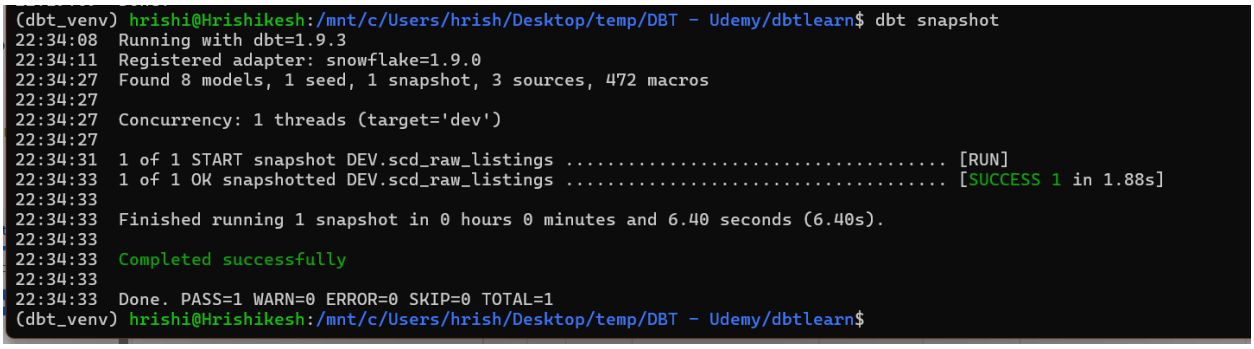
Snapshots live in the snapshots folder

**Strategies:**

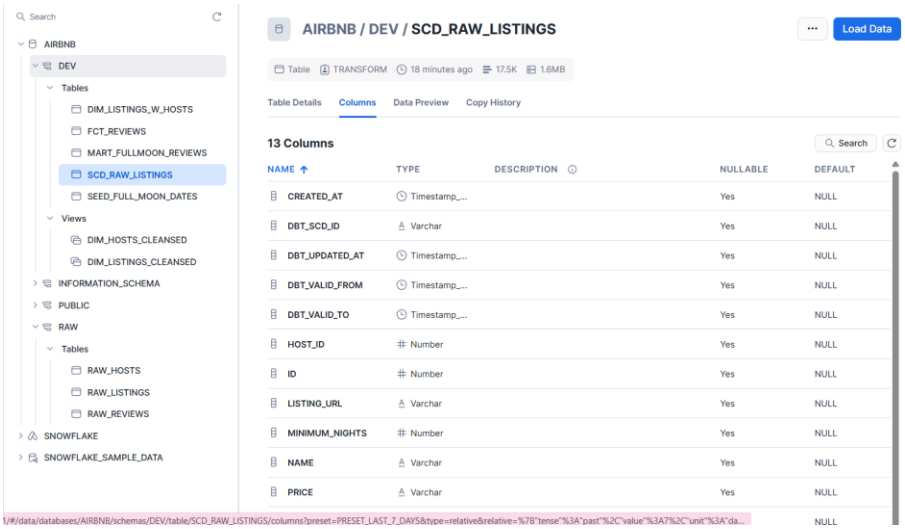
1. **Timestamp:** A unique key and an updated_at field is defined on the source model. These columns are used for determining changes.
2. **Check:** Any change in a set of columns (or all columns) will be picked up as an update.

Now, we'll be creating a total of two snapshots on top of raw_listings and raw_hosts

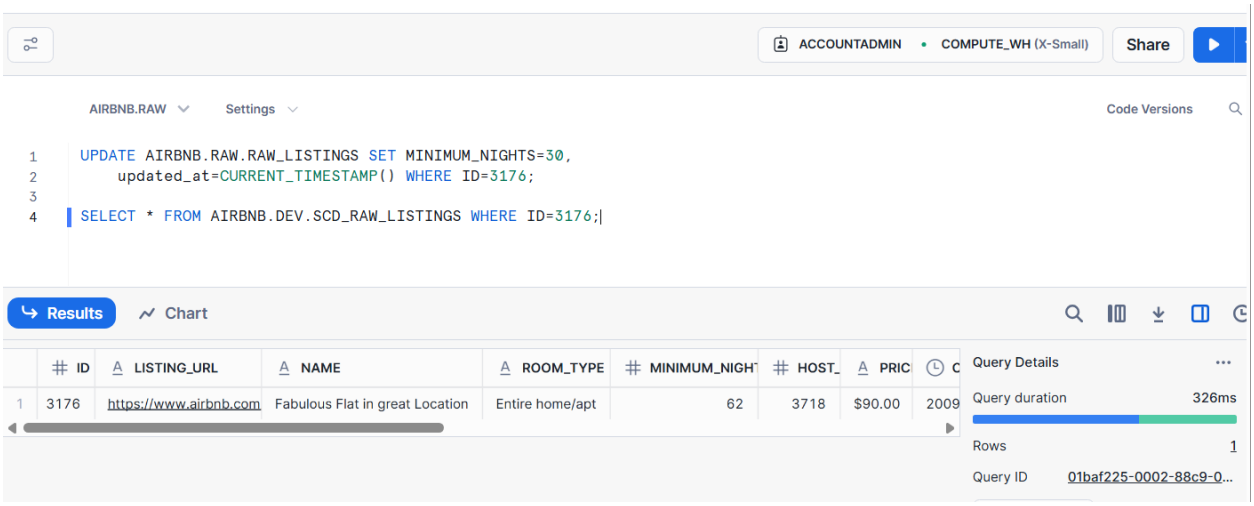


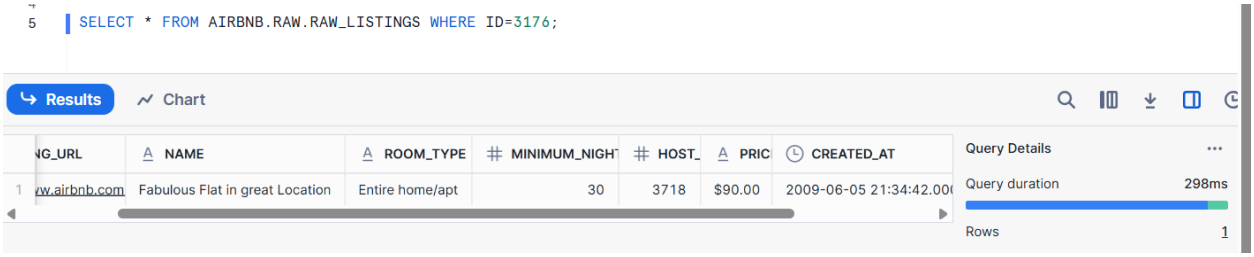

This will create a snapshot table in dev as mentioned in the .sql file above.



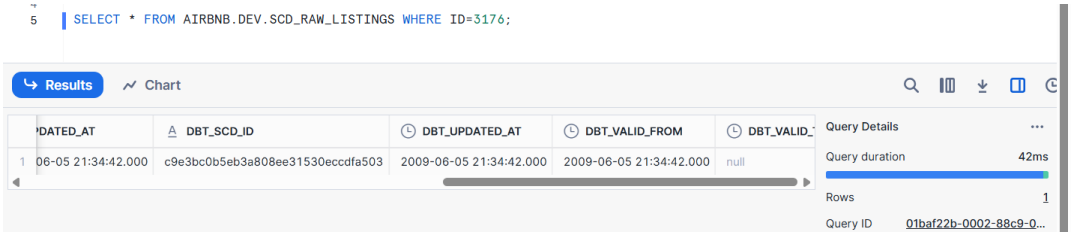

Now, we'll update manually to one of the record so that we can view if it works or not.



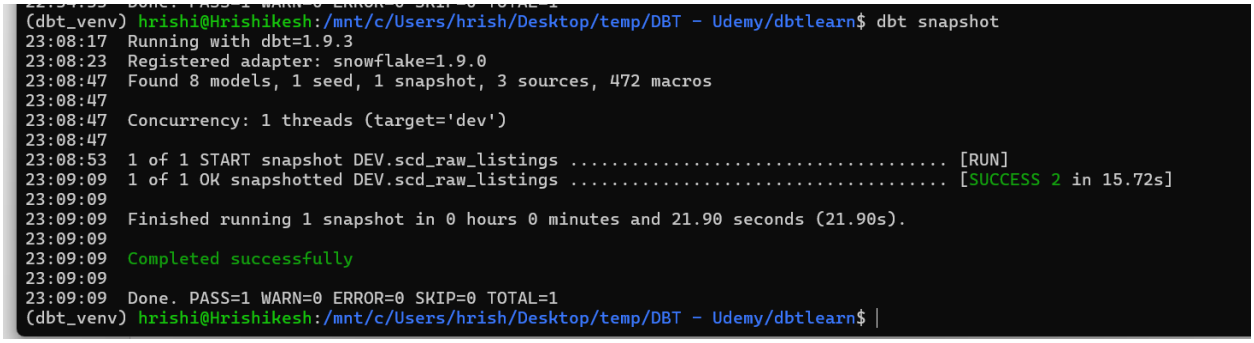

executing this query will update the record as mentioned for the record id = 3176

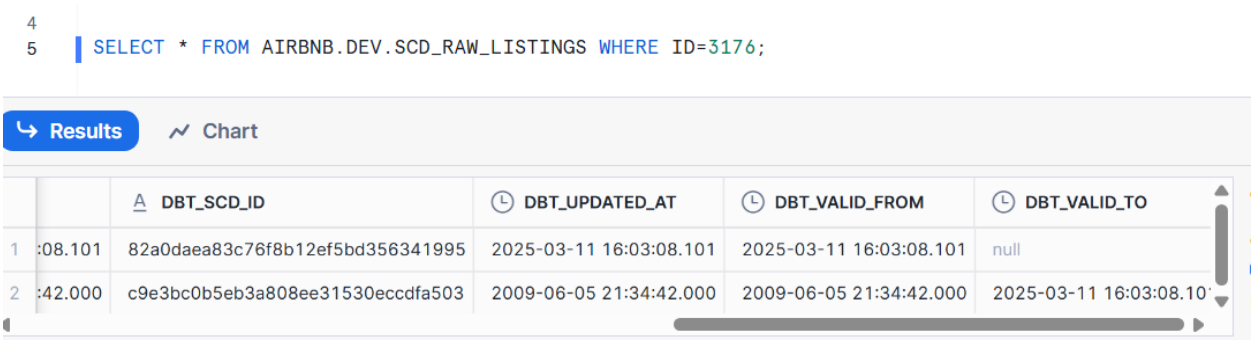



Now, we'll check for the snapshot in our scd_raw_listings.



This shows null because, We'll first need to execute dbt snapshot so that the dbt takes a snapshot of the updated data.




1st row is about the new value and 2nd is about old value.
Here's a simple way to think about each column in a **Slowly Changing Dimension (SCD)** table:

1. **updated_at** – Shows when the row was last updated in the database.
    - Every time you change something about that record, this timestamp is refreshed.
2. **valid_from** – Tells you when this particular version of the record became active.
    - If you see valid_from = 2009-05-20 15:31:42, that means this version of the record started to be true or "valid" at that moment.
3. **valid_to** – Tells you when this version of the record stopped being active.
    - If it's set to null, that means this version is currently the active one.
    - If it has a date/time, it means that at that exact moment, a newer version took over and this version became "inactive."

By looking at these three columns together, you can see exactly when each version was valid and what changed over time.

---

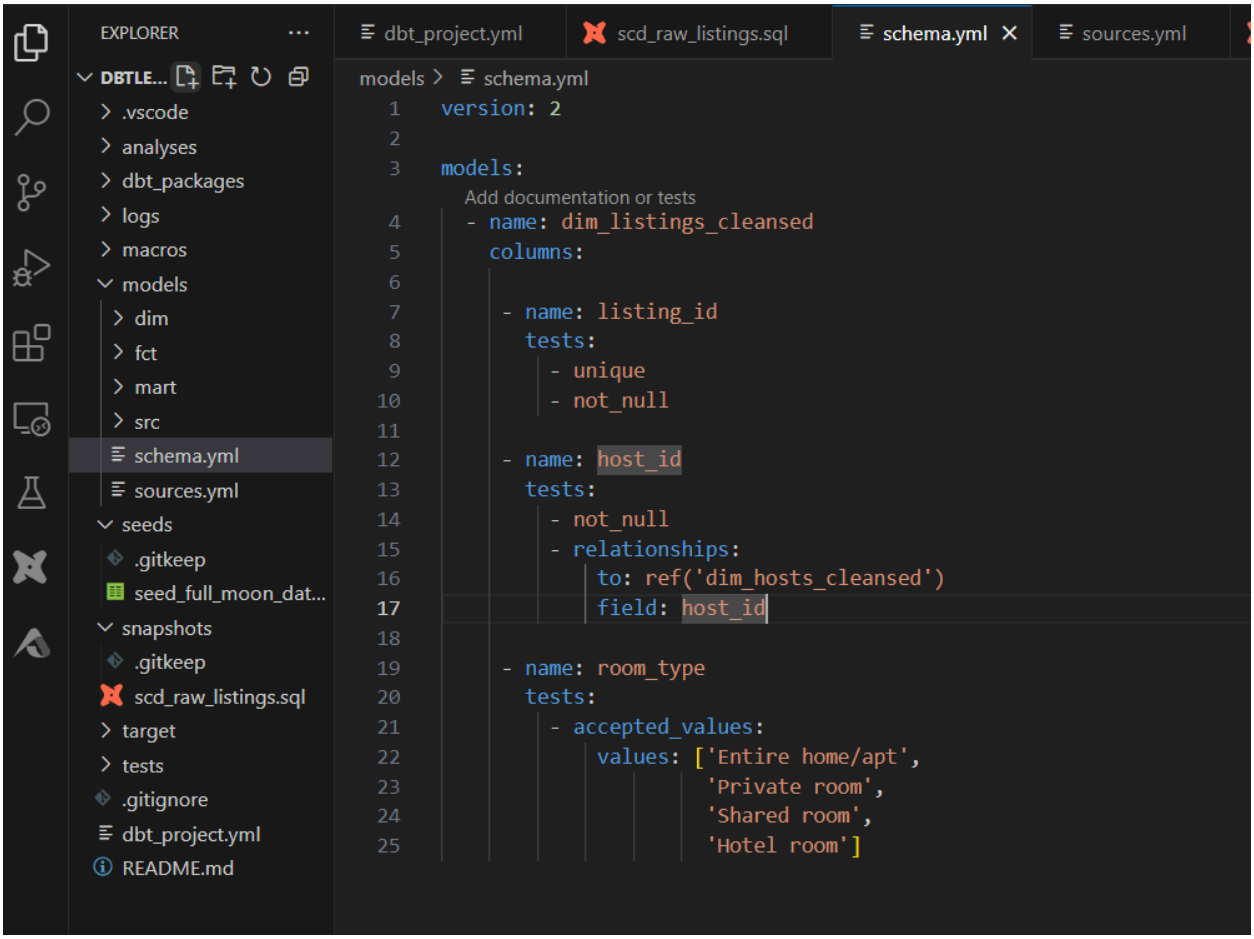
**TESTS** – Data quality tests and build tests

In dbt, **tests** help you validate the quality and integrity of your data. There are two main categories of tests you'll work with:

1. **Generic (Schema) Tests**
   These are built-in tests that you apply to columns in your models or sources by referencing them in your YAML files. They are "generic" because they follow a standard pattern and can be reused across different models. Common examples include:
   - **not_null:** Checks that a column has no missing (NULL) values.
   - **unique:** Ensures that every value in a column is distinct.
   - **accepted_values:** Makes sure a column's values are only from a specified set (like "US," "CA," "UK").
   - **relationships:** Verifies that each value in a column (like a foreign key) matches a corresponding value in another table (like a primary key).
2. **Custom (Data) Tests or SINGULAR TESTS**
   These tests are defined in SQL files (usually inside a tests directory). You write your own logic to check for whatever conditions matter most to your project. dbt expects your query to return zero rows if the test passes. For example, you might write a test to:
   - Compare the sum of sales in one table to the sum of sales in another table.
   - Verify that certain date ranges do not overlap.
   - Check that a metric stays within expected thresholds.

   In short, **generic tests** are quick and reusable for common checks on columns, while **custom tests** let you create more detailed checks specific to your data and business rules.

   So based on these, We'll build some generic tests using models/schema.yml –




This particular generic test is applied on dim_listings_cleansed table with column wise tests.

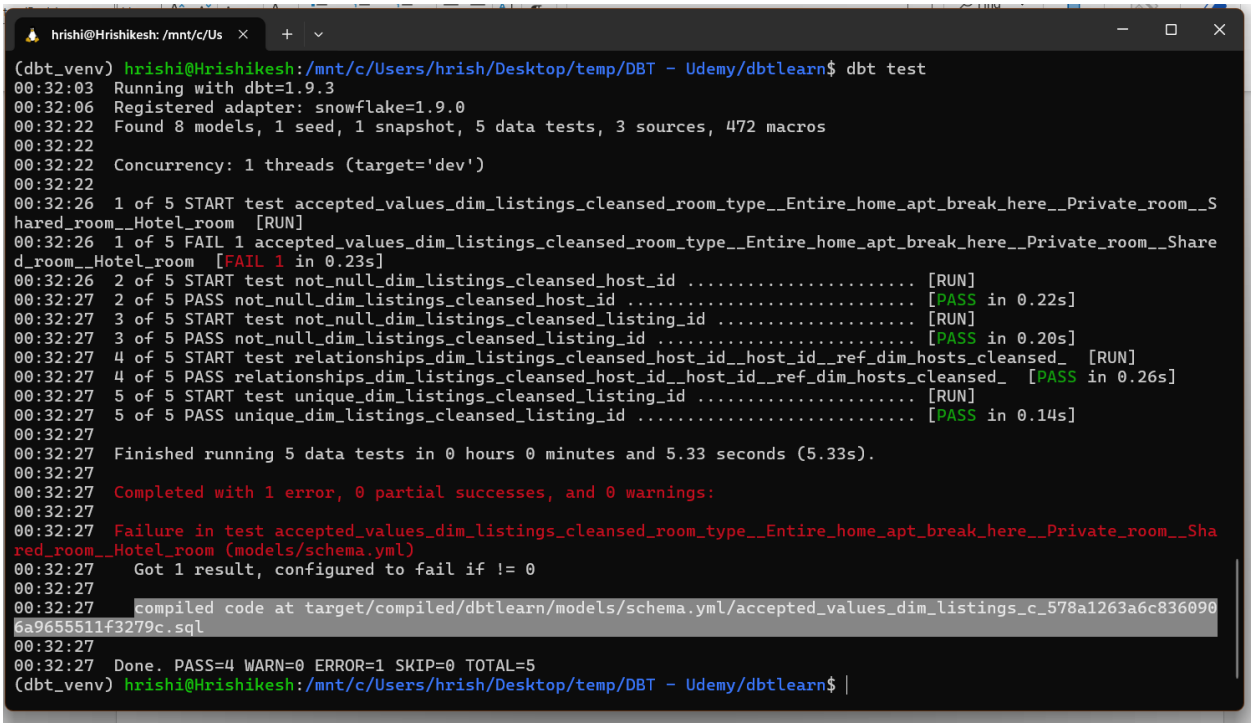
```
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$ dbt test
00:28:20  Running with dbt=1.9.3
00:28:25  Registered adapter: snowflake=1.9.0
00:28:42  Found 8 models, 1 seed, 1 snapshot, 5 data tests, 3 sources, 472 macros
00:28:42
00:28:42  Concurrency: 1 threads (target='dev')
00:28:42
00:28:46  1 of 5 START test accepted_values_dim_listings_cleansed_room_type__Entire_home_apt__Private_room__Shared_room_
_Hotel_room  [RUN]
00:28:47  1 of 5 PASS accepted_values_dim_listings_cleansed_room_type__Entire_home_apt__Private_room__Shared_room__Hotel
_room  [PASS in 0.91s]
00:28:47  2 of 5 START test not_null_dim_listings_cleansed_host_id ....................... [RUN]
00:28:48  2 of 5 PASS not_null_dim_listings_cleansed_host_id ............................. [PASS in 0.47s]
00:28:48  3 of 5 START test not_null_dim_listings_cleansed_listing_id .................... [RUN]
00:28:48  3 of 5 PASS not_null_dim_listings_cleansed_listing_id .......................... [PASS in 0.40s]
00:28:48  4 of 5 START test relationships_dim_listings_cleansed_host_id__host_id__ref_dim_hosts_cleansed_  [RUN]
00:28:49  4 of 5 PASS relationships_dim_listings_cleansed_host_id__host_id__ref_dim_hosts_cleansed_  [PASS in 0.79s]
00:28:49  5 of 5 START test unique_dim_listings_cleansed_listing_id ...................... [RUN]
00:28:49  5 of 5 PASS unique_dim_listings_cleansed_listing_id ............................ [PASS in 0.37s]
00:28:49
00:28:49  Finished running 5 data tests in 0 hours 0 minutes and 7.23 seconds (7.23s).
00:28:49
00:28:49  Completed successfully
00:28:49
00:28:49  Done. PASS=5 WARN=0 ERROR=0 SKIP=0 TOTAL=5
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$
```

The ‘pass’ shows that all the tests were successfully made.

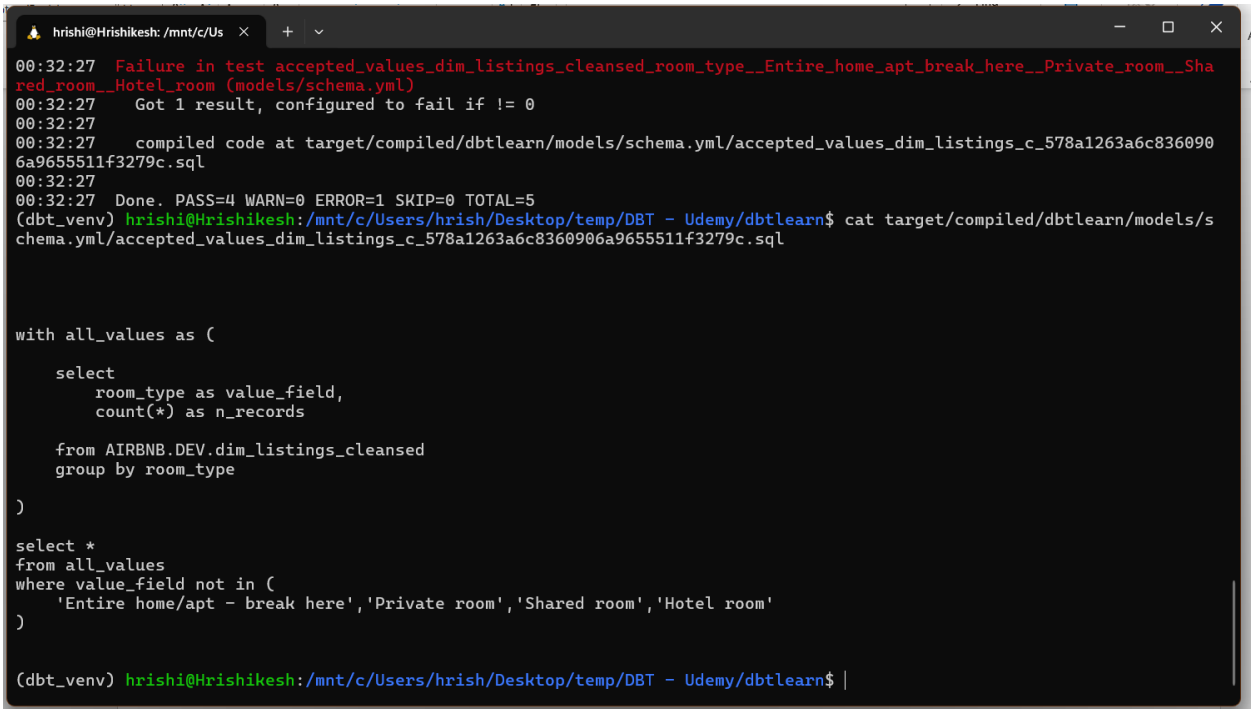
**NOW debugging if there’s any test that fails.**

```yaml
      - name: room_type
        tests:
          - accepted_values:
              values: ['Entire home/apt - break here',
                       'Private room',
                       'Shared room',
                       'Hotel room']
```

Made the changes for test in *room_type* column. This will **generate a failed test.**



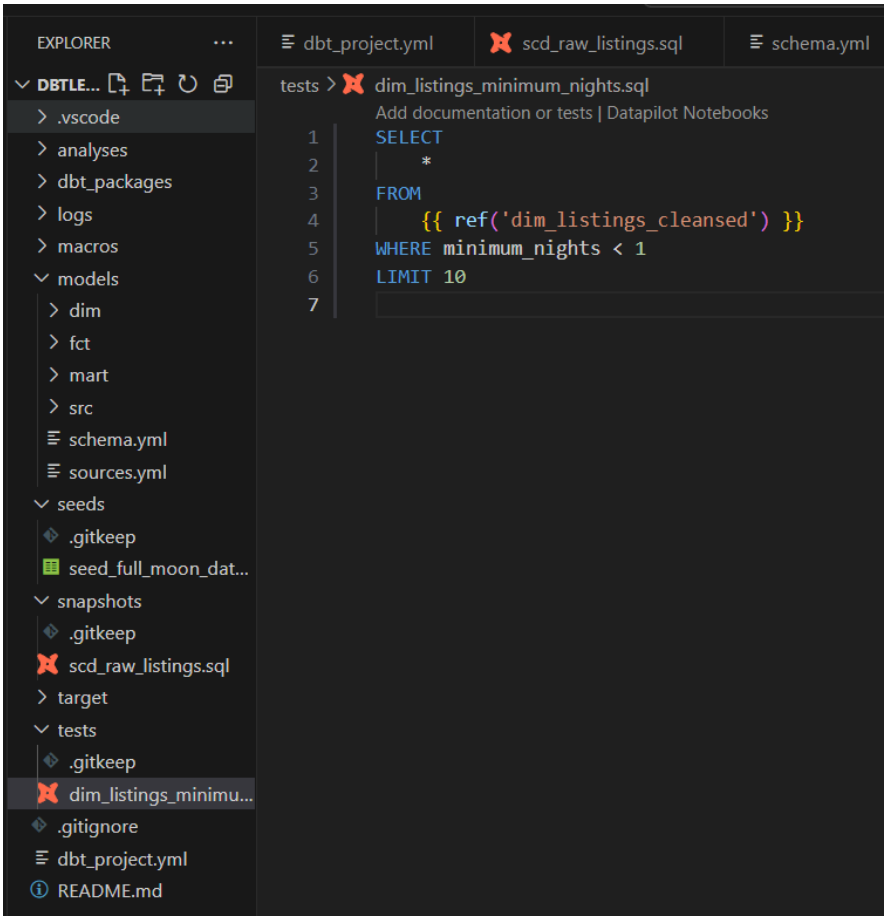

So, if you go at the highlighted file location you can view the compiled code for which the error was thrown.




This SQL code is **automatically generated** by dbt for an accepted_values test on the room_type column. Here's how it works:

1. **What It's Checking:**
   It ensures that every room_type value in the dim_listings_cleansed table is one of a specific set of allowed values:
   'Entire home/apt break here',
   'Private room',
   'Shared room',
   'Hotel room'
   If any record has a room_type outside of these values, the test flags it.
2. **How It Does This:**
   - The SQL selects room_type (renamed as value_field) from the dim_listings_cleansed table.
   - It applies a WHERE clause to filter rows that **do not** match any of the accepted values.
   - If the query returns any rows, it means those rows failed the accepted values check. dbt interprets these returned rows as a **test failure**.
3. **Outcome:**
   - If no rows are returned, the test **passes** (everything is within the accepted values).
   - **If there are rows returned**, the test **fails**, indicating some data does not match the expected room types.

Now, for **SINGULAR TEST** – which are just sql statements refer to the below screenshot. Since, earlier we cleansed the src_listings wherever there were minimum nights as 0. So, we'll run a test against it.



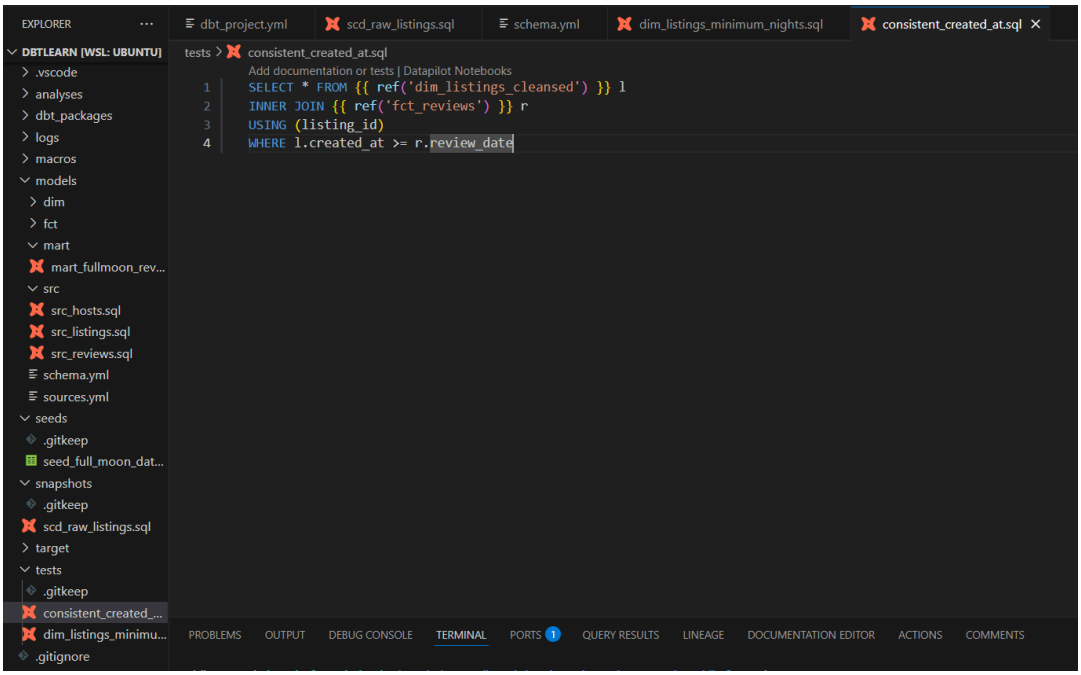

This shows there were no rows returned and hence the test passed.

```
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$ dbt test
00:54:12  Running with dbt=1.9.3
00:54:15  Registered adapter: snowflake=1.9.0
00:54:30  Found 8 models, 1 seed, 1 snapshot, 6 data tests, 3 sources, 472 macros
00:54:30
00:54:30  Concurrency: 1 threads (target='dev')
00:54:30
00:54:34  1 of 6 START test accepted_values_dim_listings_cleansed_room_type__Entire_home_apt__Private_room__
_Hotel_room  [RUN]
00:54:35  1 of 6 PASS accepted_values_dim_listings_cleansed_room_type__Entire_home_apt__Private_room__Shared
_room  [PASS in 0.21s]
00:54:35  2 of 6 START test dim_listings_minimum_nights ................................ [RUN]
00:54:35  2 of 6 PASS dim_listings_minimum_nights ..................................... [PASS in 0.65s]
00:54:35  3 of 6 START test not_null_dim_listings_cleansed_host_id ..................... [RUN]
00:54:35  3 of 6 PASS not_null_dim_listings_cleansed_host_id ........................... [PASS in 0.11s]
00:54:35  4 of 6 START test not_null_dim_listings_cleansed_listing_id .................. [RUN]
00:54:36  4 of 6 PASS not_null_dim_listings_cleansed_listing_id ........................ [PASS in 0.16s]
00:54:36  5 of 6 START test relationships_dim_listings_cleansed_host_id__host_id__ref_dim_hosts_cleansed_  [
00:54:36  5 of 6 PASS relationships_dim_listings_cleansed_host_id__host_id__ref_dim_hosts_cleansed_  [PASS i
00:54:36  6 of 6 START test unique_dim_listings_cleansed_listing_id ..................... [RUN]
00:54:36  6 of 6 PASS unique_dim_listings_cleansed_listing_id ........................... [PASS in 0.18s]
00:54:36
00:54:36  Finished running 6 data tests in 0 hours 0 minutes and 5.83 seconds (5.83s).
00:54:36
00:54:36  Completed successfully
00:54:36
00:54:36  Done. PASS=6 WARN=0 ERROR=0 SKIP=0 TOTAL=6
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$
```

Further, we can also perform test that is just concerned with just certain models only. Use this command – ***dbt test --select dim_listings_cleansed***

```
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$ dbt test --select dim_listings_cleansed
01:09:35  Running with dbt=1.9.3
01:09:39  Registered adapter: snowflake=1.9.0
01:09:54  Found 8 models, 1 seed, 1 snapshot, 6 data tests, 3 sources, 472 macros
01:09:54
01:09:54  Concurrency: 1 threads (target='dev')
01:09:54
01:09:58  1 of 6 START test accepted_values_dim_listings_cleansed_room_type__Entire_home_apt__Private_room__Shared_room__Hotel_room
  [RUN]
01:09:58  1 of 6 PASS accepted_values_dim_listings_cleansed_room_type__Entire_home_apt__Private_room__Shared_room__Hotel_room  [PAS
S in 0.23s]
01:09:58  2 of 6 START test dim_listings_minimum_nights ................................ [RUN]
01:09:58  2 of 6 PASS dim_listings_minimum_nights ..................................... [PASS in 0.13s]
01:09:58  3 of 6 START test not_null_dim_listings_cleansed_host_id ..................... [RUN]
01:09:58  3 of 6 PASS not_null_dim_listings_cleansed_host_id ........................... [PASS in 0.14s]
01:09:58  4 of 6 START test not_null_dim_listings_cleansed_listing_id .................. [RUN]
01:09:58  4 of 6 PASS not_null_dim_listings_cleansed_listing_id ........................ [PASS in 0.22s]
01:09:58  5 of 6 START test relationships_dim_listings_cleansed_host_id__host_id__ref_dim_hosts_cleansed_  [RUN]
01:09:59  5 of 6 PASS relationships_dim_listings_cleansed_host_id__host_id__ref_dim_hosts_cleansed_  [PASS in 0.13s]
01:09:59  6 of 6 START test unique_dim_listings_cleansed_listing_id .................... [RUN]
01:09:59  6 of 6 PASS unique_dim_listings_cleansed_listing_id .......................... [PASS in 0.14s]
01:09:59
01:09:59  Finished running 6 data tests in 0 hours 0 minutes and 5.00 seconds (5.00s).
01:09:59
01:09:59  Completed successfully
01:09:59
01:09:59  Done. PASS=6 WARN=0 ERROR=0 SKIP=0 TOTAL=6
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$
```

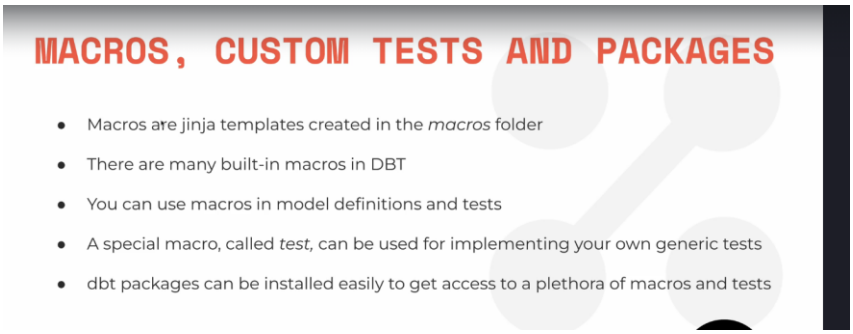
Create a singular test in tests/consistent_created_at.sql that checks that there is no review date that is submitted before its listing was created: Make sure that every review_date in fct_reviews is more recent than the associated created_at in dim_listings_cleansed.



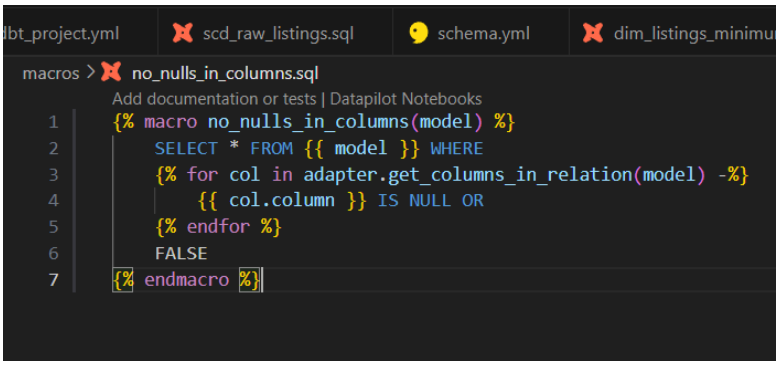

This query checks for the validation that the review is valid.

```
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$ dbt test --select path:tests/consistent_created_
at.sql
01:13:54  Running with dbt=1.9.3
01:13:57  Registered adapter: snowflake=1.9.0
01:14:12  Found 8 models, 1 seed, 1 snapshot, 7 data tests, 3 sources, 472 macros
01:14:12
01:14:12  Concurrency: 1 threads (target='dev')
01:14:12
01:14:16  1 of 1 START test consistent_created_at ..................................... [RUN]
01:14:18  1 of 1 PASS consistent_created_at ........................................... [PASS in 1.19s]
01:14:18
01:14:18  Finished running 1 test in 0 hours 0 minutes and 5.98 seconds (5.98s).
01:14:18
01:14:18  Completed successfully
01:14:18
01:14:18  Done. PASS=1 WARN=0 ERROR=0 SKIP=0 TOTAL=1
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$
```

MARCOS




Now, we create a macro under macro folder -




Here's a quick breakdown of what each part of this macro does:

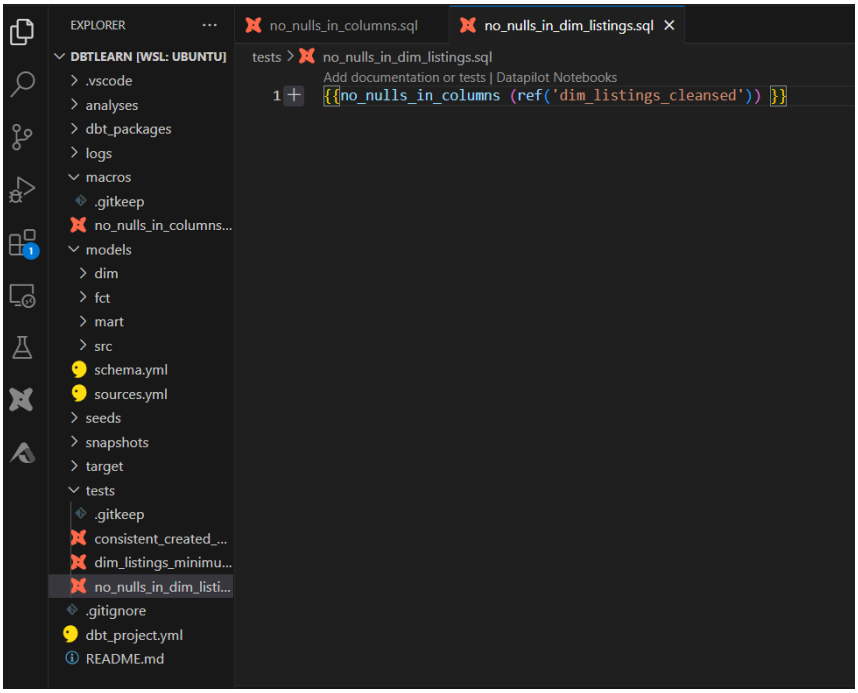
1. **{% macro no_nulls_in_columns(model) %}**
   - This starts the definition of a macro named no_nulls_in_columns that takes one parameter: model.
2. **SELECT * FROM {{ model }} WHERE**
   - This creates a SQL SELECT statement to retrieve all columns from the table (or model) passed into the macro.
   - {{ model }} gets replaced by the actual table name or reference when the macro runs.
3. **{% for col in adapter.get_columns_in_relation(model) %}**
   - This is a loop that goes through every column in the specified model.
   - adapter.get_columns_in_relation(model) is a built-in dbt function that returns a list of column objects for the given model.
4. **{{ col.column }} IS NULL OR**
   - For each column, this line adds a condition to check if that column is NULL.
   - The OR keyword means the query will look for rows where *any* column is NULL.
5. **{% endfor %}**
   - This marks the end of the for loop.
6. **FALSE**
   - After listing all column IS NULL OR conditions, the query ends with FALSE.

- This makes sure the SQL statement is valid by ending the WHERE clause logically (so you don't end with an extra OR).

7. **{% endmacro %}**
   - This ends the macro definition.

Overall, this macro builds a SQL query that returns rows where **any column** in the model is NULL. If no rows are returned, that means there are no null values in any columns.

Now, since the macro is created it can be executed as a singular test. So, we create a singular test in test and give the reference of model and the macro –

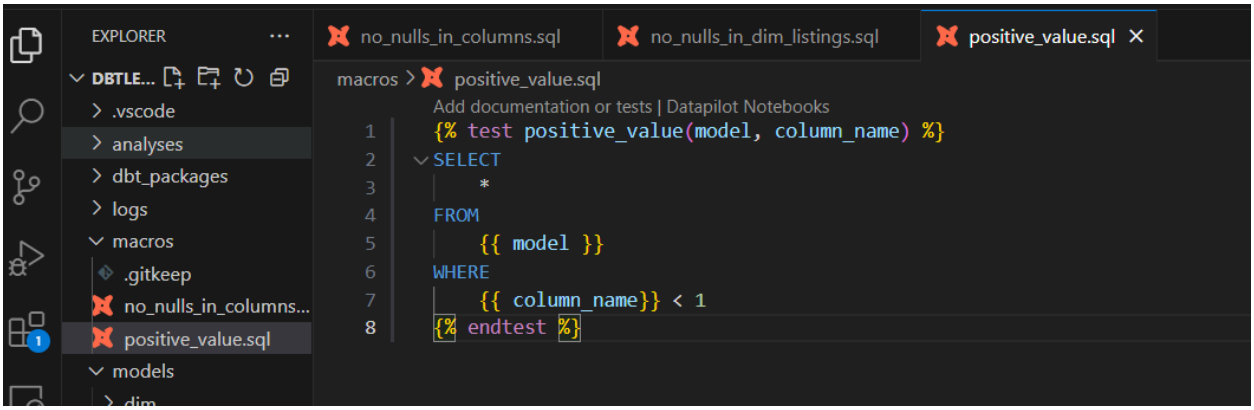



As you know, we can call a particular test on a specific model for which the test is to be runned as shown below. The highlighted text shows that the test has been passed successfully.

```
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$ dbt test --select dim_listings_cleans
ed
00:24:19  Running with dbt=1.9.3
00:24:23  Registered adapter: snowflake=1.9.0
00:24:40  Found 8 models, 1 seed, 1 snapshot, 8 data tests, 3 sources, 473 macros
00:24:40
00:24:40  Concurrency: 1 threads (target='dev')
00:24:40
00:24:46  1 of 8 START test accepted_values_dim_listings_cleansed_room_type__Entire_home_apt__Private_room__Shared_room_
_Hotel_room  [RUN]
00:24:47  1 of 8 PASS accepted_values_dim_listings_cleansed_room_type__Entire_home_apt__Private_room__Shared_room__Hotel
_room  [PASS in 1.21s]
00:24:47  2 of 8 START test consistent_created_at ..................................... [RUN]
00:24:48  2 of 8 PASS consistent_created_at ..................................... [PASS in 1.13s]
00:24:48  3 of 8 START test dim_listings_minimum_nights ..............................[RUN]
00:24:49  3 of 8 PASS dim_listings_minimum_nights ..................................... [PASS in 0.68s]
00:24:49  4 of 8 START test no_nulls_in_dim_listings ..................................[RUN]
00:24:50  4 of 8 PASS no_nulls_in_dim_listings ..................................... [PASS in 0.54s]
00:24:50  5 of 8 START test not_null_dim_listings_cleansed_host_id ....................[RUN]
00:24:50  5 of 8 PASS not_null_dim_listings_cleansed_host_id ..... [PASS in 0.25s]
00:24:50  6 of 8 START test not_null_dim_listings_cleansed_listing_id ..... [RUN]
00:24:50  6 of 8 PASS not_null_dim_listings_cleansed_listing_id ..... [PASS in 0.13s]
00:24:50  7 of 8 START test relationships_dim_listings_cleansed_host_id__host_id__ref_dim_hosts_cleansed_  [RUN]
00:24:51  7 of 8 PASS relationships_dim_listings_cleansed_host_id__host_id__ref_dim_hosts_cleansed_  [PASS in 0.53s]
00:24:51  8 of 8 START test unique_dim_listings_cleansed_listing_id ..... [RUN]
00:24:51  8 of 8 PASS unique_dim_listings_cleansed_listing_id ..... [PASS in 0.51s]
00:24:51
```

CUSTOM GENERIC TEST -
these are nothing but the macros with special signature. So, with these signatures attached to your macros you can use them as a generic tests to be performed in you project which is defined with the signatures in the dbt_project.yml file.

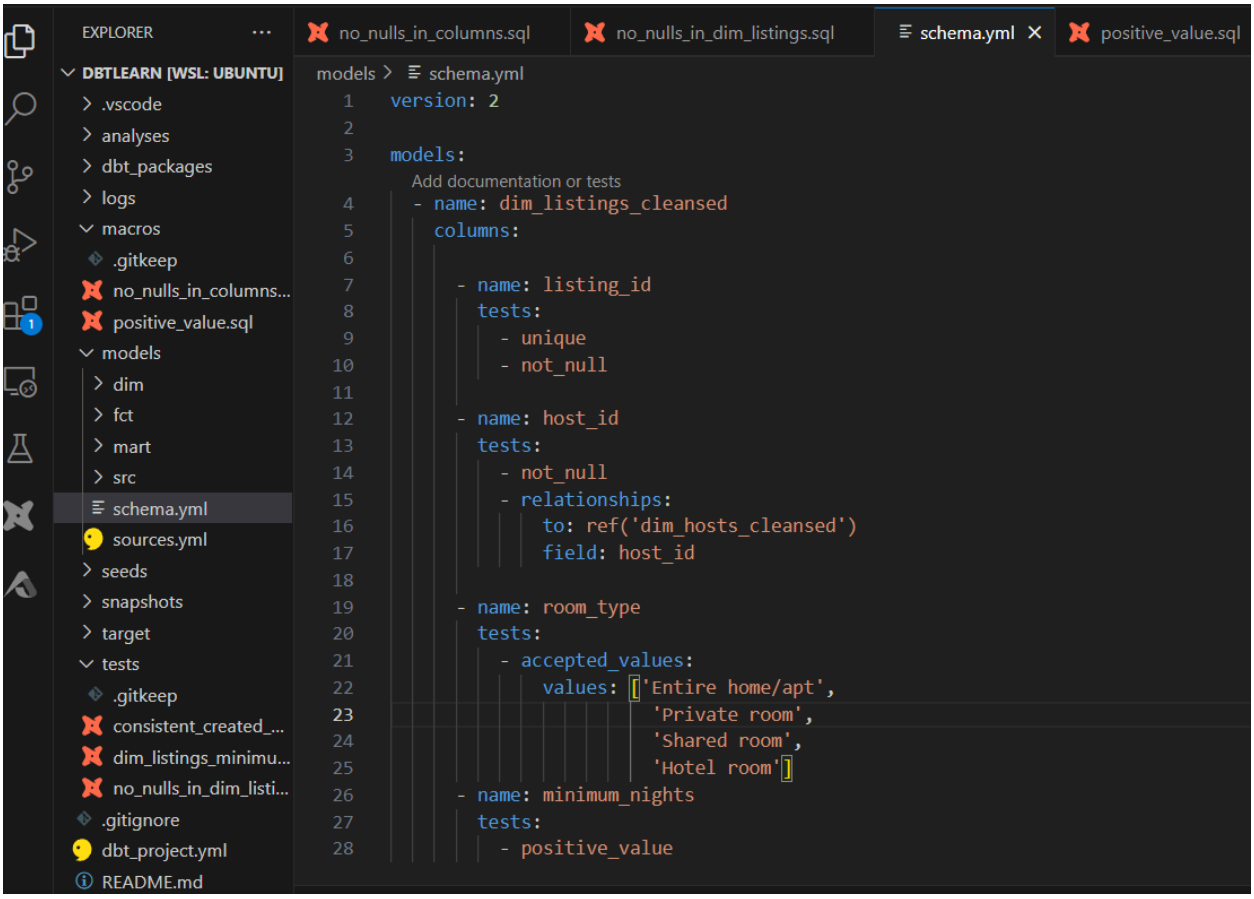
We'll be creating a generic test to check for dim_listings_cleansed column where the minimum night is set to 0 or not? As we did set a test earlier we'll not set it as a generic custom test. -




This is a custom dbt generic test called positive_value. It takes two arguments model (the table or view to test) and column_name (the specific column to check). The query selects rows where the specified column's values are **less than 1**. If any rows match that condition, the test fails, indicating that not all values in that column are positive. If no rows are returned, it means all values are at least 1, and the test passes

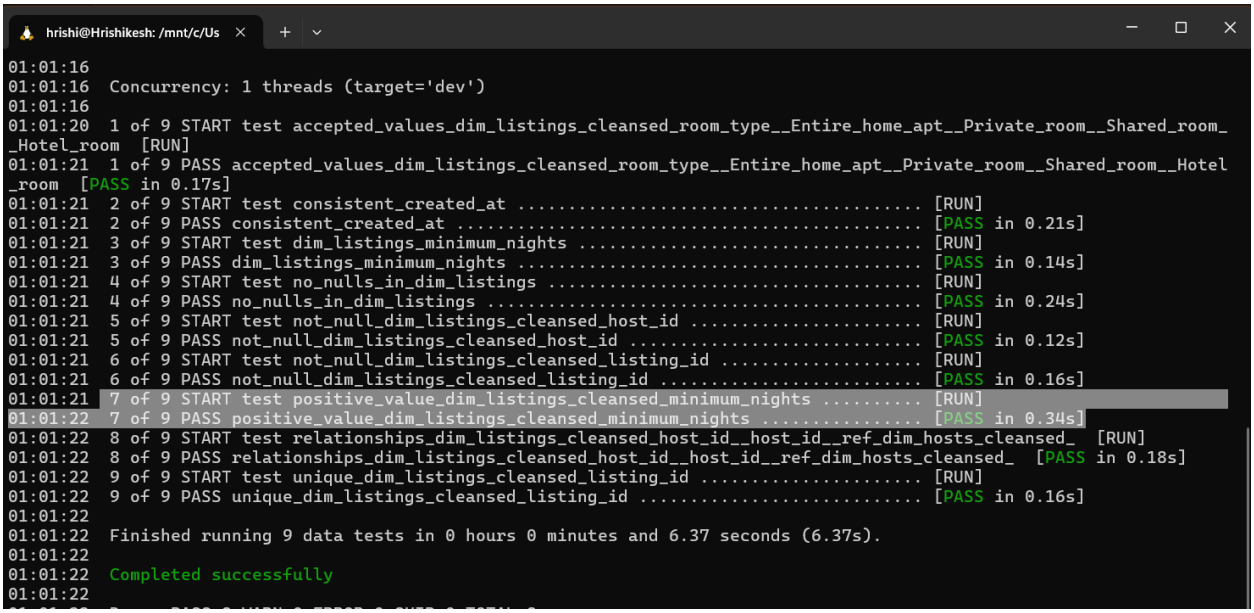
This test is created in the marcos folder.

Now, we'll mention this custom test in the schema.yml. Earlier the schema.yml was made to execute GENERIC_TESTS, now we'll just add out CUSTOM_GENERIC_TESTS to it.




name: minimum_nights = column name

tests: - positive_value = the name of the custom test that gets run on that column



**INSTALLING DBT PACKAGES**

To implement a package example we can look at fct_reviews as it has no unique row or values in the column so that it can be uniquely distinguished.

To tackle this problem, we'll be using hash functions –
Hash functions in **dbt-utils** let you generate a unique "fingerprint" for a piece of data. You might use them to:

1. **Anonymize** sensitive information (by storing a hash instead of the actual data).
2. **Compare** data quickly (if two hashes match, the underlying data is the same).

Essentially, these functions take in a value (like a text string) and return a scrambled, fixed-length output that can't be reversed to reveal the original value.

***Which is in short generating a (hashed) surrogate key.***

### generate_surrogate_key (source)

This macro implements a cross-database way to generate a hashed surrogate key using the fields specified.

**Usage:**

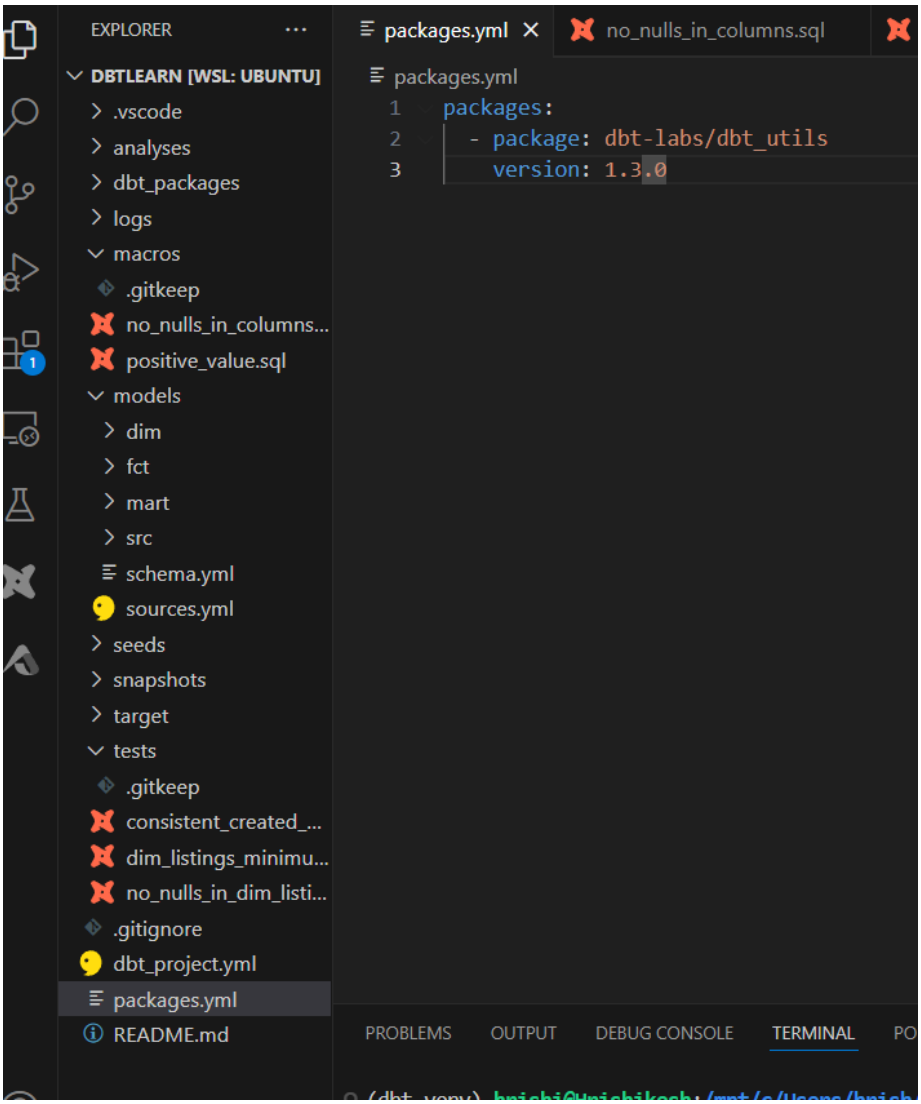
```
{{ dbt_utils.generate_surrogate_key(['field_a', 'field_b'[,...]]) }}
```

A precursor to this macro, `surrogate_key()`, treated nulls and blanks strings the same. If you need to enable this incorrect behaviour for backward compatibility reasons, add the following variable to your `dbt_project.yml`:

```
#dbt_project.yml
vars:
  surrogate_key_treat_nulls_as_empty_strings: true #turn on legacy behaviour
```

But first we'll need to install the dbt utils package

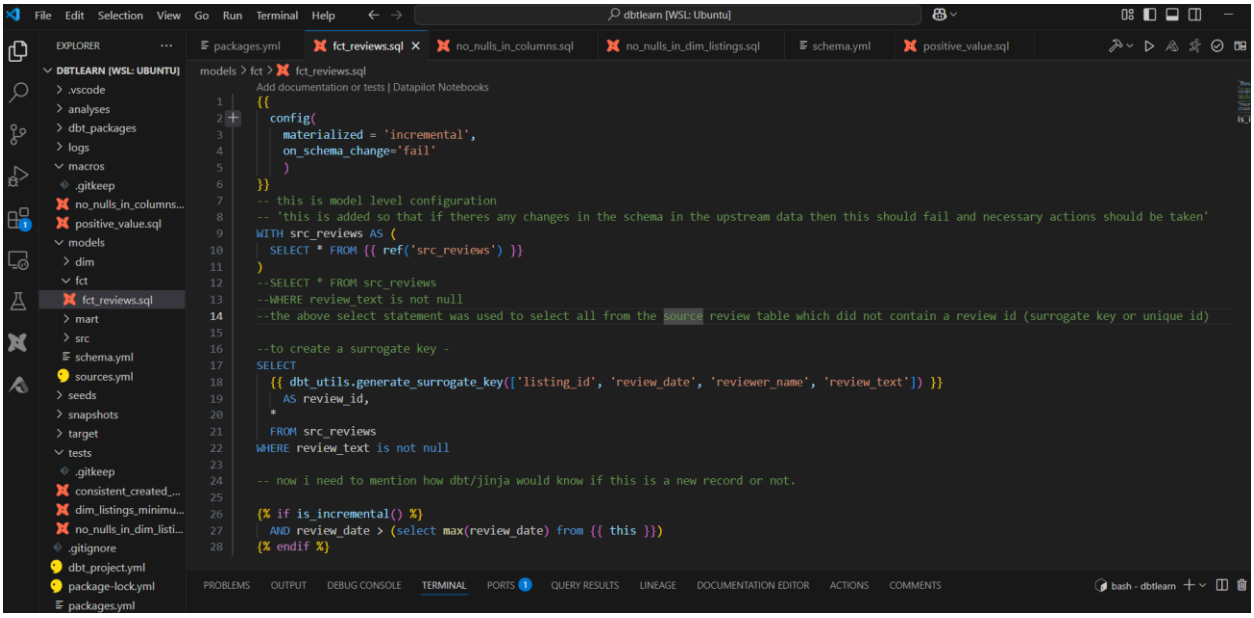
So to get a package in the project you need to create a yml file in root directory to explicitly mention the packages.




Now, run dbt folder – ***dbt deps***

```
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$ dbt deps
01:49:29  Running with dbt=1.9.3
01:49:31  Updating lock file in file path: /mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn/package-lock.yml
01:49:31  Installing dbt-labs/dbt_utils
01:49:39  Installed from version 1.3.0
01:49:39  Up to date!
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$
```

Now, we'll generate surrogate key function –




So, the above *fct reviews was the fact dimension that we already created earlier*. Fact dimension which is sourced from src_reviews where we edited this part as shown in below to generate a surrogate key -

```
SELECT
  {{ dbt_utils.generate_surrogate_key(['listing_id', 'review_date', 'reviewer_name', 'review_text']) }}
    AS review_id,
  *
  FROM src_reviews
WHERE review_text is not null
```

So, we used dbt run –select fct_reviews to run this particular model as needed and encountered a error –

```
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$ dbt run --select fct_reviews
01:58:27  Running with dbt=1.9.3
01:58:33  Registered adapter: snowflake=1.9.0
01:59:01  Unable to do partial parsing because a project dependency has been added
01:59:16  Found 8 models, 1 snapshot, 9 data tests, 1 seed, 3 sources, 590 macros
01:59:16
01:59:16  Concurrency: 1 threads (target='dev')
01:59:16
01:59:28  1 of 1 START sql incremental model DEV.fct_reviews ............................. [RUN]
01:59:32  1 of 1 ERROR creating sql incremental model DEV.fct_reviews .................... [ERROR in 4.13s]
01:59:32
01:59:32  Finished running 1 incremental model in 0 hours 0 minutes and 16.25 seconds (16.25s).
01:59:32
01:59:32  Completed with 1 error, 0 partial successes, and 0 warnings:
01:59:32
01:59:32    Compilation Error in model fct_reviews (models/fct/fct_reviews.sql)

              The source and target schemas on this incremental model are out of sync!
              They can be reconciled in several ways:
                - set the `on_schema_change` config to either append_new_columns or sync_all_columns, depending on your situation.
                - Re-run the incremental model with `full_refresh: True` to update the target schema.
                - update the schema manually and re-run the process.

              Additional troubleshooting context:
                 Source columns not in target: [SnowflakeColumn(column='REVIEW_ID', dtype='VARCHAR', char_size=32, numeric_precision=None, numeric_scale=N
one)]
                 Target columns not in source: []
                 New column types: []


  > in macro process_schema_changes (macros/materializations/models/incremental/on_schema_change.sql)
  > called by macro materialization_incremental_snowflake (macros/materializations/incremental.sql)
  > called by model fct_reviews (models/fct/fct_reviews.sql)
01:59:32
01:59:32  Done. PASS=0 WARN=0 ERROR=1 SKIP=0 TOTAL=1
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$
```

This is because we earlier set the materialization settings that it should fail on a schema change, since adding a new column in the model means schema change there for it failed –

models > fct > fct_reviews.sql

Add documentation or tests | Datapilot Notebooks

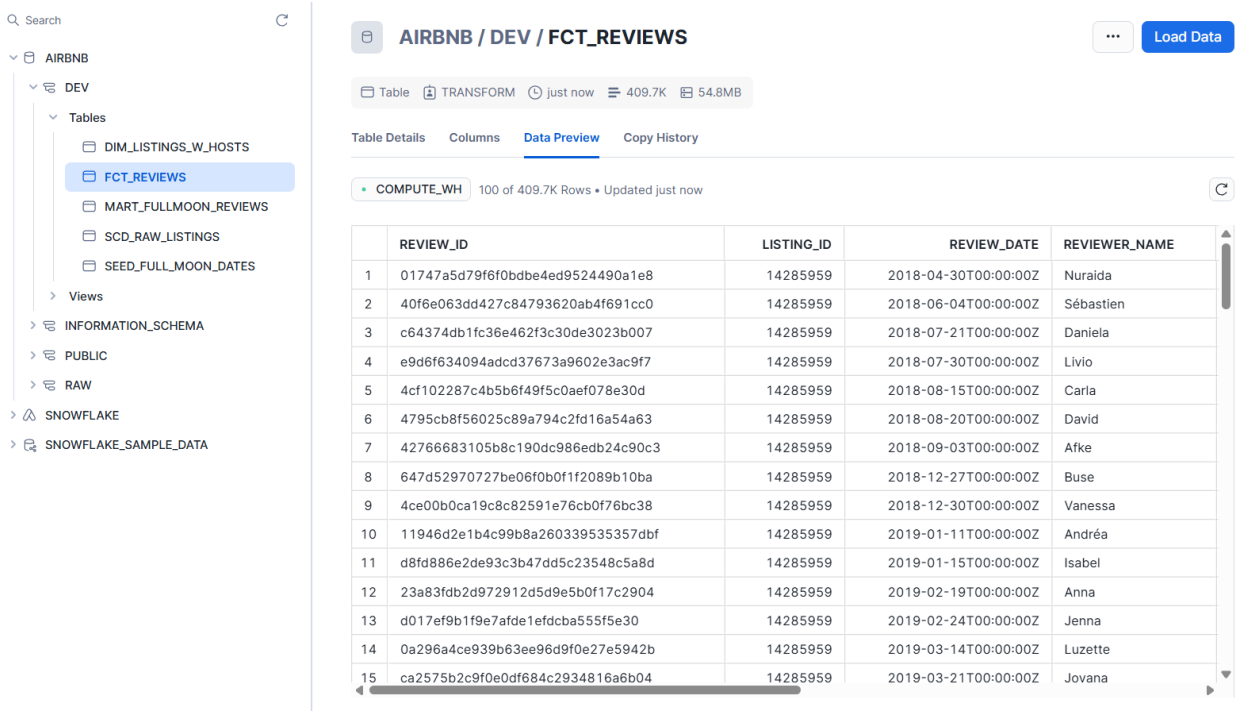
```
1   {{
2     config(
3       materialized = 'incremental',
4       on_schema_change='fail'
5       )
6   }}
7   -- this is model level configuration
8   -- 'this is added so that if theres any changes in
9   WITH src_reviews AS (
10    SELECT * FROM {{ ref('src_reviews') }}
11  )
12  --SELECT * FROM src_reviews
13  --WHERE review_text is not null
```

Having this set to ***'fail'*** is a good thing as we wont need a ***unsupervised change*** in our model/schema in production unit or dev unit.

To tackle this what we can do based on our situation is to recreate the entire fct_reviews model again.
use – dbt run –select fct_reviews –full-refresh

```
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$ dbt run --select fct_reviews --full-refresh
02:04:26  Running with dbt=1.9.3
02:04:34  Registered adapter: snowflake=1.9.0
02:05:13  Found 8 models, 1 snapshot, 9 data tests, 1 seed, 3 sources, 590 macros
02:05:13
02:05:13  Concurrency: 1 threads (target='dev')
02:05:13
02:05:24  1 of 1 START sql incremental model DEV.fct_reviews ............................. [RUN]
02:05:32  1 of 1 OK created sql incremental model DEV.fct_reviews ........................ [SUCCESS 1 in 8.06s]
02:05:32
02:05:32  Finished running 1 incremental model in 0 hours 0 minutes and 19.13 seconds (19.13s).
02:05:32
02:05:32  Completed successfully
02:05:32
02:05:32  Done. PASS=1 WARN=0 ERROR=0 SKIP=0 TOTAL=1
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$
```

This can be verified with snowflake since we used dbt run therefore it has updated the model –




| | REVIEW_ID | LISTING_ID | REVIEW_DATE | REVIEWER_NAME |
|---|---|---|---|---|
| 1 | 01747a5d79f6f0bdbe4ed9524490a1e8 | 14285959 | 2018-04-30T00:00:00Z | Nuraida |
| 2 | 40f6e063dd427c84793620ab4f691cc0 | 14285959 | 2018-06-04T00:00:00Z | Sébastien |
| 3 | c64374db1fc36e462f3c30de3023b007 | 14285959 | 2018-07-21T00:00:00Z | Daniela |
| 4 | e9d6f634094adcd37673a9602e3ac9f7 | 14285959 | 2018-07-30T00:00:00Z | Livio |
| 5 | 4cf102287c4b5b6f49f5c0aef078e30d | 14285959 | 2018-08-15T00:00:00Z | Carla |
| 6 | 4795cb8f56025c89a794c2fd16a54a63 | 14285959 | 2018-08-20T00:00:00Z | David |
| 7 | 42766683105b8c190dc986edb24c90c3 | 14285959 | 2018-09-03T00:00:00Z | Afke |
| 8 | 647d52970727be06f0b0f1f2089b10ba | 14285959 | 2018-12-27T00:00:00Z | Buse |
| 9 | 4ce00b0ca19c8c82591e76cb0f76bc38 | 14285959 | 2018-12-30T00:00:00Z | Vanessa |
| 10 | 11946d2e1b4c99b8a2603395353 57dbf | 14285959 | 2019-01-11T00:00:00Z | Andréa |
| 11 | d8fd886e2de93c3b47dd5c23548c5a8d | 14285959 | 2019-01-15T00:00:00Z | Isabel |
| 12 | 23a83fdb2d972912d5d9e5b0f17c2904 | 14285959 | 2019-02-19T00:00:00Z | Anna |
| 13 | d017ef9b1f9e7afde1efdcba555f5e30 | 14285959 | 2019-02-24T00:00:00Z | Jenna |
| 14 | 0a296a4ce939b63ee96d9f0e27e5942b | 14285959 | 2019-03-14T00:00:00Z | Luzette |
| 15 | ca2575b2c9f0e0df684c2934816a6b04 | 14285959 | 2019-03-21T00:00:00Z | Jovana |

We can view the review_id is created with hash value based on our surrogate key input.

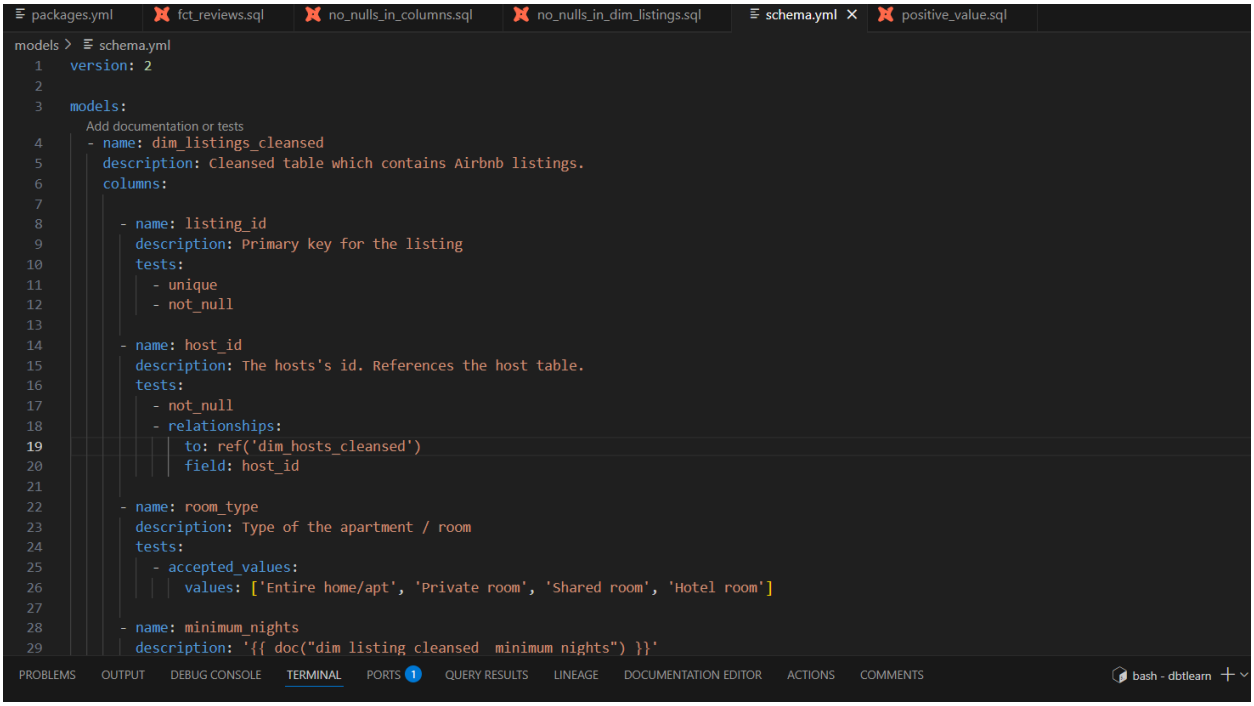
DOCUMENTATION

- Understand how to document models
- Use the documentation generator and server
- Add assets and markdown to the documentation
- Discuss dve vs production documentation serving

Documentation overview

- Documentations can be degined two ways
  - In yaml files (like schema.yml)
  - In standalone markdown files
- Dbt ships with a lightweight documentation web server
- For customizing the landing page, a special file, overview.md is used
- You can add your own assets (like images) to a special project folder.

Adding schema level documentation -




```yaml
version: 2

models:
  - name: dim_listings_cleansed
    description: Cleansed table which contains Airbnb listings.
    columns:

      - name: listing_id
        description: Primary key for the listing
        tests:
          - unique
          - not_null

      - name: host_id
        description: The hosts's id. References the host table.
        tests:
          - not_null
          - relationships:
              to: ref('dim_hosts_cleansed')
              field: host_id

      - name: room_type
        description: Type of the apartment / room
        tests:
          - accepted_values:
              values: ['Entire home/apt', 'Private room', 'Shared room', 'Hotel room']

      - name: minimum_nights
        description: '{{ doc("dim_listing_cleansed__minimum_nights") }}'
```

***Dbt doc generate*** is the command to generate docs.

```
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$ dbt docs generate
05:29:56  Running with dbt=1.9.3
05:30:00  Registered adapter: snowflake=1.9.0
05:30:21  Found 8 models, 1 snapshot, 16 data tests, 1 seed, 3 sources, 590 macros
05:30:21
05:30:21  Concurrency: 1 threads (target='dev')
05:30:21
05:30:27  Building catalog
05:30:32  Catalog written to /mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn/target/catalog.json
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$
```

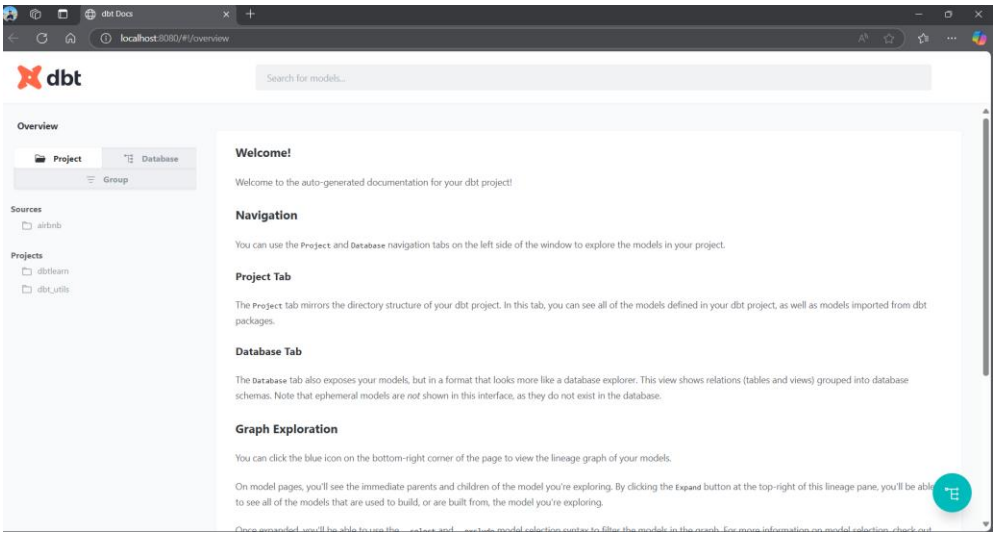
Shows the taget folder to which generated html file with json file is present as a document.

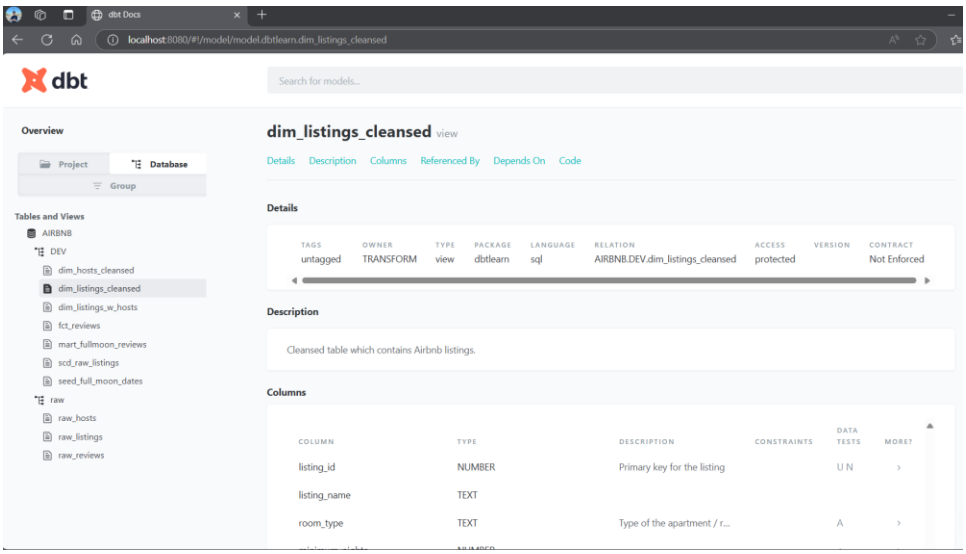
```
05:30:32  Catalog written to /mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn/target/catalog.json
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$ cd target
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn/target$ ls
catalog.json  graph.gpickle       index.html     partial_parse.msgpack  run_results.json       sources.json
compiled      graph_summary.json  manifest.json  run                    semantic_manifest.json
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn/target$
```

these are the files it generated as documentation.

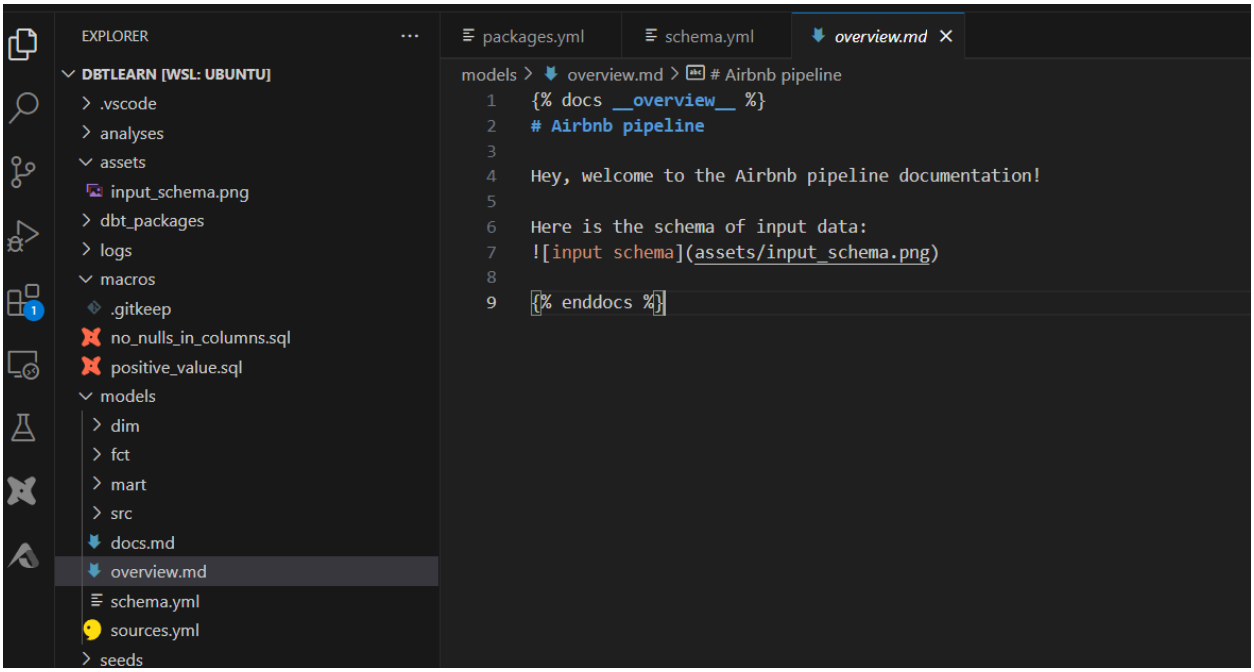
Use dbt docs serve to launch the documentation in html

```
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn/target$ cd ..
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$ dbt docs serve
05:33:04  Running with dbt=1.9.3
Serving docs at 8080
To access from your browser, navigate to: http://localhost:8080



Press Ctrl+C to exit.
gio: http://localhost:8080: Operation not supported
127.0.0.1 - - [15/Mar/2025 00:33:46] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [15/Mar/2025 00:33:47] "GET /manifest.json?cb=1742016827168 HTTP/1.1" 200 -
127.0.0.1 - - [15/Mar/2025 00:33:47] "GET /catalog.json?cb=1742016827168 HTTP/1.1" 200 -
127.0.0.1 - - [15/Mar/2025 00:33:47] code 404, message File not found
127.0.0.1 - - [15/Mar/2025 00:33:47] "GET /$%7Brequire('./assets/favicons/favicon.ico')%7D HTTP/1.1" 404 -
```



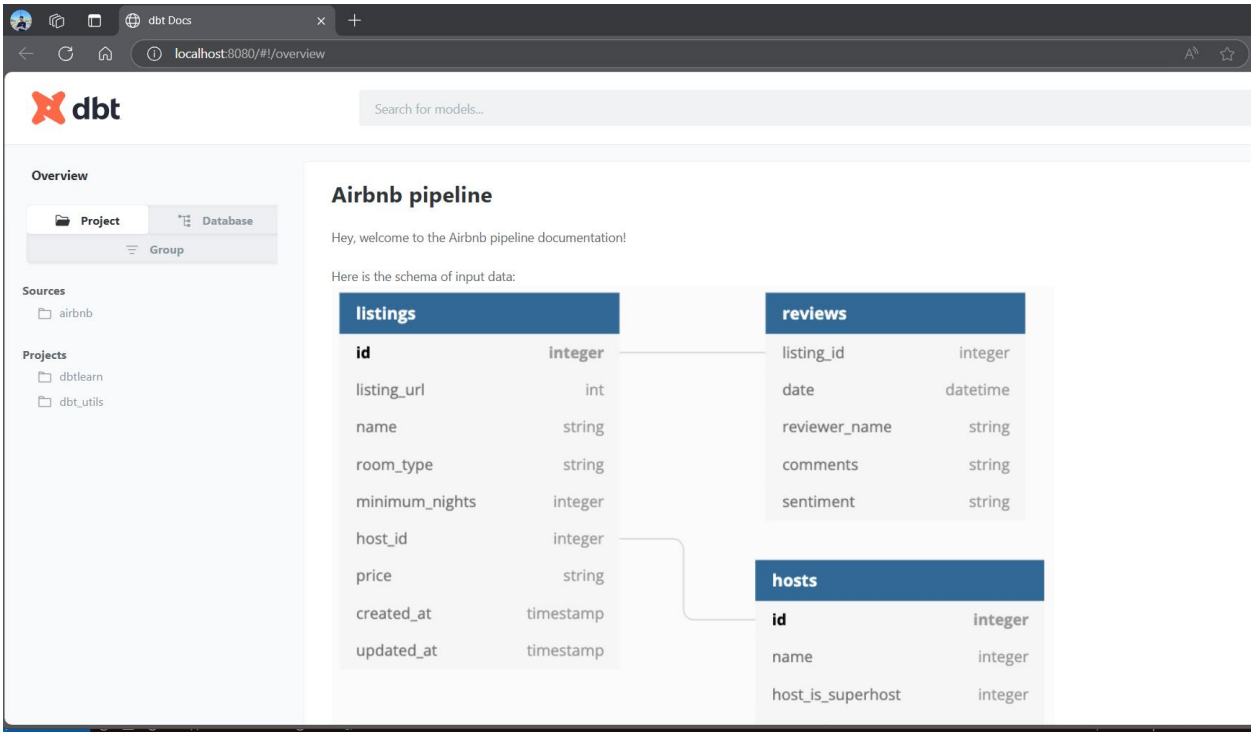



So, the landing page can be changed as well. (overview)

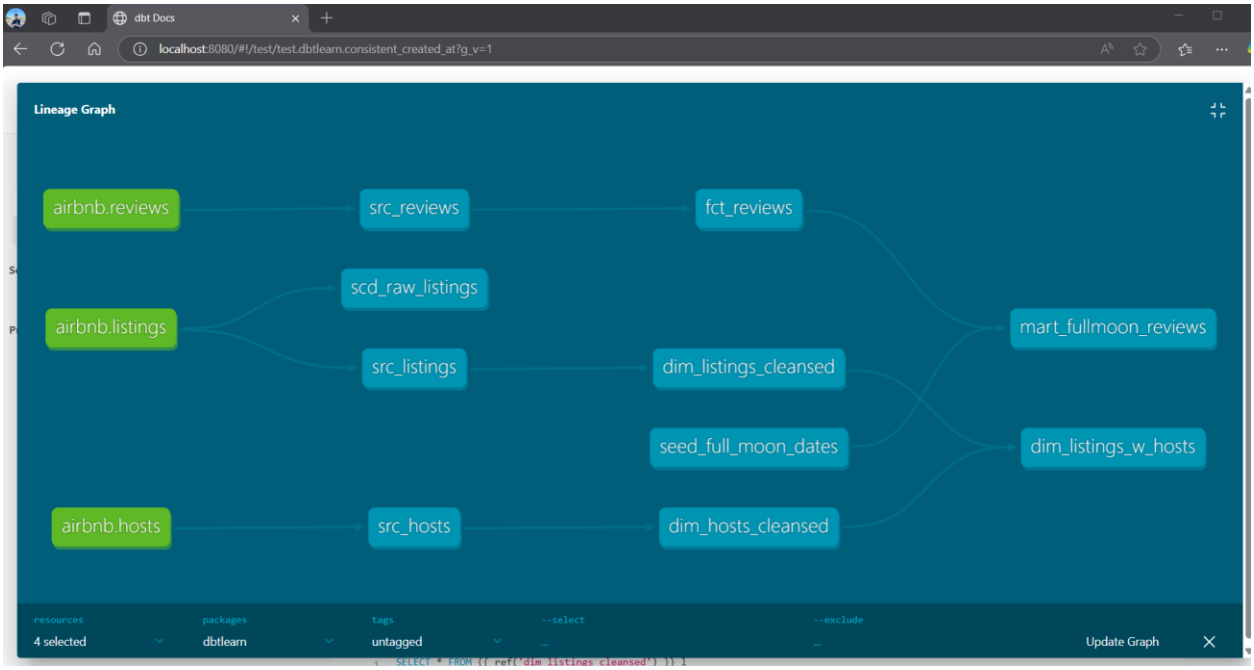



here, I have mentioned the landing page image which is a database schema representation.

This is how it looks -



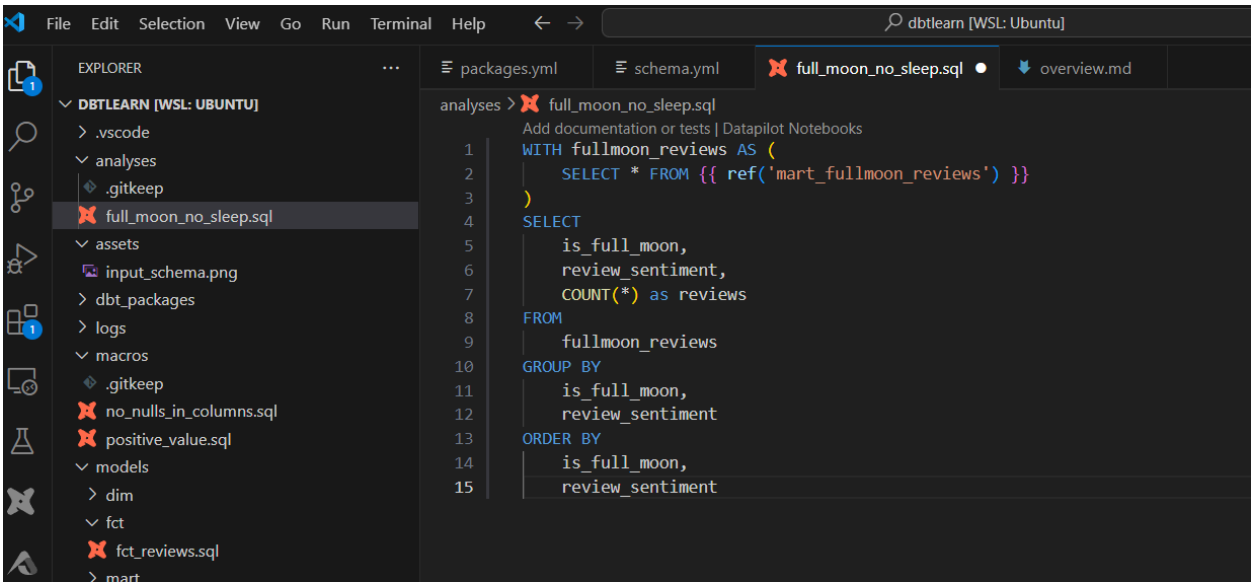
This is the data Lineage graph (DAG) which shows the data flow of the model provided through documentation -




---

**ANALYSIS**

We'll be performing analytical queries here, which can be stored, referenced and reused in dbt.

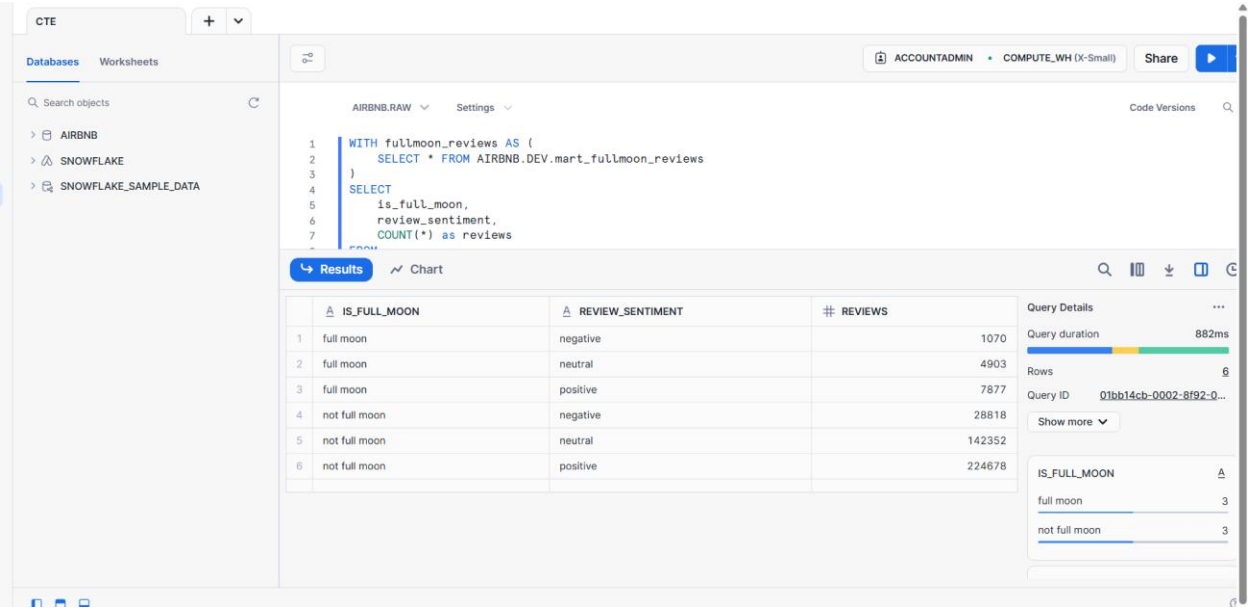



Created full_moon_no_sleep.sql in analysis folder and added this query which does -
This query pulls all rows from mart_fullmoon_reviews, then groups them by whether it was a full moon (is_full_moon) and the review's sentiment. It counts how many reviews fall into each group, and then orders the results by is_full_moon and sentiment. Essentially, it's giving you the total number of reviews for each full-moon vs. not-full-moon and sentiment combination.

So this is how it is referencing and building the query -

```
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$ dbt compile
02:46:46  Running with dbt=1.9.3
02:46:54  Registered adapter: snowflake=1.9.0
02:47:28  Found 8 models, 1 snapshot, 16 data tests, 1 seed, 1 analysis, 3 sources, 590 macros
02:47:28
02:47:28  Concurrency: 1 threads (target='dev')
02:47:28
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$ less target/compiled/dbtlearn/analyse
s/full_moon_no_sleep.sql
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$ cat target/compiled/dbtlearn/analyses
/full_moon_no_sleep.sql
WITH fullmoon_reviews AS (
    SELECT * FROM AIRBNB.DEV.mart_fullmoon_reviews
)
SELECT
    is_full_moon,
    review_sentiment,
    COUNT(*) as reviews
FROM
    fullmoon_reviews
GROUP BY
    is_full_moon,
    review_sentiment
ORDER BY
    is_full_moon,
    review_sentiment(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$
```

Use this query on snowflake to look for the results –




---

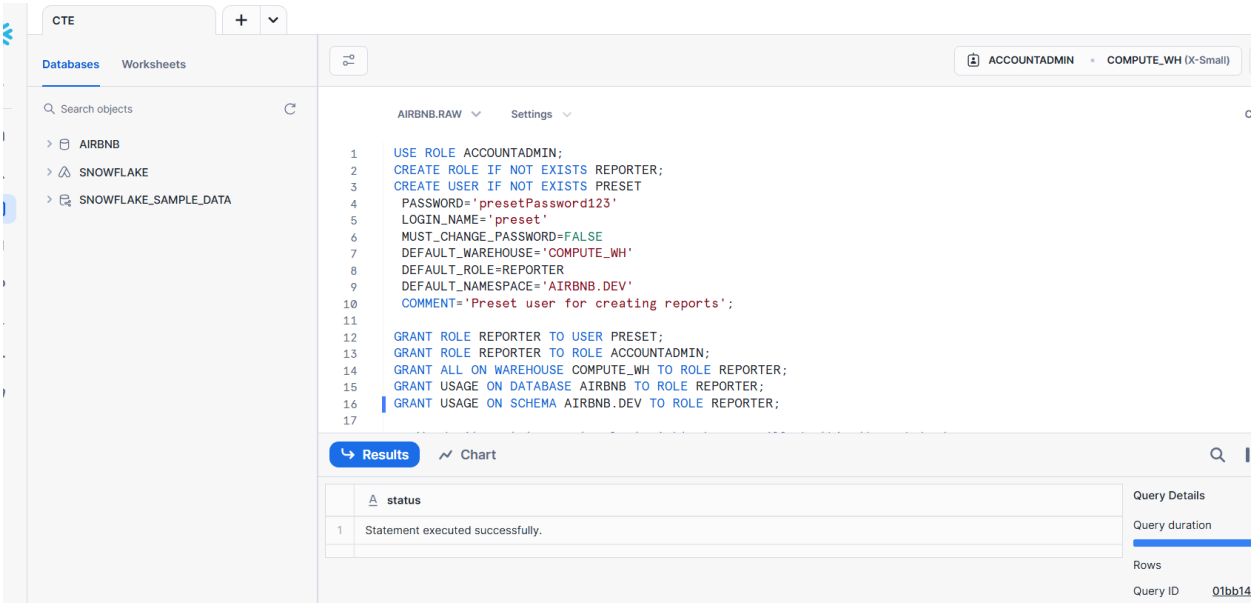
## HOOKS

In DBT, hooks are like custom scripts that run automatically before or after certain tasks (like running models or tests). They let you add extra steps—such as logging information, cleaning up data, or notifying someone—without having to change your main code. Essentially, hooks let you automate and customize actions around your DBT projects.

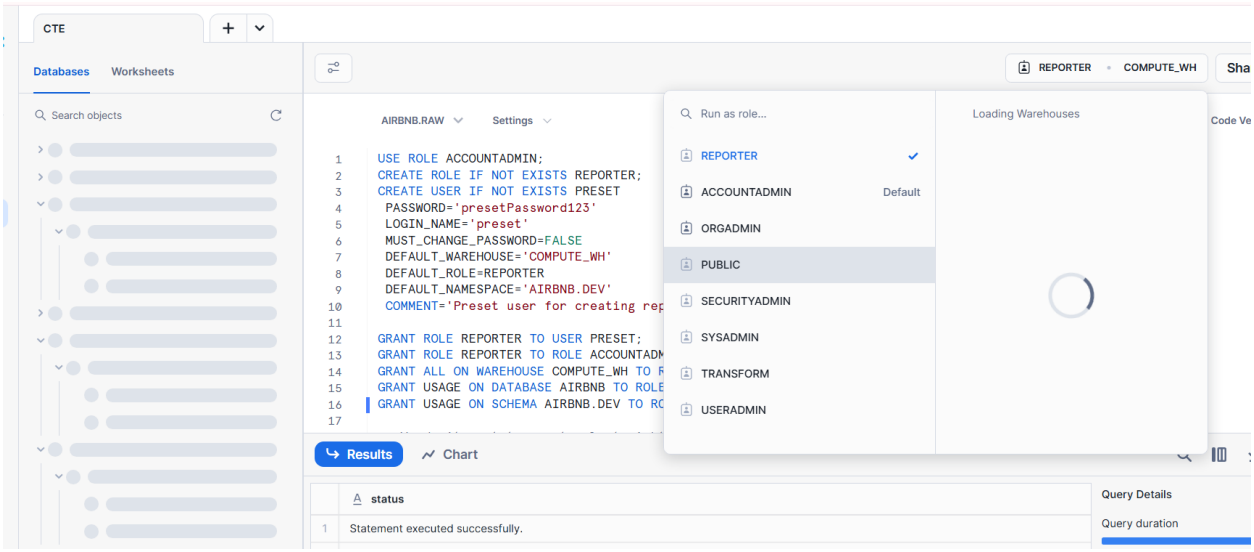
- Hooks are SQLs that are executed at predefined times.
- Hooks can be configured on the project, subfolder or model level.
- Hooks types:
    - On_run_start: executed at the start of dbt {run,seed,snapshot}

- on_run_end: executed at the end of dbt {run, seed, snapshot}
- pre-hook: executed before a model/seed/snapshot is built.
- post-hook: executed after a model/seed/snapshot is built.

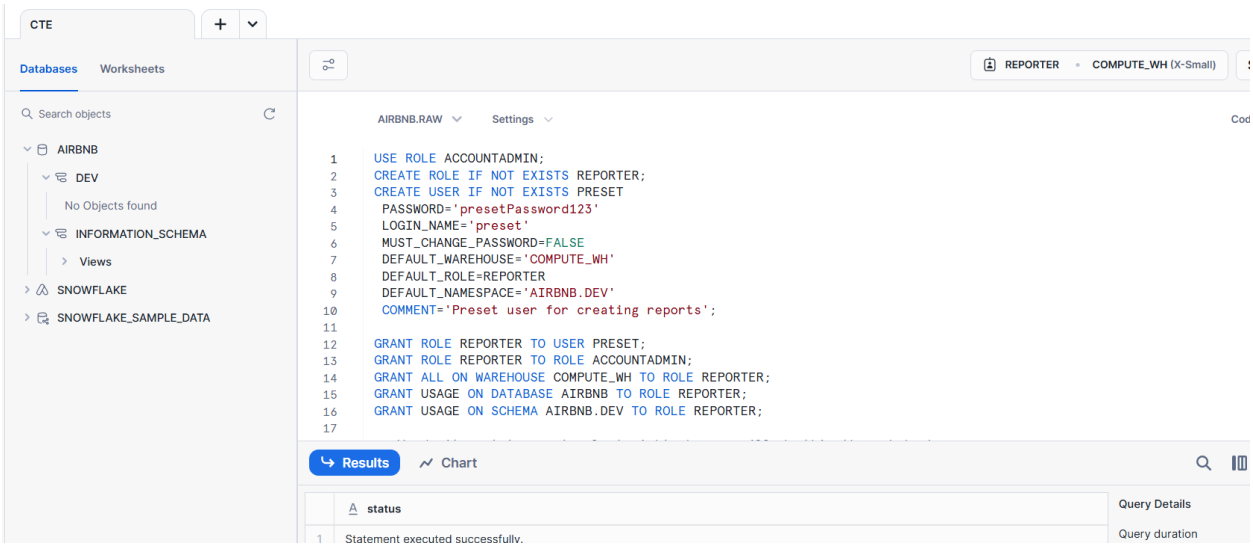
We'll no implement a post hook. Create a reporter role ->



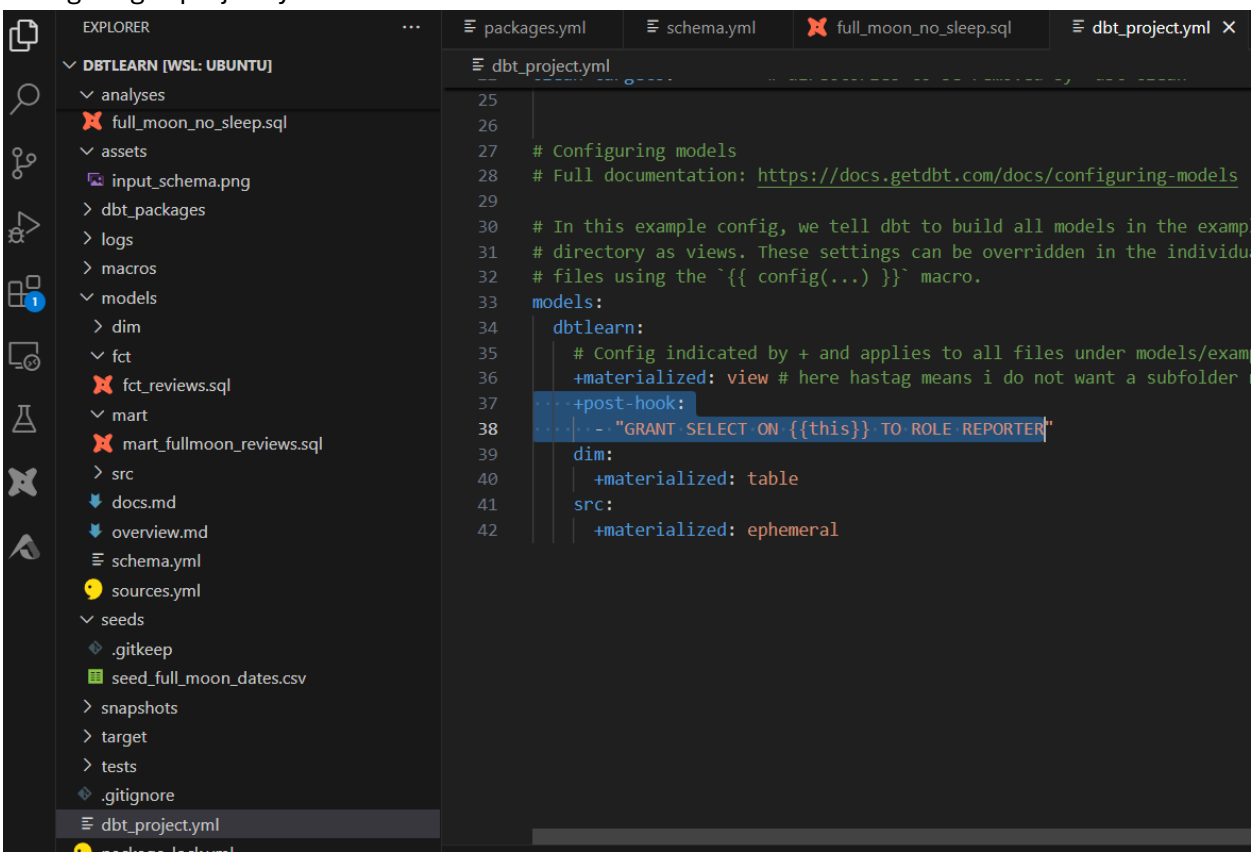

select reporter role -




You can see that reporter does not have access to Airbnb – tables and views -

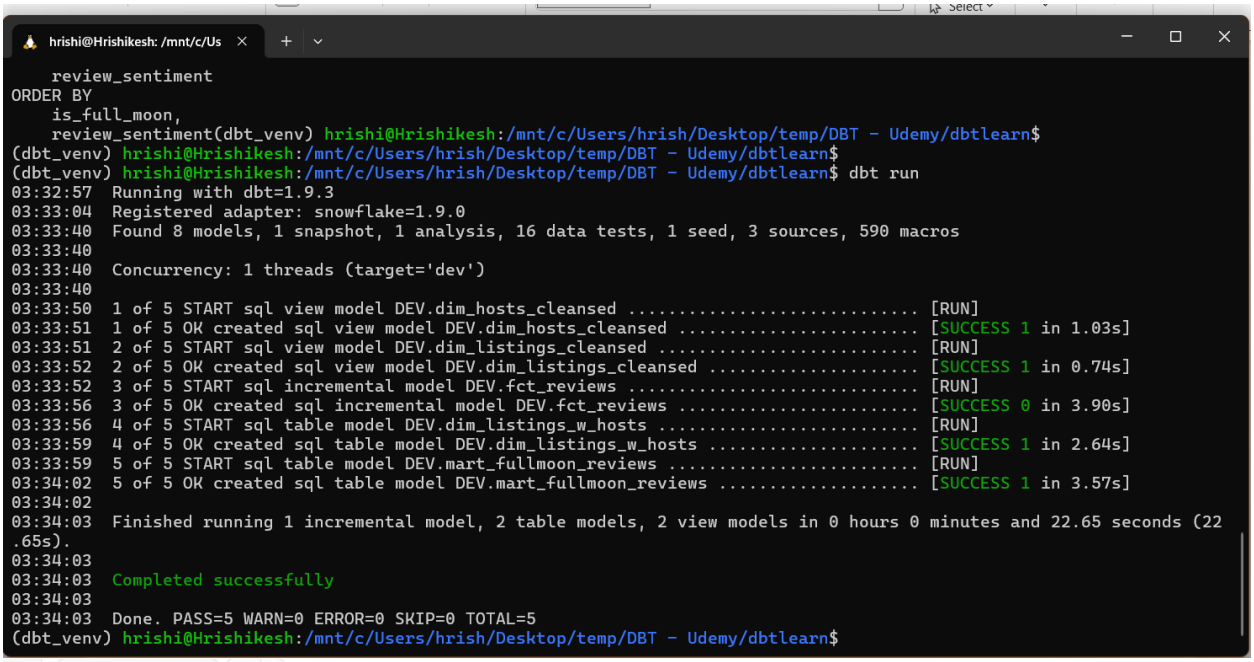


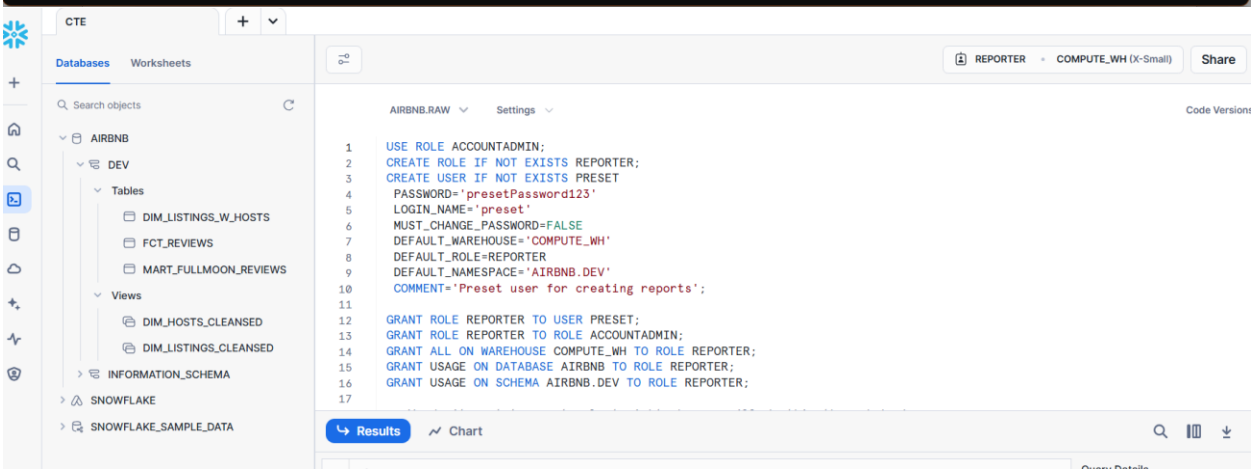

since, explicitly the grants are not stated to reporter we can still give the access to the schema by configuring in project.yml-




Now, post hook does is - After dbt finishes creating the table or view for the model, it automatically runs any **post-hooks**. In that moment, dbt takes the code written (for example, GRANT SELECT ON {{ this }} TO ROLE REPORTER), **replaces {{ this }} with the actual table name**, and then **executes** the resulting SQL statement on the data warehouse. That's how the GRANT statement gets run.

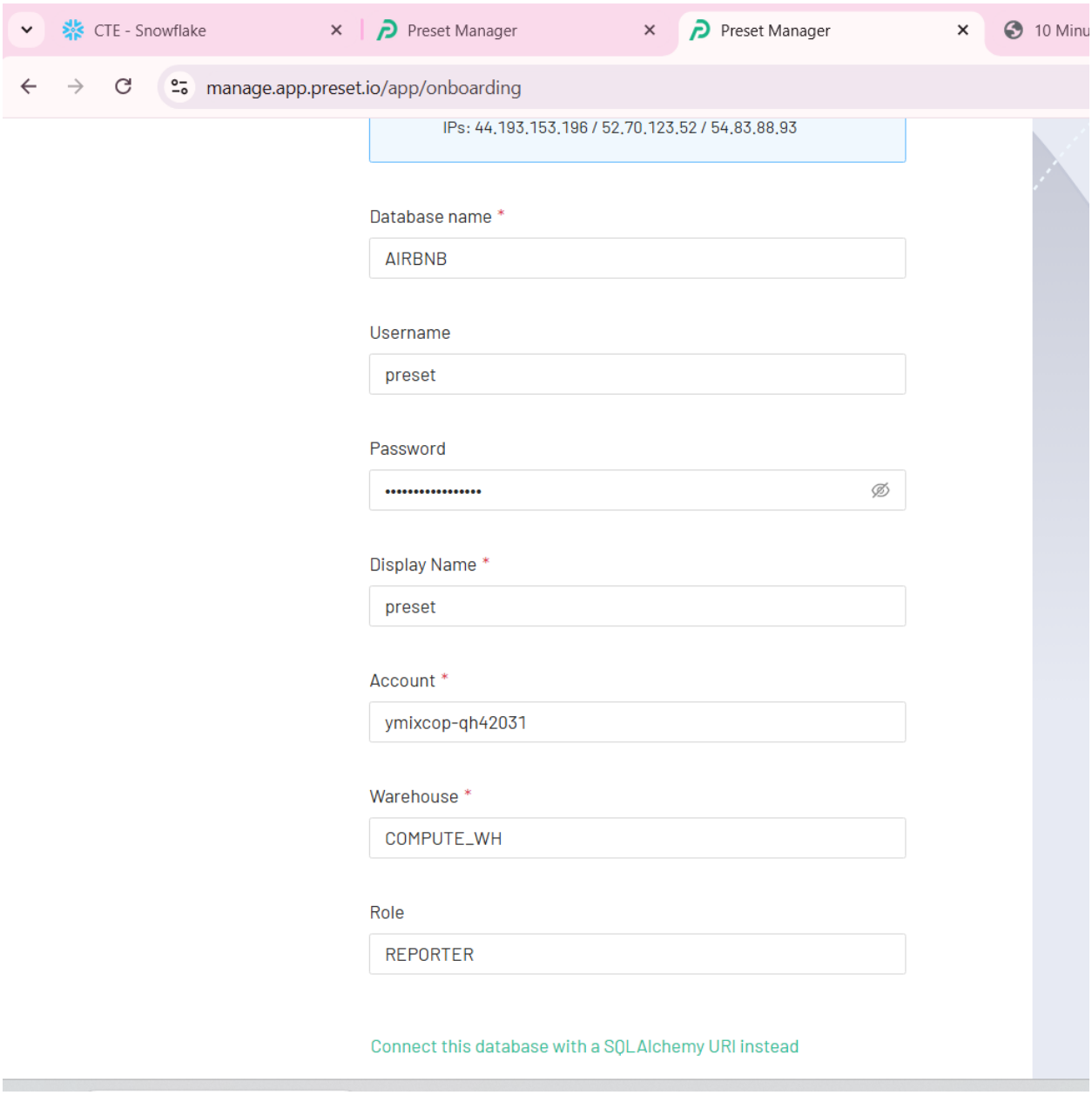
Remember the scope of this post hook is within **DBTLEARN** so any models under this will be affected.




so now, due to post hook grant statement we can see that reporter role has access to the schemas now.

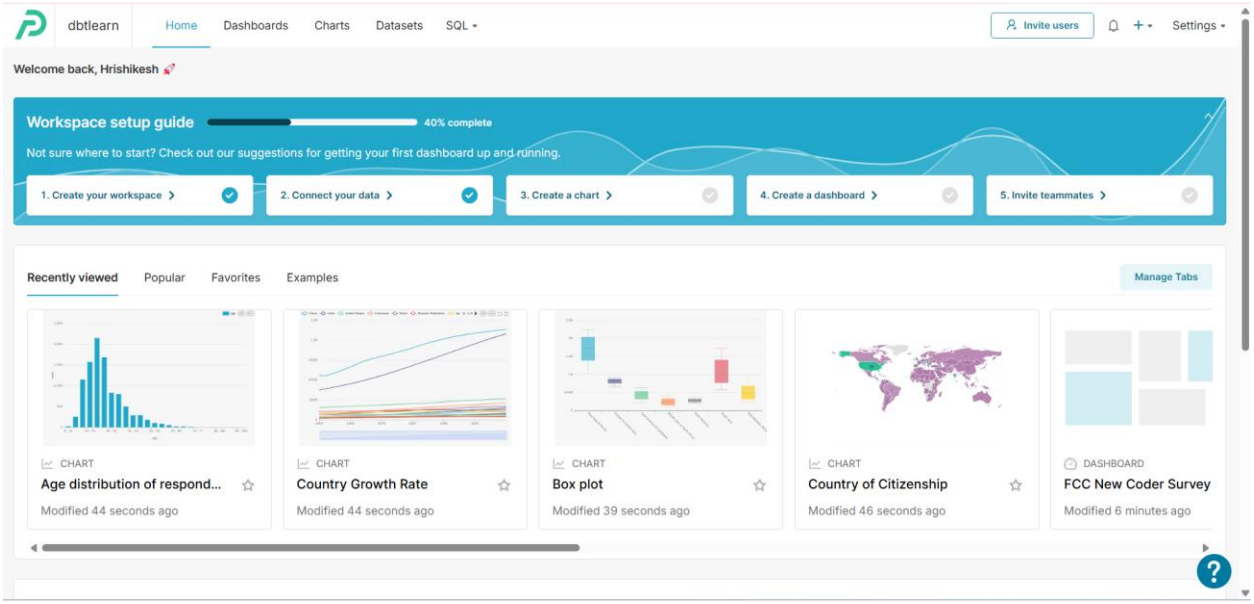
## SETTING UP PRESET



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS 1  QUERY RESULTS  LINEAGE  DOCUMENTATION EDITOR  ACTIONS  COMMENTS

(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$ cat ~/.dbt/profiles.yml
dbtlearn:
  outputs:
    dev:
      account: ymixcop-qh42031
      database: AIRBNB
      password: Hrishikesh$261999
      role: transform
      schema: DEV
      threads: 1
      type: snowflake
      user: dbtlearner
      warehouse: COMPUTE_WH
  target: dev
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$

Defer                                                    Ln 42, Col 31
```

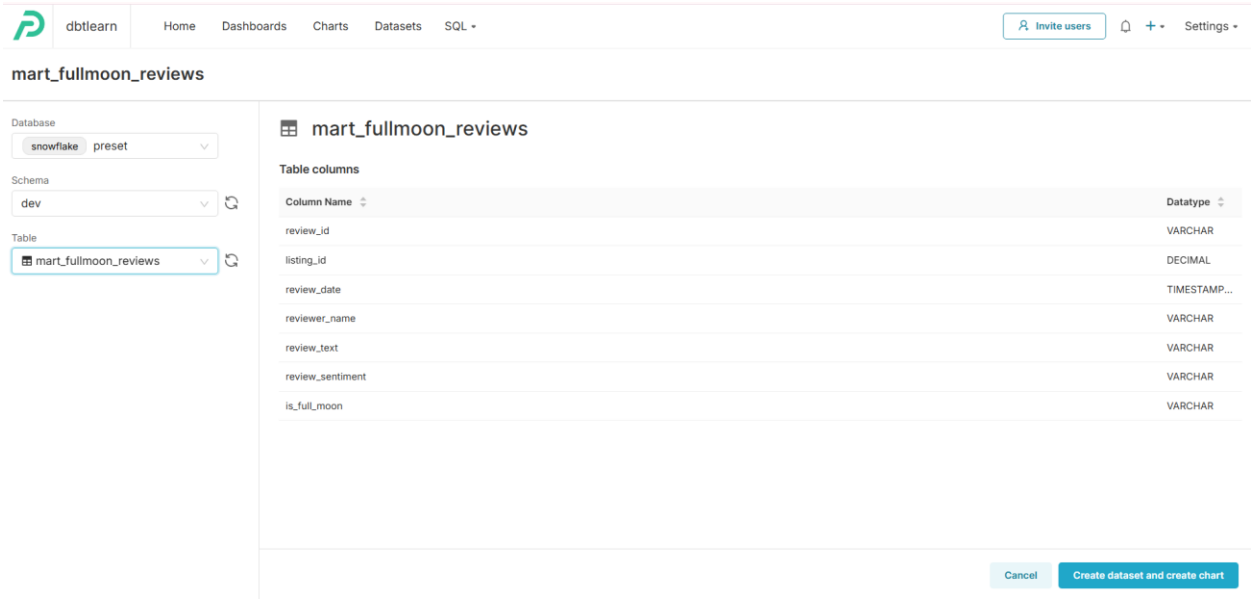
Can use some of the info from here to set up preset account.

And this will be successful home screen display –

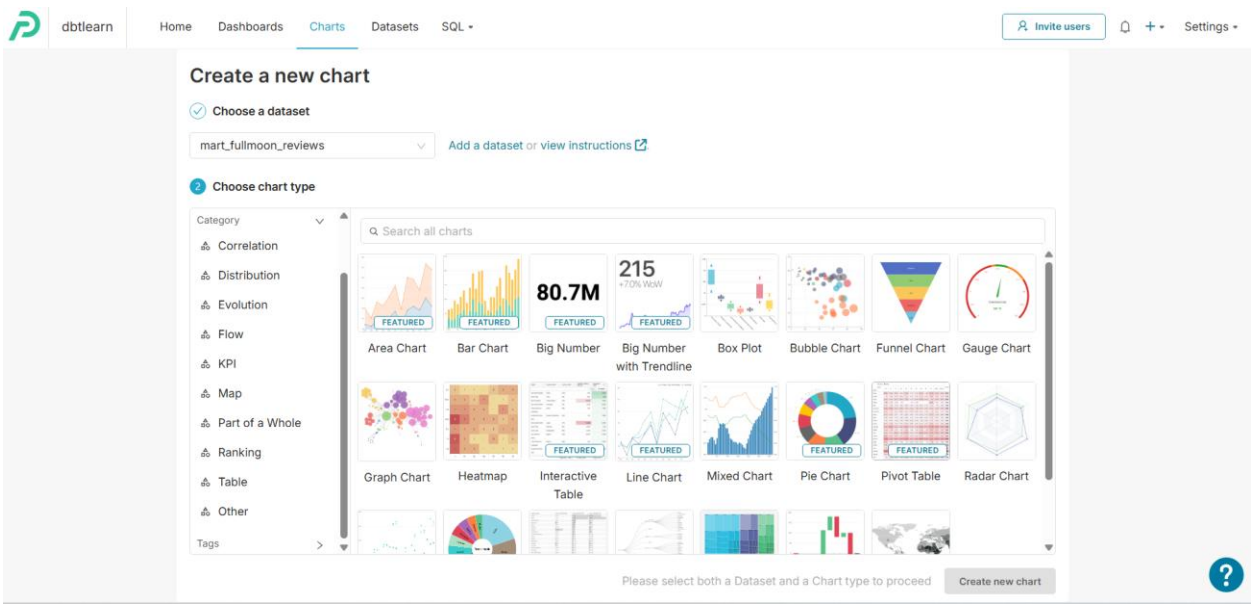


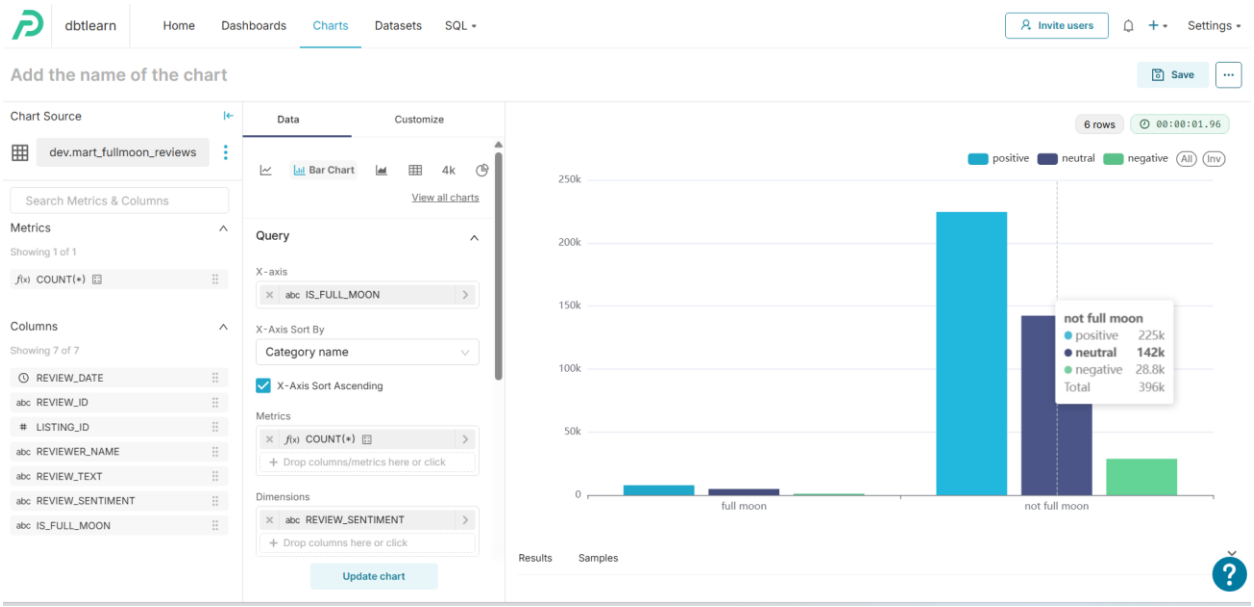

Now, connect the dataset of Airbnb which is connected through snowflake as we performed data ingestion and make a visual representation for bookings with full moon and no full moon.

Connected to the database – Airbnb



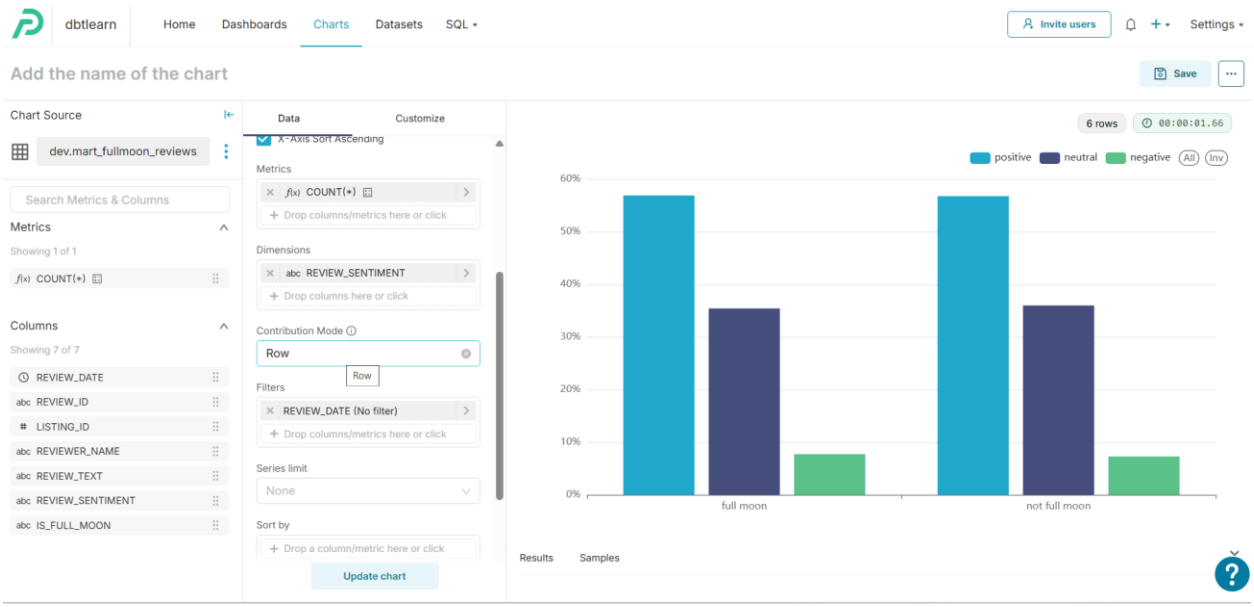

Create new chart –



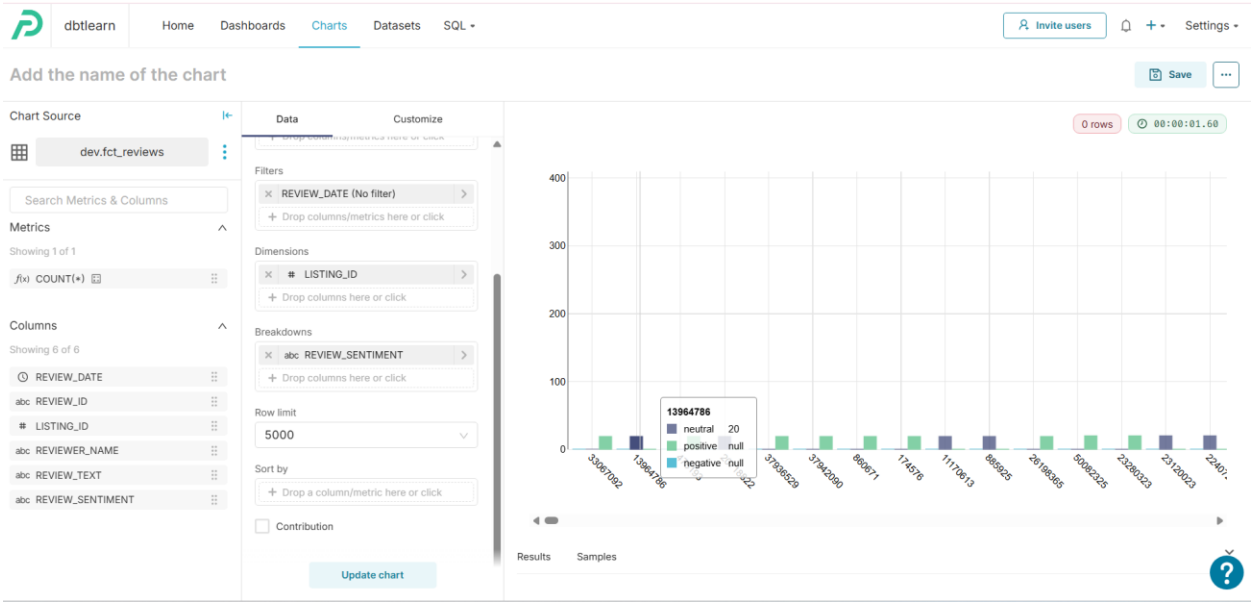



But as we can see there's more data with not full moon and it seems the dataset is quite skewed.

To tackle this we have to just do one adjustment and i.e. we need to change the contribution mode from 'none' to 'row' so that it will show the contribution of that particular category in its own data.




The negative review In full moon is just 8% which is just 1% more than not full moon negative reviews and neutral reviews are 36% in not full moon data which is 1% more than full moon data. This is not very significant insight which we thought initially would be a considering factors in listings.




This chart shows the total number of reviews per listings and the breakdown for the count for type of sentiment received.

Dashboard url - https://469e14e3.us2a.app.preset.io/superset/dashboard/8/?native_filters_key=qH0HzKQNifAN5gQ8kBPsFsoDMajK_SSPyXAdoU3D1WXodt109WTiWsvXBixyXC5I

---

**EXPOSURES**

Exposures in dbt are a way to formally document and manage the external "consumers" of your dbt models—such as dashboards, reports, applications, or any downstream data products. Here's a breakdown of what exposures are and why they're useful:

**What Are Exposures?**

- **Documentation of Downstream Use:** Exposures allow you to declare how your dbt models are being used outside of dbt. For example, if a particular dashboard relies on certain transformed models, you can record that relationship.
- **Configuration File:** They're typically defined in a YAML file (often named exposures.yml) within your dbt project. This file specifies details like the name of the exposure, its type (e.g., dashboard, report, ML model), owner, URL, and a description of its purpose.

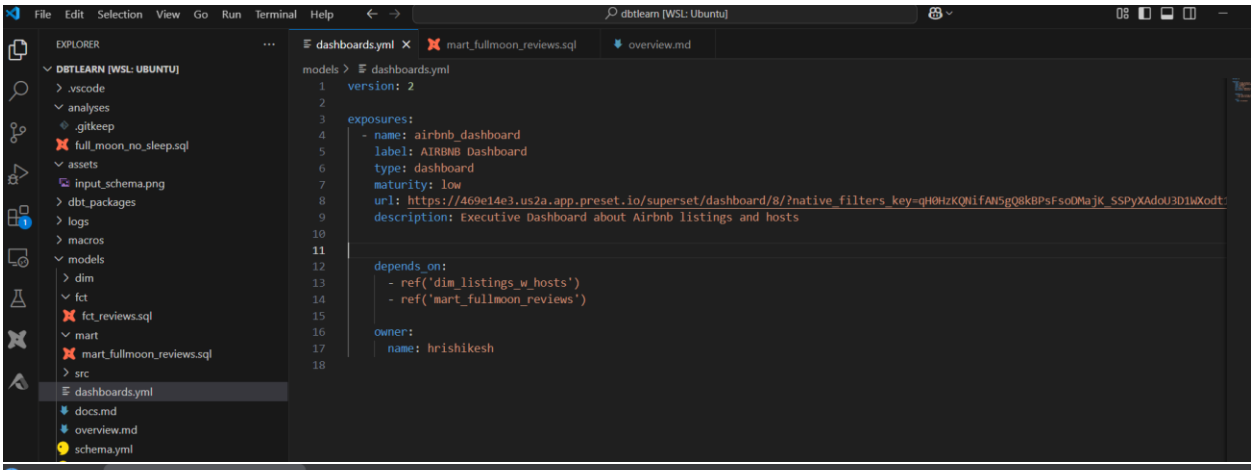
**Why Use Exposures?**

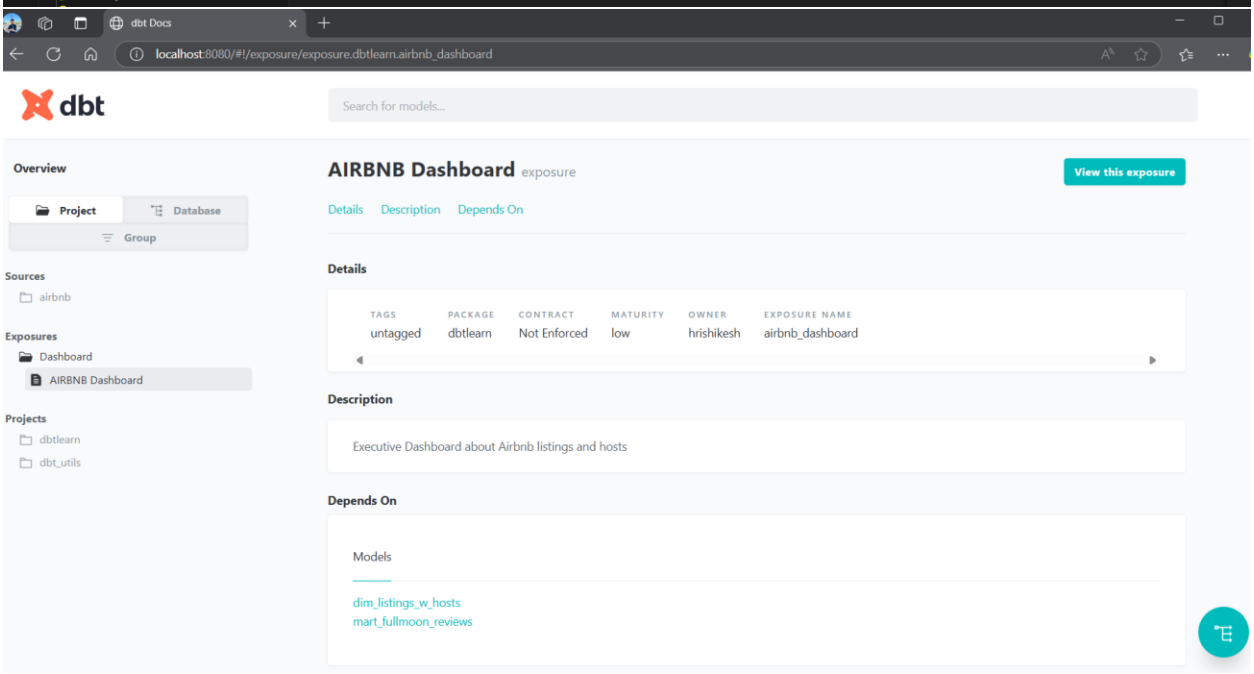
- **Improved Visibility:** By mapping out which dashboards or applications depend on which models, you gain better insight into the impact of changes. This is especially useful during refactoring or when troubleshooting data issues.
- **Enhanced Collaboration:** Documenting exposures creates a shared understanding among teams (data engineers, analysts, BI developers, etc.) about how data flows from your models to business insights.
- **Impact Analysis:** When a change is made to a model, the exposure definitions help identify which downstream assets might be affected, allowing for more careful change management.
- **Audit and Compliance:** For organizations that need to track data lineage and usage for compliance purposes, exposures serve as a formal record of how data is consumed.

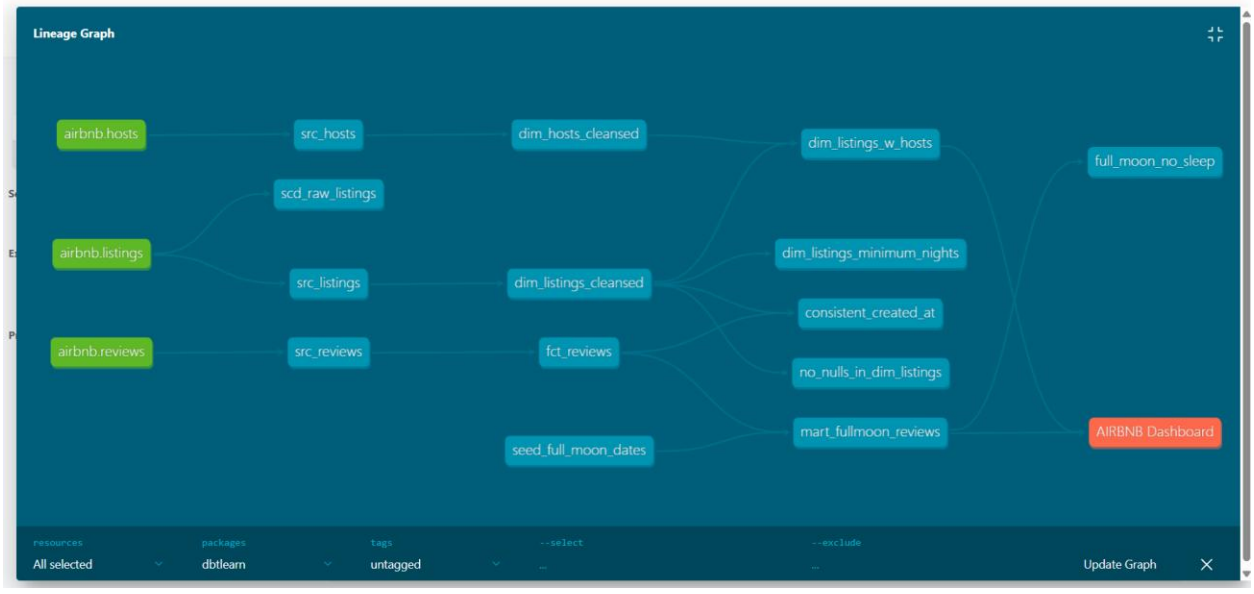
**How Do Exposures Work in a dbt Project?**

1. **Define the Exposure:** You create a YAML file (e.g., exposures.yml) where you define each exposure with attributes like:
    - **name:** A unique identifier for the exposure.
    - **type:** The kind of downstream asset (e.g., dashboard, report).
    - **owner:** Who is responsible for this exposure.
    - **url:** A link to the external asset (if applicable).
    - **depends_on:** A list of dbt models or sources that feed into the exposure.
    - **description:** Additional context about the exposure.
2. **Integrate into dbt's Graph:** Once defined, dbt integrates exposures into its DAG (Directed Acyclic Graph). This means when you run dbt commands (like dbt docs generate), the exposures show up in the lineage, illustrating how data flows from models to external tools.
3. **Use for Impact Analysis:** When updating a model, you can quickly check which exposures (and thus which dashboards or reports) rely on that model. This facilitates proactive communication and planning to mitigate any potential disruptions.

Exposures in dbt are a powerful feature for managing data dependencies beyond your immediate transformations. They help ensure that any changes in your dbt models are made with an understanding of their broader business implications. This leads to more robust, reliable, and transparent data workflows.

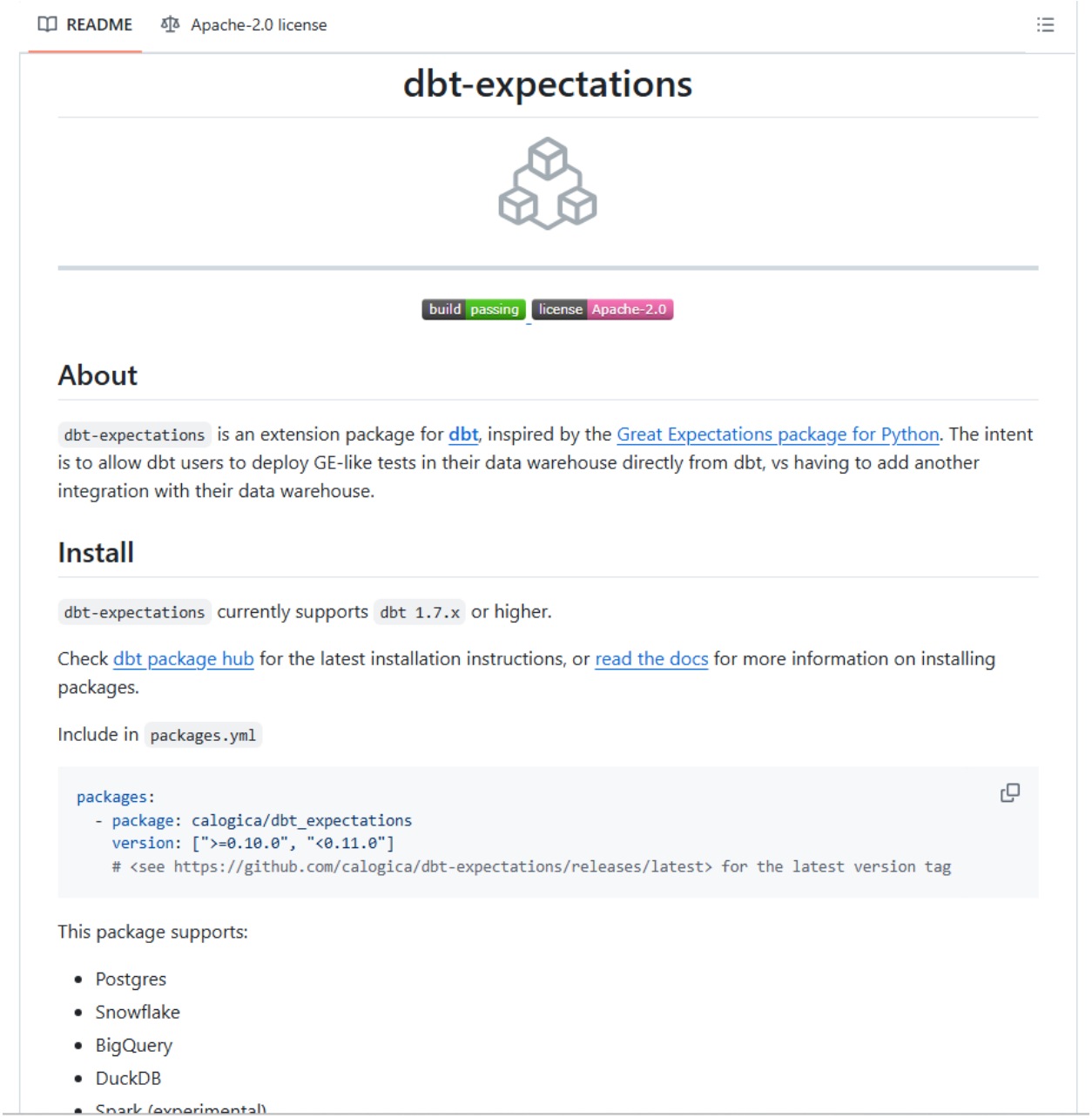

File Edit Selection View Go Run Terminal Help
dbtlearn [WSL: Ubuntu]
EXPLORER
DBTLEARN [WSL: UBUNTU]
.vscode
analyses
.gitkeep
full_moon_no_sleep.sql
assets
input_schema.png
dbt_packages
logs
macros
models
dim
fct
fct_reviews.sql
mart
mart_fullmoon_reviews.sql
src
dashboards.yml
docs.md
overview.md
schema.yml
dashboards.yml
mart_fullmoon_reviews.sql
overview.md
models > dashboards.yml
version: 2
exposures:
- name: airbnb_dashboard
label: AIRBNB Dashboard
type: dashboard
maturity: low
url: https://469e14e3.us2a.app.preset.io/superset/dashboard/8/?native_filters_key=qH0HzKQNifAN5gQ8kBPsFsoDMajK_5SPyXAdoU3D1WXodt
description: Executive Dashboard about Airbnb listings and hosts
depends_on:
- ref('dim_listings_w_hosts')
- ref('mart_fullmoon_reviews')
owner:
name: hrishikesh


dbt Docs
localhost:8080/#!/exposure/exposure.dbtlearn.airbnb_dashboard
dbt
Search for models...
Overview
Project
Database
Group
Sources
airbnb
Exposures
Dashboard
AIRBNB Dashboard
Projects
dbtlearn
dbt_utils
AIRBNB Dashboard exposure
View this exposure
Details
Description
Depends On
Details
TAGS
PACKAGE
CONTRACT
MATURITY
OWNER
EXPOSURE NAME
untagged
dbtlearn
Not Enforced
low
hrishikesh
airbnb_dashboard
Description
Executive Dashboard about Airbnb listings and hosts
Depends On
Models
dim_listings_w_hosts
mart_fullmoon_reviews


Lineage Graph
airbnb.hosts
src_hosts
dim_hosts_cleansed
dim_listings_w_hosts
full_moon_no_sleep
scd_raw_listings
airbnb.listings
src_listings
dim_listings_cleansed
dim_listings_minimum_nights
consistent_created_at
airbnb.reviews
src_reviews
fct_reviews
no_nulls_in_dim_listings
mart_fullmoon_reviews
AIRBNB Dashboard
seed_full_moon_dates
resources
All selected
packages
dbtlearn
tags
untagged
--select
--exclude
Update Graph

**GREAT EXPECTATIONS**

This is a testing open-source package to have a sanity check in your data pipeline.



README Apache-2.0 license

# dbt-expectations

build passing license Apache-2.0

## About

`dbt-expectations` is an extension package for **dbt**, inspired by the Great Expectations package for Python. The intent is to allow dbt users to deploy GE-like tests in their data warehouse directly from dbt, vs having to add another integration with their data warehouse.

## Install

`dbt-expectations` currently supports `dbt 1.7.x` or higher.

Check dbt package hub for the latest installation instructions, or read the docs for more information on installing packages.

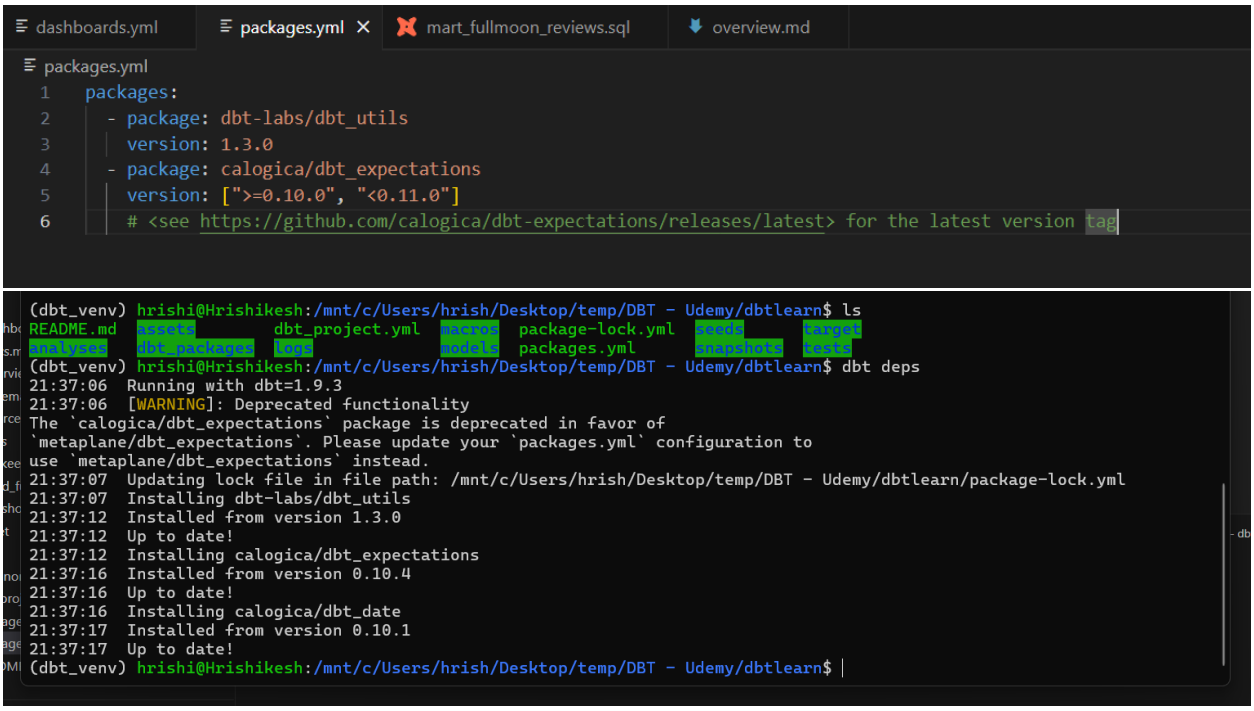
Include in `packages.yml`

```
packages:
  - package: calogica/dbt_expectations
    version: [">=0.10.0", "<0.11.0"]
    # <see https://github.com/calogica/dbt-expectations/releases/latest> for the latest version tag
```

This package supports:

- Postgres
- Snowflake
- BigQuery
- DuckDB
- Spark (experimental)

url -https://github.com/calogica/dbt-expectations
installing this package into our dbt project.




this package contains several tests which can be used as a generic test.

**IMPLEMENTING TESTS**

Implementing one of the tests from **dbt_expectations package –**

### expect_table_row_count_to_equal_other_table

Expect the number of rows in a model match another model.

*Applies to:* Model, Seed, Source

```yaml
models: # or seeds:
  - name: my_model
    tests:
      - dbt_expectations.expect_table_row_count_to_equal_other_table:
          compare_model: ref("other_model")
          group_by: [col1, col2] # (Optional)
          compare_group_by: [col1, col2] # (Optional)
          factor: 1 # (Optional)
          row_condition: "id is not null" # (Optional)
          compare_row_condition: "id is not null" # (Optional)
```

dashboards.yml | packages.yml | schema.yml | sources.yml | mart_fullmoon_r

models > schema.yml

```yaml
models:
  - name: dim_hosts_cleansed
    columns:
      - name: host_id
        tests:
      - name: host_name
        tests:
          - not_null
      - name: is_superhost
        tests:
          - accepted_values:
              values: ['t', 'f']

Add documentation or tests
  - name: dim_listings_w_hosts
    tests:
      - dbt_expectations.expect_table_row_count_to_equal_other_table:
          compare_model: source('airbnb', 'listings')
```

```
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$ dbt test --select dim_listings_w_host
s
00:23:15  Running with dbt=1.9.3
00:23:18  Registered adapter: snowflake=1.9.0
00:23:32  Unable to do partial parsing because a project dependency has been added
00:23:42  Found 8 models, 1 snapshot, 1 analysis, 17 data tests, 1 seed, 3 sources, 1 exposure, 857 macros
00:23:42
00:23:42  Concurrency: 1 threads (target='dev')
00:23:42
00:23:47  1 of 1 START test dbt_expectations_expect_table_row_count_to_equal_other_table_dim_listings_w_hosts_source_air
bnb_listings_  [RUN]
00:23:48  1 of 1 PASS dbt_expectations_expect_table_row_count_to_equal_other_table_dim_listings_w_hosts_source_airbnb_li
stings_  [PASS in 1.01s]
00:23:48
00:23:48  Finished running 1 test in 0 hours 0 minutes and 6.21 seconds (6.21s).
00:23:48
00:23:48  Completed successfully
00:23:48
00:23:48  Done. PASS=1 WARN=0 ERROR=0 SKIP=0 TOTAL=1
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$
```

This output shows successful test that my raw/listings and dim_listings_w_hosts model both have the same number of row count i.e. 17.5k

**Looking for outliers in the data**

this test is done to check for the prices in the dataset as some might be very misleading with the price listings and to check this we have the below test.

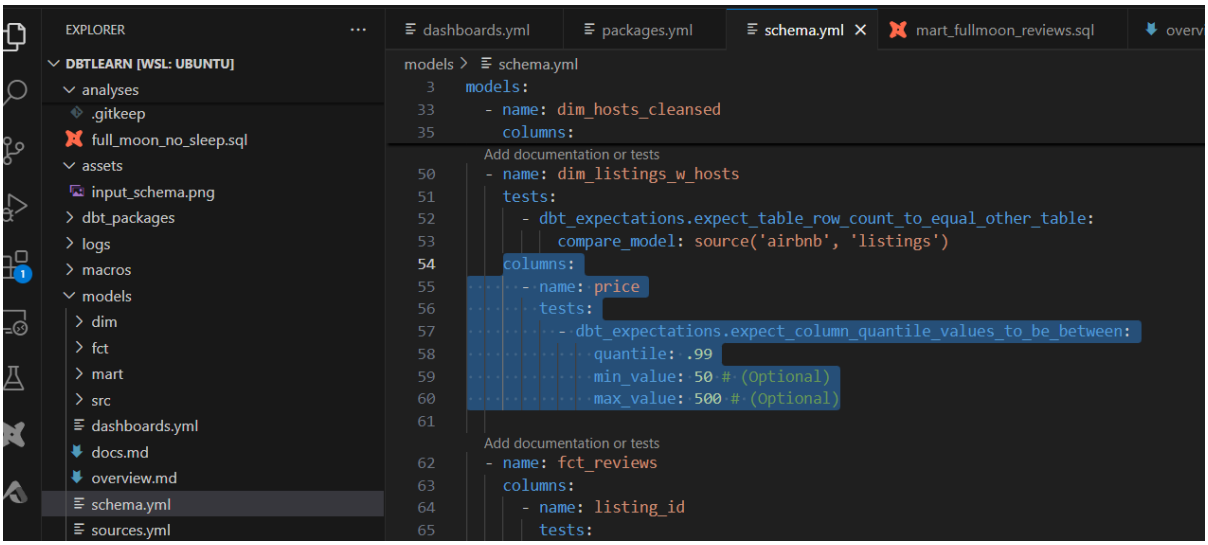
So, from the dbt_expectations package we will be using '*expect_column_quantile_values_to_be_between'* test. This will allow us to check the value in some range. Lets say 50-500. And since this will be applied on column – 'price' in dim_listings_w_hostings table we need to make sure we mention it as shown below with column parameter-

### expect_column_quantile_values_to_be_between

Expect specific provided column quantiles to be between provided min_value and max_value values.

*Applies to:* Column

```
tests:
  - dbt_expectations.expect_column_quantile_values_to_be_between:
      quantile: .95
      min_value: 0 # (Optional)
      max_value: 2 # (Optional)
      group_by: [group_id, other_group_id, ...] # (Optional)
      row_condition: "id is not null" # (Optional)
      strictly: false # (Optional. Default is 'false'. Adds an 'or equal to' to the comparison operator
```



```
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$ dbt test --select dim_listings_w_host
s
00:34:38  Running with dbt=1.9.3
00:34:42  Registered adapter: snowflake=1.9.0
00:35:01  Found 8 models, 1 snapshot, 1 analysis, 18 data tests, 1 seed, 3 sources, 1 exposure, 857 macros
00:35:01
00:35:01  Concurrency: 1 threads (target='dev')
00:35:01
00:35:05  1 of 2 START test dbt_expectations_expect_column_quantile_values_to_be_between_dim_listings_w_hosts_price__500
__50__0_99  [RUN]
00:35:06  1 of 2 PASS dbt_expectations_expect_column_quantile_values_to_be_between_dim_listings_w_hosts_price__500__50__
0_99  [PASS in 0.86s]
00:35:06  2 of 2 START test dbt_expectations_expect_table_row_count_to_equal_other_table_dim_listings_w_hosts_source_air
bnb_listings_  [RUN]
00:35:06  2 of 2 PASS dbt_expectations_expect_table_row_count_to_equal_other_table_dim_listings_w_hosts_source_airbnb_li
stings_  [PASS in 0.16s]
00:35:07
00:35:07  Finished running 2 data tests in 0 hours 0 minutes and 5.28 seconds (5.28s).
00:35:07
00:35:07  Completed successfully
00:35:07
00:35:07  Done. PASS=2 WARN=0 ERROR=0 SKIP=0 TOTAL=2
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$
```

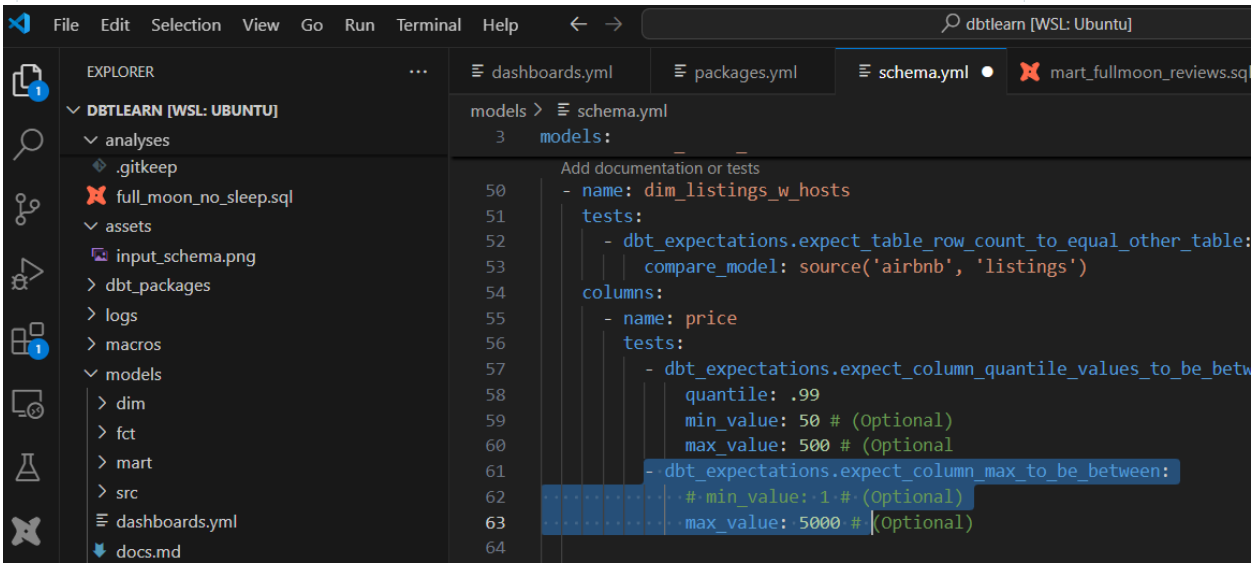
We can also set the max price for the column for which we can accept the data. This test would help in having a real data instead of a vague data which will be an outlier and would not make sense.

## expect_column_max_to_be_between

Expect the column max to be between a min and max value

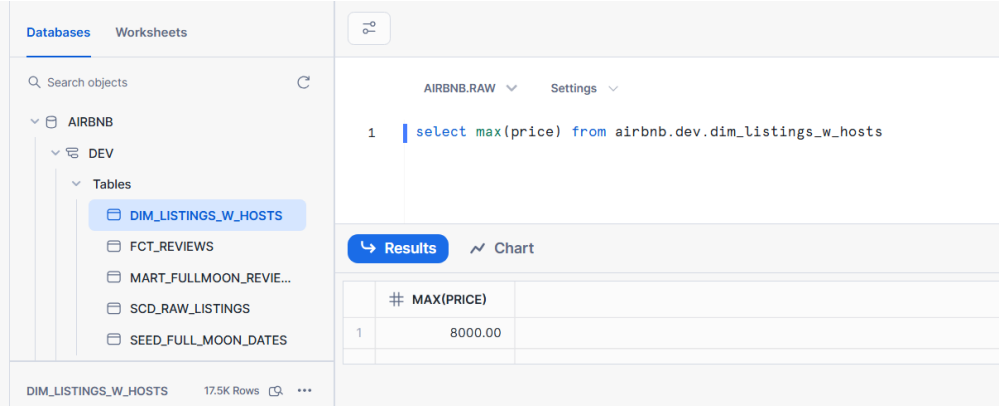
*Applies to:* Column

```
tests:
  - dbt_expectations.expect_column_max_to_be_between:
      min_value: 1 # (Optional)
      max_value: 1 # (Optional)
      group_by: [group_id, other_group_id, ...] # (Optional)
      row_condition: "id is not null" # (Optional)
      strictly: false # (Optional. Default is 'false'. Adds an 'or equal to' to the comparison operator
```



```
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$ dbt test --select dim_listings_w_hosts
00:42:52  Running with dbt=1.9.3
00:42:55  Registered adapter: snowflake=1.9.0
00:43:18  Found 8 models, 1 snapshot, 1 analysis, 19 data tests, 1 seed, 3 sources, 1 exposure, 857 macros
00:43:18
00:43:18  Concurrency: 1 threads (target='dev')
00:43:18
00:43:23  1 of 3 START test dbt_expectations_expect_column_max_to_be_between_dim_listings_w_hosts_price__5000  [RUN]
00:43:23  1 of 3 FAIL 1 dbt_expectations_expect_column_max_to_be_between_dim_listings_w_hosts_price__5000  [FAIL 1 in 0.55s]
00:43:23  2 of 3 START test dbt_expectations_expect_column_quantile_values_to_be_between_dim_listings_w_hosts_price__500__50__0_99
  [RUN]
00:43:23  2 of 3 PASS dbt_expectations_expect_column_quantile_values_to_be_between_dim_listings_w_hosts_price__500__50__0_99  [PAS
S in 0.14s]
00:43:23  3 of 3 START test dbt_expectations_expect_table_row_count_to_equal_other_table_dim_listings_w_hosts_source_airbnb_listin
gs_  [RUN]
00:43:24  3 of 3 PASS dbt_expectations_expect_table_row_count_to_equal_other_table_dim_listings_w_hosts_source_airbnb_listings_  [
PASS in 0.13s]
00:43:24
00:43:24  Finished running 3 data tests in 0 hours 0 minutes and 5.64 seconds (5.64s).
00:43:24
00:43:24  Completed with 1 error, 0 partial successes, and 0 warnings:
00:43:24
00:43:24  Failure in test dbt_expectations_expect_column_max_to_be_between_dim_listings_w_hosts_price__5000 (models/schema.yml)
00:43:24    Got 1 result, configured to fail if != 0
00:43:24
00:43:24    compiled code at target/compiled/dbtlearn/models/schema.yml/dbt_expectations_expect_column_c59e300e0dddb335c4211147100
ac1c6.sql
00:43:24
00:43:24  Done. PASS=2 WARN=0 ERROR=1 SKIP=0 TOTAL=3
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$
```

This shows that test failed which means that we have more expensive airbnb's than 5000 in price.




So, since now we can't say if this is true listing or not so to avoid errors we can change the severity from error to warn.

```
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$ dbt test --select dim_listings_w_hosts
00:48:59  Running with dbt=1.9.3
00:49:03  Registered adapter: snowflake=1.9.0
00:49:24  Found 8 models, 1 snapshot, 1 analysis, 19 data tests, 1 seed, 3 sources, 1 exposure, 857 macros
00:49:24
00:49:24  Concurrency: 1 threads (target='dev')
00:49:24
00:49:28  1 of 3 START test dbt_expectations_expect_column_max_to_be_between_dim_listings_w_hosts_price__5000  [RUN]
00:49:28  1 of 3 WARN 1 dbt_expectations_expect_column_max_to_be_between_dim_listings_w_hosts_price__5000  [WARN 1 in 0.17s]
00:49:28  2 of 3 START test dbt_expectations_expect_column_quantile_values_to_be_between_dim_listings_w_hosts_price__500__50__0_99
  [RUN]
00:49:28  2 of 3 PASS dbt_expectations_expect_column_quantile_values_to_be_between_dim_listings_w_hosts_price__500__50__0_99  [PAS
S in 0.17s]
00:49:28  3 of 3 START test dbt_expectations_expect_table_row_count_to_equal_other_table_dim_listings_w_hosts_source_airbnb_listin
gs_  [RUN]
00:49:29  3 of 3 PASS dbt_expectations_expect_table_row_count_to_equal_other_table_dim_listings_w_hosts_source_airbnb_listings_  [
PASS in 0.30s]
00:49:29
00:49:29  Finished running 3 data tests in 0 hours 0 minutes and 5.16 seconds (5.16s).
00:49:29
00:49:29  Completed with 1 warning:
00:49:29
00:49:29  Warning in test dbt_expectations_expect_column_max_to_be_between_dim_listings_w_hosts_price__5000 (models/schema.yml)
00:49:29  Got 1 result, configured to warn if != 0
00:49:29
00:49:29    compiled code at target/compiled/dbtlearn/models/schema.yml/dbt_expectations_expect_column_c59e300e0dddb335c4211147100
ac1c6.sql
00:49:29
00:49:29  Done. PASS=2 WARN=1 ERROR=0 SKIP=0 TOTAL=3
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$
```

This is how it has been changed from error to warning.

Let’s perform another test where we check for price column to be numerical value –

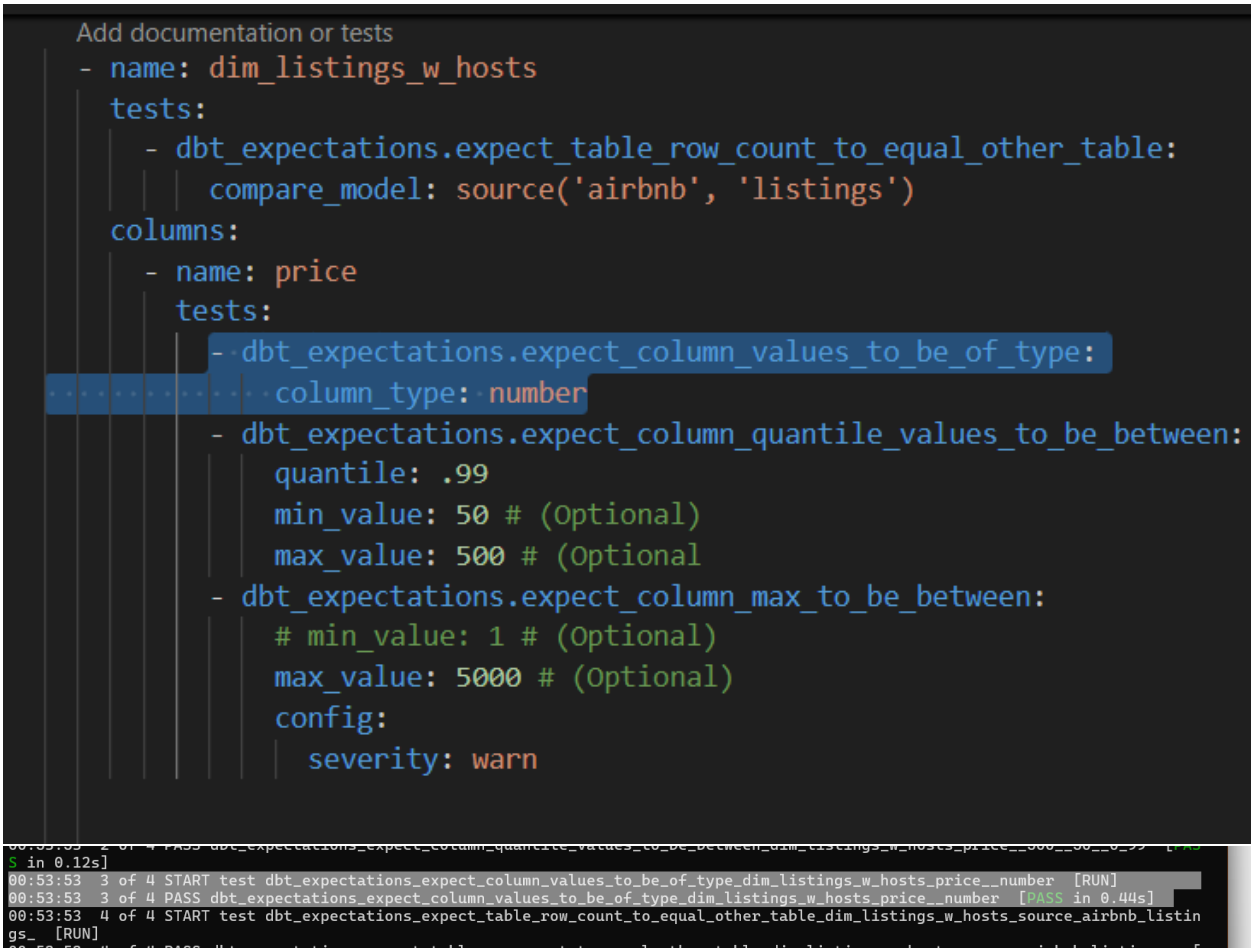
### expect_column_values_to_be_of_type

Expect a column to be of a specified data type.

*Applies to:* Column

```
tests:
  - dbt_expectations.expect_column_values_to_be_of_type:
      column_type: date
```

This is the entire price column test we are performing.

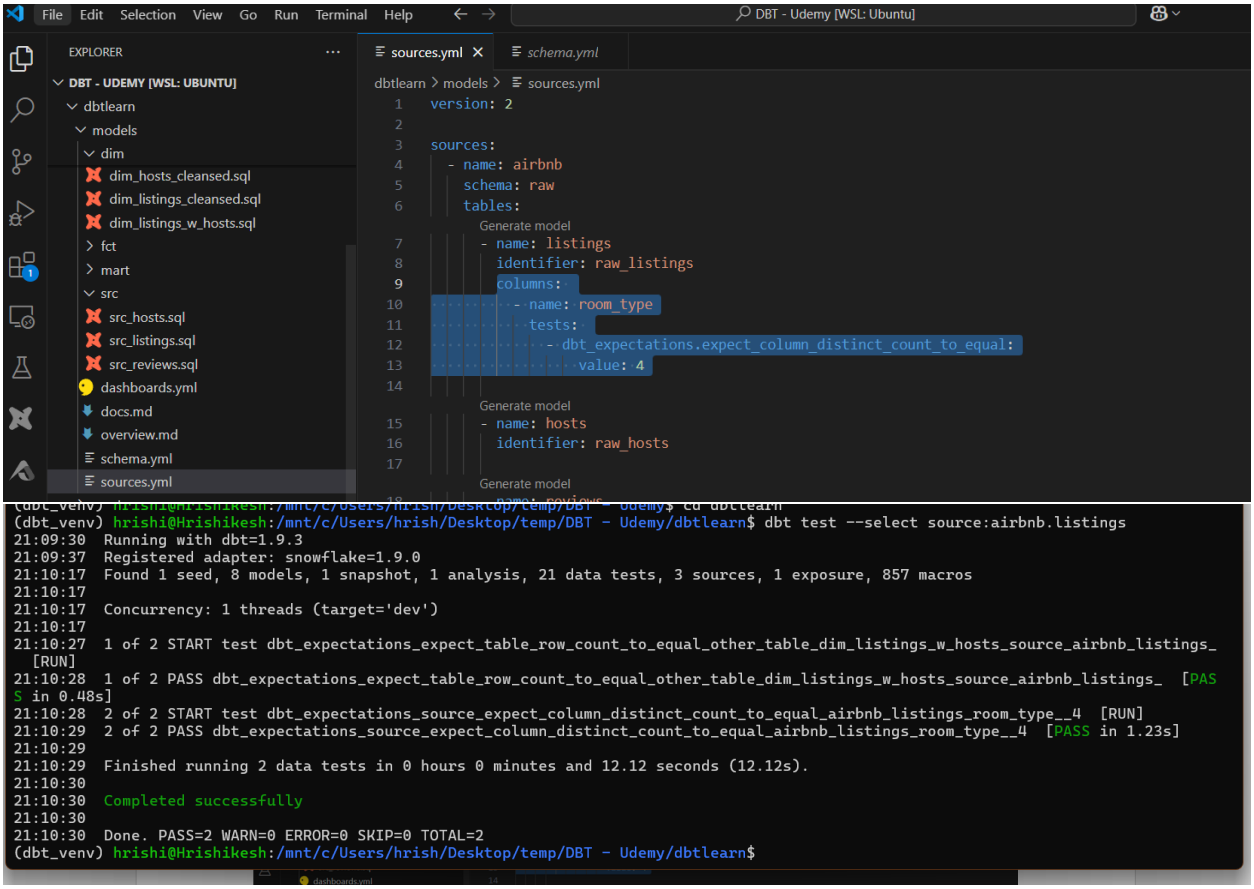



It shows test passed.

---

**Another important part of data pipeline is to test the source data.**

These tests are now being performed on the source data which are ingested in our model with data pipeline.




dbt also executed the tests which are associated or affected by source-listings table. Source tables are not dimensions they are models.

Now, the below test we are testing if our price column has only the accepted values. i.e. it starts with $ sign and then two digits then '.' And then two decimal points.

### expect_column_values_to_match_regex

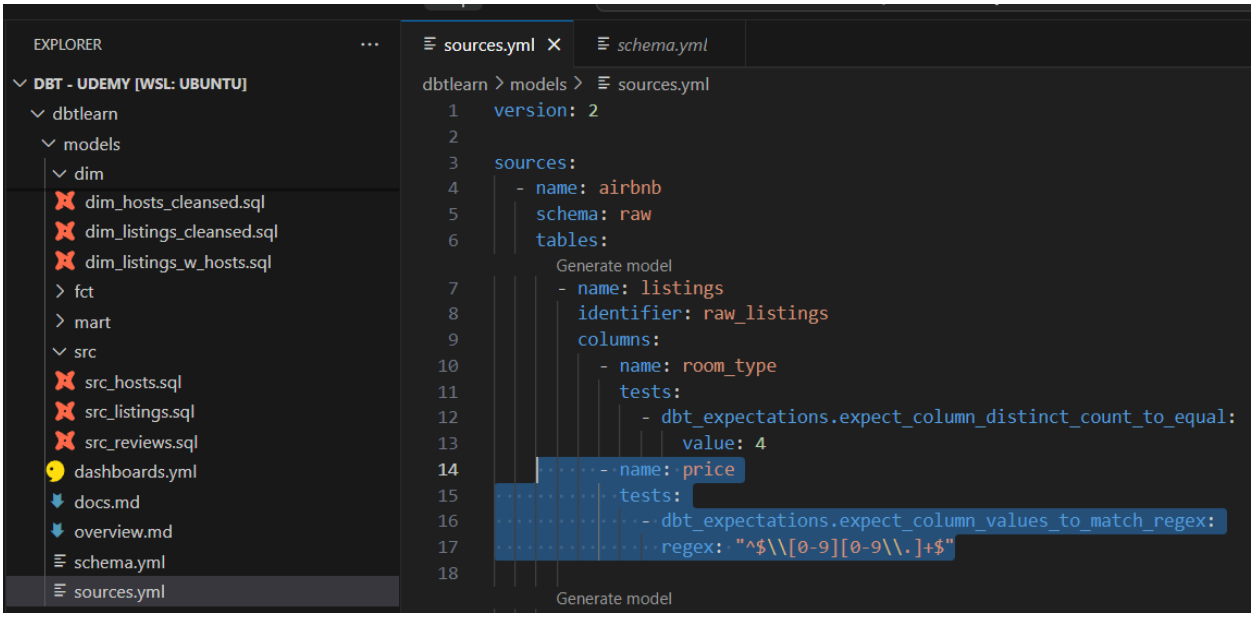
Expect column entries to be strings that match a given regular expression. Valid matches can be found anywhere in the string, for example "[at]+" will identify the following strings as expected: "cat", "hat", "aa", "a", and "t", and the following strings as unexpected: "fish", "dog".

Optional (keyword) arguments:

- `is_raw` indicates the `regex` pattern is a "raw" string and should be escaped. The default is `False`.
- `flags` is a string of one or more characters that are passed to the `regex` engine as flags (or parameters). Allowed flags are adapter-specific. A common flag is `i`, for case-insensitive matching. The default is no flags.

*Applies to:* Column

```
tests:
  - dbt_expectations.expect_column_values_to_match_regex:
      regex: "[at]+"
      row_condition: "id is not null" # (Optional)
      is_raw: True # (Optional)
      flags: i # (Optional)
```




Depending on YAML or Jinja escaping, you might see double backslashes \\. instead of a single backslash \..

Here's how it breaks down:

1. **^**
   The caret (^) means "start of the string." This ensures we're checking from the very beginning.
2. **[0-9][0-9]**
   This matches exactly **two digits** in a row. For example, 00, 12, 99, etc.
3. **.**
   The backslash-escaped dot (\.) matches a **literal period** . (rather than "any character").
4. **.+**
   The dot (.) inside a regex means "match any character," and the plus sign (+) means "one or more times." So .+ matches **one or more of any characters** after the period.
5. **$**
   The dollar sign ($) means "end of the string." It ensures we're checking until the very end and not allowing extra characters after this pattern.

```
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$ dbt test --select source:airbnb.listings
21:36:47  Running with dbt=1.9.3
21:36:51  Registered adapter: snowflake=1.9.0
21:37:12  Found 1 seed, 8 models, 1 snapshot, 1 analysis, 22 data tests, 3 sources, 1 exposure, 857 macros
21:37:12
21:37:12  Concurrency: 1 threads (target='dev')
21:37:12
21:37:16  1 of 3 START test dbt_expectations_expect_table_row_count_to_equal_other_table_dim_listings_w_hosts_source_airbnb_listings_
  [RUN]
21:37:16  1 of 3 PASS dbt_expectations_expect_table_row_count_to_equal_other_table_dim_listings_w_hosts_source_airbnb_listings_  [PAS
S in 0.26s]
21:37:16  2 of 3 START test dbt_expectations_source_expect_column_distinct_count_to_equal_airbnb_listings_room_type__4  [RUN]
21:37:17  2 of 3 PASS dbt_expectations_source_expect_column_distinct_count_to_equal_airbnb_listings_room_type__4  [PASS in 0.19s]
21:37:17  3 of 3 START test dbt_expectations_source_expect_column_values_to_match_regex_airbnb_listings_price___0_9_0_9_  [RUN]
21:37:18  3 of 3 FAIL 17499 dbt_expectations_source_expect_column_values_to_match_regex_airbnb_listings_price___0_9_0_9_  [FAIL 17499
 in 1.12s]
21:37:18
21:37:18  Finished running 3 data tests in 0 hours 0 minutes and 5.93 seconds (5.93s).
21:37:18
21:37:18  Completed with 1 error, 0 partial successes, and 0 warnings:
21:37:18
21:37:18  Failure in test dbt_expectations_source_expect_column_values_to_match_regex_airbnb_listings_price___0_9_0_9_ (models/source
s.yml)
21:37:18    Got 17499 results, configured to fail if != 0
21:37:18
21:37:18    compiled code at target/compiled/dbtlearn/models/sources.yml/dbt_expectations_source_expect_a60b59a84fbc4577a11df360c5001
3bb.sql
21:37:18
21:37:18  Done. PASS=2 WARN=0 ERROR=1 SKIP=0 TOTAL=3
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$
```

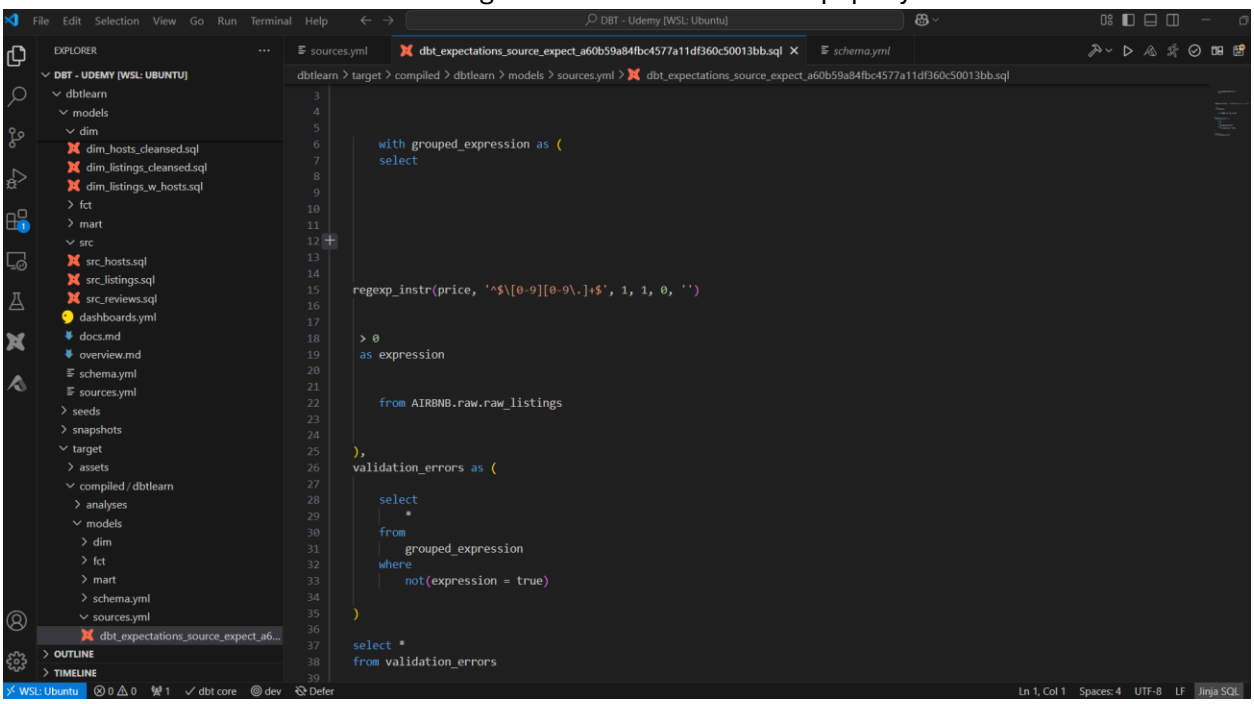
so, the output for the above regex test shows failed status with 17499 records failing for the test. to tackle failed tests you can always use the debug method to look whats actually happening in the dbt.

Use this command to debug – *dbt -–debug test –select source:airbnb.listings*

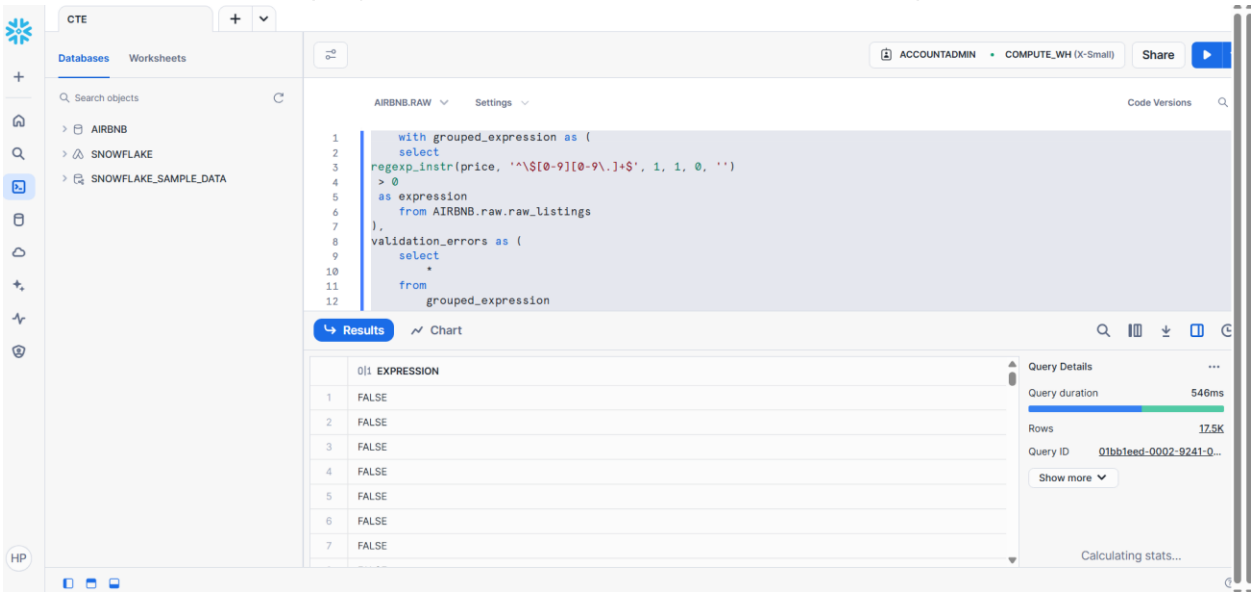
So this will return a big debug screen but what you need to look is for the specific test being performed for which the error occurred and its sql file -

```
hrishi@Hrishikesh: /mnt/c/Us
)

select *
from validation_errors




    ) dbt_internal_test
21:43:43  SQL status: SUCCESS 1 in 0.124 seconds
21:43:43  3 of 3 FAIL 17499 dbt_expectations_source_expect_column_values_to_match_regex_airbnb_listings_price___0_9_0_9_  [FAIL 17499 in 0.17s]
21:43:43  Finished running node test.dbtlearn.dbt_expectations_source_expect_column_values_to_match_regex_airbnb_listings_price___0_9_0_9_.0f20c8833a
21:43:43  Connection 'master' was properly closed.
21:43:43  Connection 'test.dbtlearn.dbt_expectations_source_expect_column_values_to_match_regex_airbnb_listings_price___0_9_0_9_.0f20c8833a' was left open.
21:43:43  On test.dbtlearn.dbt_expectations_source_expect_column_values_to_match_regex_airbnb_listings_price___0_9_0_9_.0f20c8833a: Close
21:43:43
21:43:43  Finished running 3 data tests in 0 hours 0 minutes and 4.54 seconds (4.54s).
21:43:43  Command end result
21:43:43  Wrote artifact WritableManifest to /mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn/target/manifest.json
21:43:43  Wrote artifact SemanticManifest to /mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn/target/semantic_manifest.json
21:43:43  Wrote artifact RunExecutionResult to /mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn/target/run_results.json
21:43:43
21:43:43  Completed with 1 error, 0 partial successes, and 0 warnings:
21:43:43
21:43:43  Failure in test dbt_expectations_source_expect_column_values_to_match_regex_airbnb_listings_price___0_9_0_9_ (models/sources.yml)
21:43:43    Got 17499 results, configured to fail if != 0
21:43:43
21:43:43    compiled code at target/compiled/dbtlearn/models/sources.yml/dbt_expectations_source_expect_a60b59a84fbc4577a11df360c50013bb.sql
21:43:43
21:43:43  Done. PASS=2 WARN=0 ERROR=1 SKIP=0 TOTAL=3
21:43:43  Resource report: {"command_name": "test", "command_success": false, "command_wall_clock_time": 27.17321, "process_in_blocks": "27120", "process_ke
rnel_time": 2.458787, "process_mem_max_rss": "134788", "process_out_blocks": "16", "process_user_time": 2.96079}
21:43:43  Command `dbt test` failed at 16:43:43.961298 after 27.18 seconds
21:43:43  Sending event: {'category': 'dbt', 'action': 'invocation', 'label': 'end', 'context': [<snowplow_tracker.self_describing_json.SelfDescribingJson o
bject at 0x7fd6a351a350>, <snowplow_tracker.self_describing_json.SelfDescribingJson object at 0x7fd6a355e650>, <snowplow_tracker.self_describing_json.SelfDe
scribingJson object at 0x7fd6a0cdfc90>]}
21:43:43  Flushing usage events
21:43:44  An error was encountered while trying to flush usage events
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$
```

Used code command and then the target folder link to view the sql query in code editor -

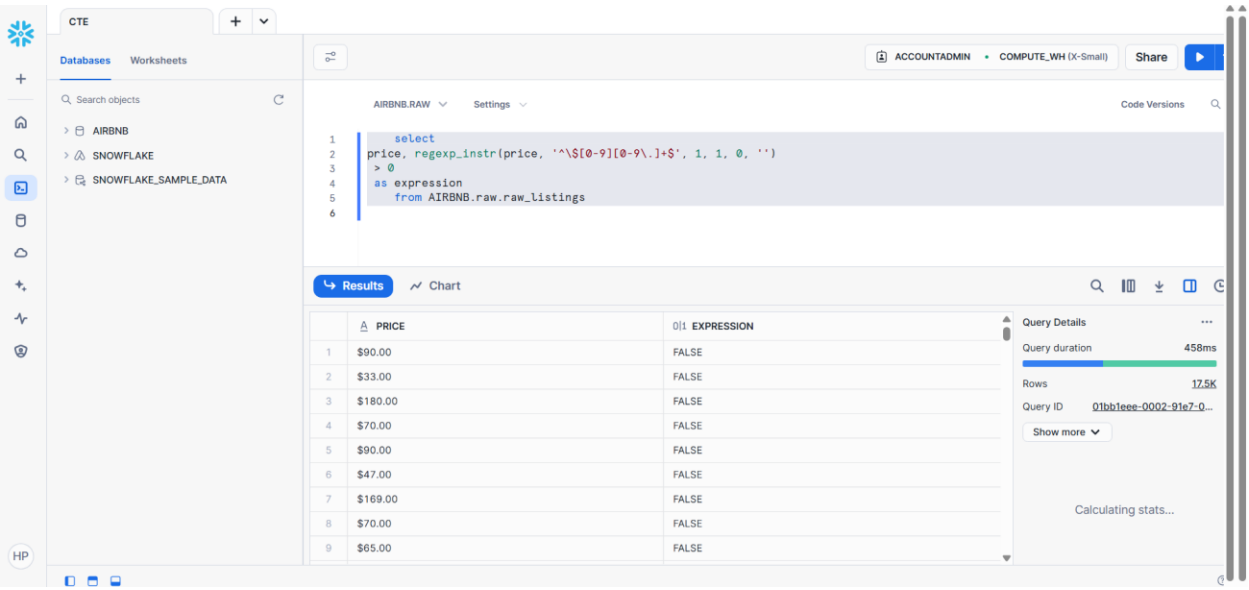



now we'll look how this query runs in our data warehouse and what it outputs.




looking at the output I believe it failed for almost every single row.

making some changes In the query to look and compare at the formats it failed.




Suspecting a error in the regex logic. Since with correct format it says the expression is false. This might be problem with a single back slash '\' which the snowflake converted automatically.

AIRBNB.RAW Settings

```
   select
price, regexp_instr(price, '^\\$[0-9][0-9\\.]+$', 1, 1, 0, '')
 > 0
 as expression
   from AIRBNB.raw.raw_listings
```

Results Chart

| | PRICE | EXPRESSION |
|---|---|---|
| 1 | $90.00 | TRUE |
| 2 | $33.00 | TRUE |
| 3 | $180.00 | TRUE |
| 4 | $70.00 | TRUE |
| 5 | $90.00 | TRUE |
| 6 | $47.00 | TRUE |
| 7 | $169.00 | TRUE |
| 8 | $70.00 | TRUE |
| 9 | $65.00 | TRUE |
| 10 | $120.00 | TRUE |

After adding the doubt back slash we got the expressions true. So the snowflake also required double backslash '\\'

So the simple way to solve this at times is to simple use lame methods hahaha – use 4 backslashesh –

```
          value: 4
      - name: price
        tests:
          - dbt_expectations.expect_column_values_to_match_regex:
              regex: "^\\\\$[0-9][0-9\\\\.]+$"

Generate model
- name: hosts
```

So that yaml will strip down automatically two of them and the snowflake will receive the remaining two backslashes which executes the query perfectly.

```
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$ dbt test --select source:airbnb.listings
22:12:51  Running with dbt=1.9.3
22:12:54  Registered adapter: snowflake=1.9.0
22:13:14  Found 1 seed, 8 models, 1 snapshot, 1 analysis, 22 data tests, 3 sources, 1 exposure, 857 macros
22:13:14
22:13:14  Concurrency: 1 threads (target='dev')
22:13:14
22:13:17  1 of 3 START test dbt_expectations_expect_table_row_count_to_equal_other_table_dim_listings_w_hosts_source_airbnb_listings_  [RUN]
22:13:18  1 of 3 PASS dbt_expectations_expect_table_row_count_to_equal_other_table_dim_listings_w_hosts_source_airbnb_listings_  [PASS in 0.21s]
22:13:18  2 of 3 START test dbt_expectations_source_expect_column_distinct_count_to_equal_airbnb_listings_room_type__4  [RUN]
22:13:18  2 of 3 PASS dbt_expectations_source_expect_column_distinct_count_to_equal_airbnb_listings_room_type__4  [PASS in 0.15s]
22:13:18  3 of 3 START test dbt_expectations_source_expect_column_values_to_match_regex_airbnb_listings_price___0_9_0_9_  [RUN]
22:13:18  3 of 3 PASS dbt_expectations_source_expect_column_values_to_match_regex_airbnb_listings_price___0_9_0_9_  [PASS in 0.56s]
22:13:18
22:13:18  Finished running 3 data tests in 0 hours 0 minutes and 4.86 seconds (4.86s).
22:13:18
22:13:18  Completed successfully
22:13:18
22:13:18  Done. PASS=3 WARN=0 ERROR=0 SKIP=0 TOTAL=3
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$
```

and the test passed successfully.

sources.yml | dbt_expectations_source_expect_a60b59a84fbc4577a11df360c50013bb.sql | schema.yml

dbtlearn > target > compiled > dbtlearn > models > sources.yml > dbt_expectations_source_expect_a60b59a84fbc4577a11df360

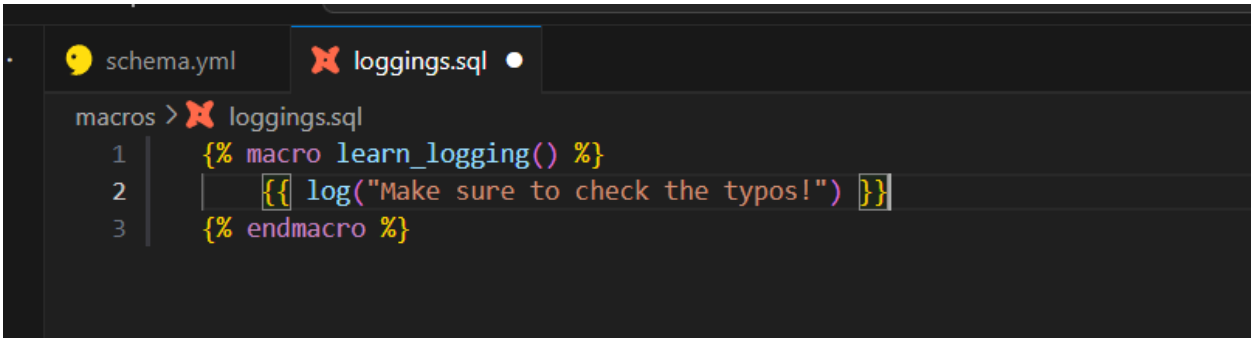
```
3
4
5
6       with grouped_expression as (
7       select
8
9
10
11
12
13
14
15  regexp_instr(price, '^\\$[0-9][0-9\\.]+$', 1, 1, 0, '')
16
17
18      > 0
19      as expression
20
21
```

because our yaml was able to keep two out of four ‘\’

**DEBUGGING WITH LOGGING**

In dbt, **logging** refers to the structured recording of everything dbt does during a run — from compiling models and executing SQL to reporting compilation errors, runtime warnings, and performance timings. dbt writes these messages both to your console (stdout) and to a persistent log file so you can audit what happened (and when) after the fact.

Created logging file in macro -




Logs that message at the **INFO** level (default). You'll see it in your console and in target/logs/dbt.log. here by default the log() messages are info – level.

This will execute the message in the logs folder.

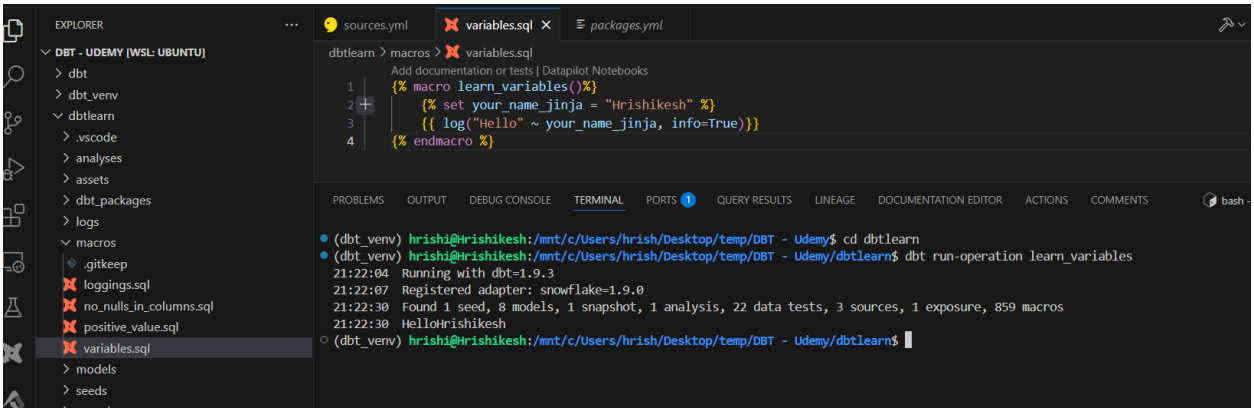
---

**Variables**

- Use Jinja variables for one-off template logic.
- Use dbt variables to externalize configuration so you can change behavior without touching SQL code.

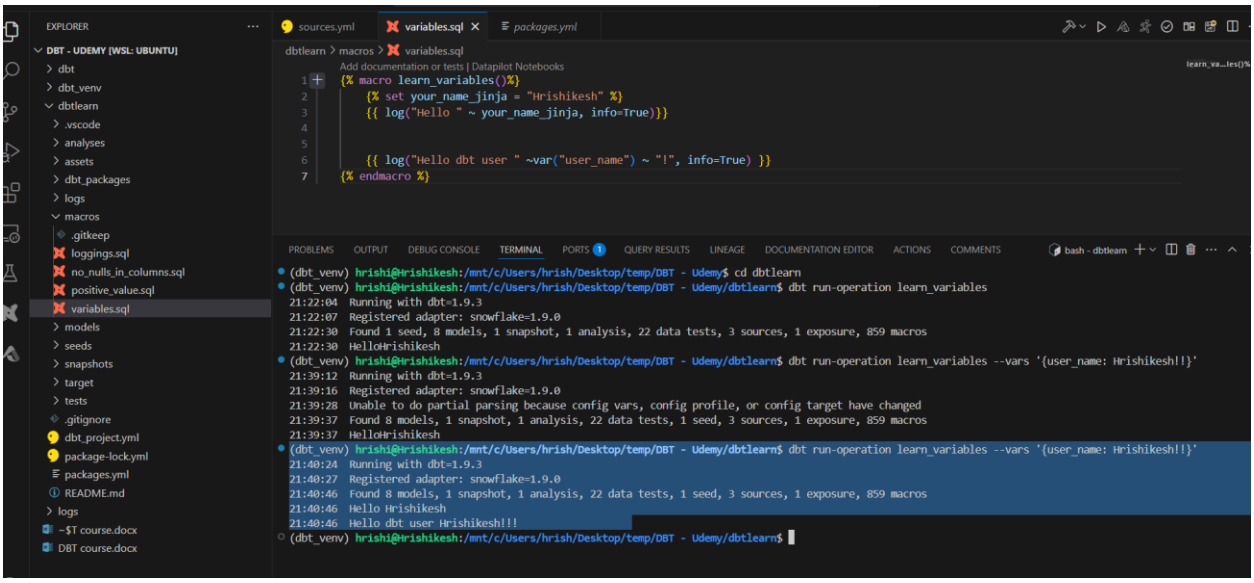
**Jinja variables**

- You need a local temporary placeholder for string concatenation, loops, or logic inside a macro or model.
- Scope never escapes the file or macro where it's declared.-



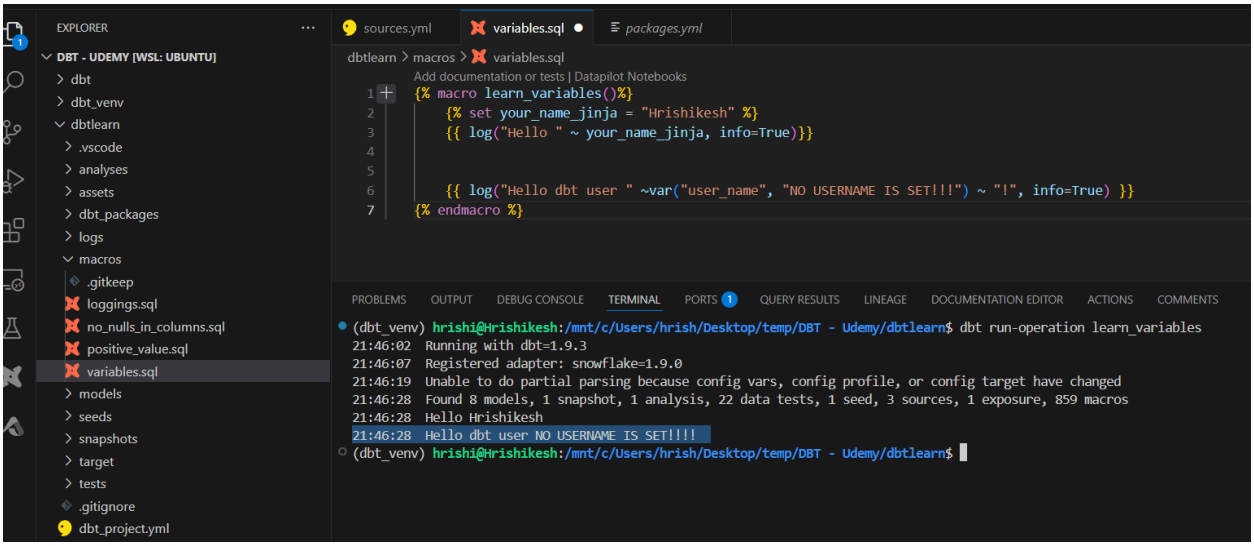
## dbt variables

- You want to configure behavior from outside your SQL (e.g., toggle a feature flag, supply a schema name, change a threshold).
- You want to override a value per environment (dev vs prod), per invocation (dbt run --vars '{"threshold": 100}'), or at model level (via vars: in dbt_project.yml).
- These are also called as project vairbales, and are managed by dbt. Can be passed through command lines as and when needed.



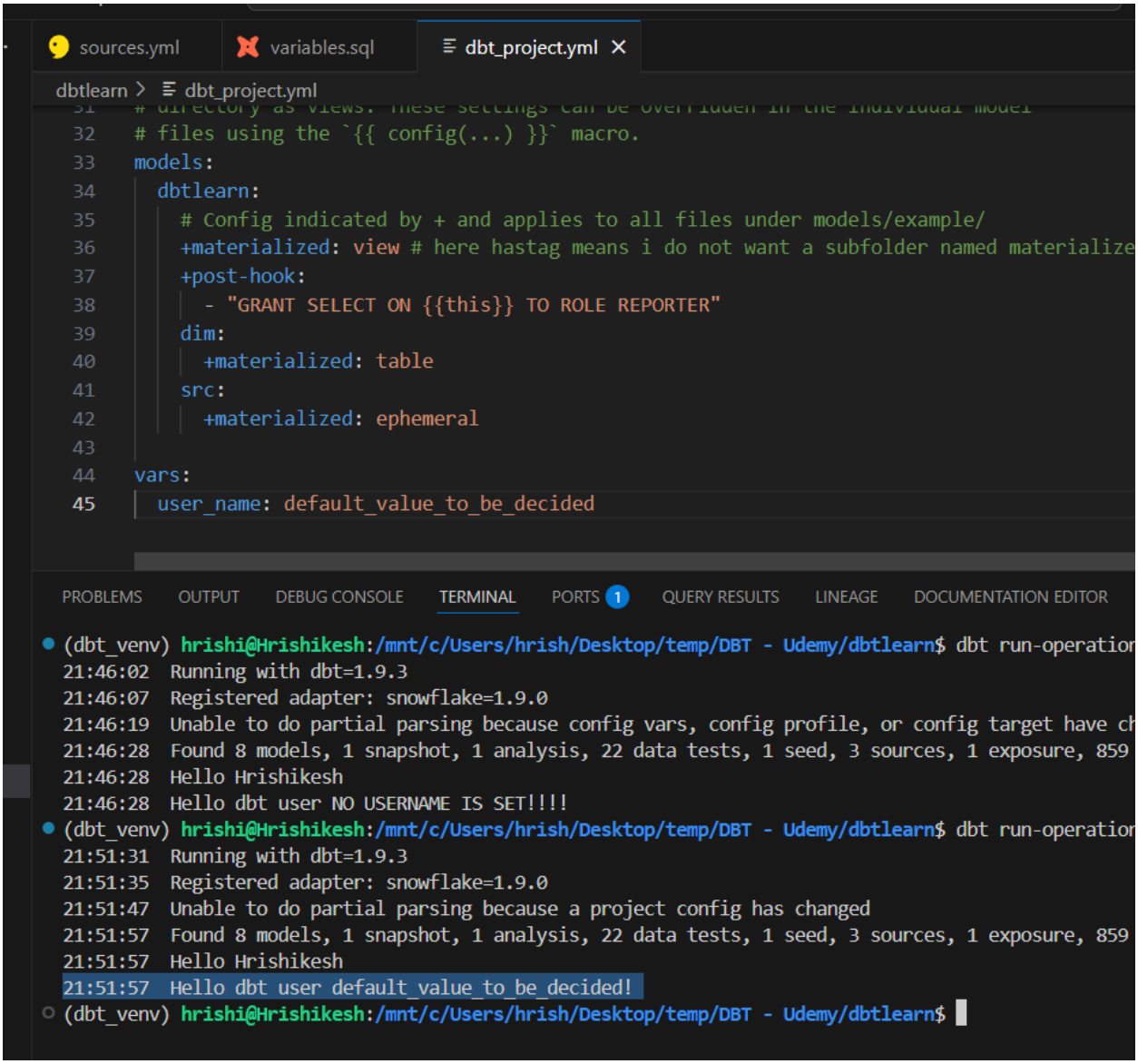

Here I have explicitly mentioned the variable name with the value – '{user_name: Hrishikesh}'
We, don't necessarily have to come and execute the variable name and value in the command line, we can avoid this and can mention this once in the dbt_project.yml

**What if there's no value passed for the variable?**
**To avoid this you can set a default value –**




After you define a var in jinja temp log() function you can add the default value after the variable name.

**Other method** is to define the default value in dbt_project.yml –

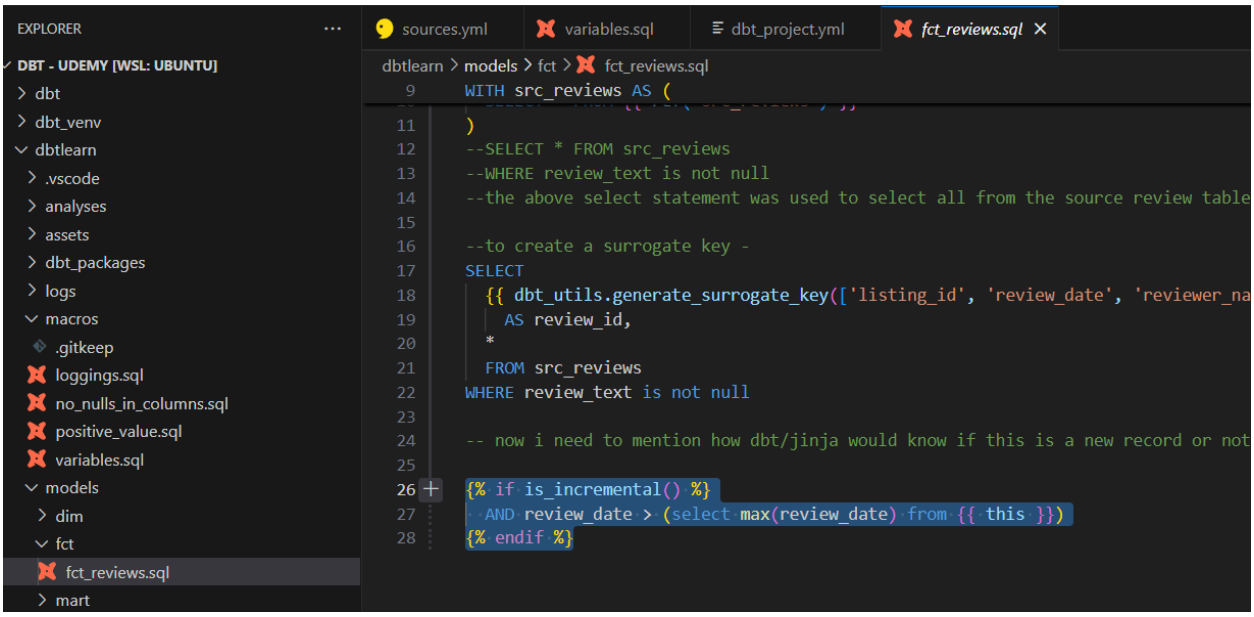



So, now what happened here?

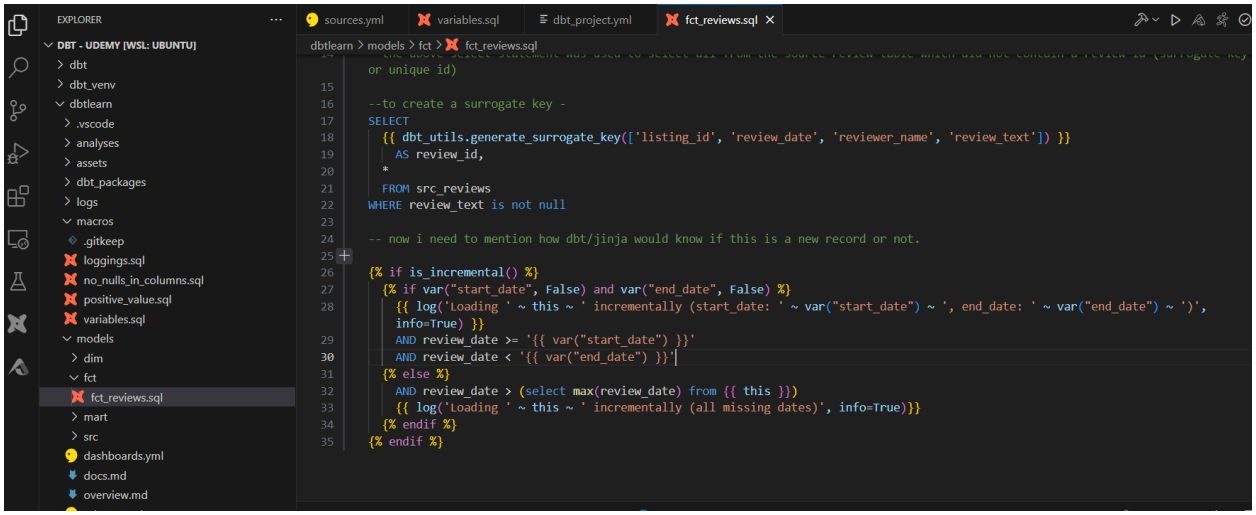
Whenever you execute dbt, it will first pick all the variable name mentioned in the dbt_prject.yml and will be treated as default. But when you explicitly give the variable value through cli then it will get override.

**NOTE:** The variable value mentioned with the variable definition is executed only when there's no cli input and no default var value mentioned.

Now, we'll make changes to fct_reviews table so that we can achieve incremental load when we pass a date range. This is done because in production env if some things goes wrong and we need to implement incremental load to a specific date range data and not the entire fact table. This will also save the compute cost in the warehouse.




This snippet is part of an incremental model in dbt. When you run dbt normally (without ***--full-refresh***), dbt treats the model as incremental and ***is_incremental()*** returns true. In that case, the code inside the ***{% if is_incremental() %} ... {% endif %}*** block is added to your SELECT statement. Specifically, it appends an AND-clause that only selects rows whose review_date is greater than the maximum ***review_date*** already stored in the target table. The expression ***SELECT MAX(review_date) FROM {{ this }}*** queries the existing version of this model's table (where ***{{ this }}*** refers to the current model). By filtering dates newer than that maximum, dbt only loads new records on each incremental run, avoiding reprocessing rows you have already inserted. If you run a full refresh***, is_incremental()*** returns false, the filter is skipped, and the model rebuilds completely from scratch.




- Now, **Original logic** only ran when ***is_incremental()*** was true, and simply added every row whose ***review_date*** was greater than the maximum date already in the target table (***SELECT MAX(review_date) FROM {{ this }}).***
- **New logic** still checks ***is_incremental()***, but first looks for two dbt variables (start_date and end_date).
    - If both vars are provided, it filters to only load rows where ***review_date*** falls between those dates.
    - Otherwise (***no vars***), it falls back to the original behavior of loading rows newer than the existing max review_date.

In both cases the incremental block runs only on non-full refreshes, but the new version lets you **override the date filter manually** by passing ***--vars '{"start_date":"YYYY-MM-DD","end_date":"YYYY-MM-DD"}'***.

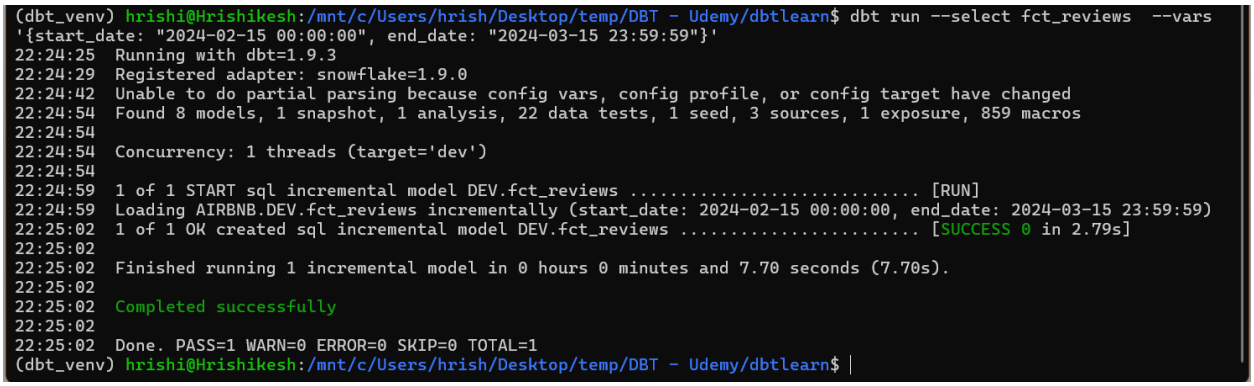
This is a standard practice to parameterize.

```
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$ dbt run --select fct_reviews
22:20:06  Running with dbt=1.9.3
22:20:10  Registered adapter: snowflake=1.9.0
22:20:29  Found 8 models, 1 snapshot, 1 analysis, 22 data tests, 1 seed, 3 sources, 1 exposure, 859 macros
22:20:29
22:20:29  Concurrency: 1 threads (target='dev')
22:20:29
22:20:34  1 of 1 START sql incremental model DEV.fct_reviews ............................. [RUN]
22:20:35  Loading AIRBNB.DEV.fct_reviews incrementally (all missing dates)
22:20:38  1 of 1 OK created sql incremental model DEV.fct_reviews ........................ [SUCCESS 0 in 3.27s]
22:20:38
22:20:38  Finished running 1 incremental model in 0 hours 0 minutes and 8.53 seconds (8.53s).
22:20:38
22:20:38  Completed successfully
22:20:38
22:20:38  Done. PASS=1 WARN=0 ERROR=0 SKIP=0 TOTAL=1
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$
```

We ran fct_reviews with no parameters and will be treated as incremental load (remember the original logic) it will check the review date and will see if new are added and then the new data will be added. This isn't full refresh.

dbt sees what the newest review date is in the already-loaded table, then only pulls in newer reviews. It **doesn't rebuild** the whole table — it just appends any new data since the last run

now, with parameters-




**dbt looked only at reviews in your specified date window**, added any of those it didn't already have, and left everything else untouched.

***dbt's simple incremental filter (review_date > MAX(review_date)) only adds new records; it won't overwrite or patch rows that already exist.***

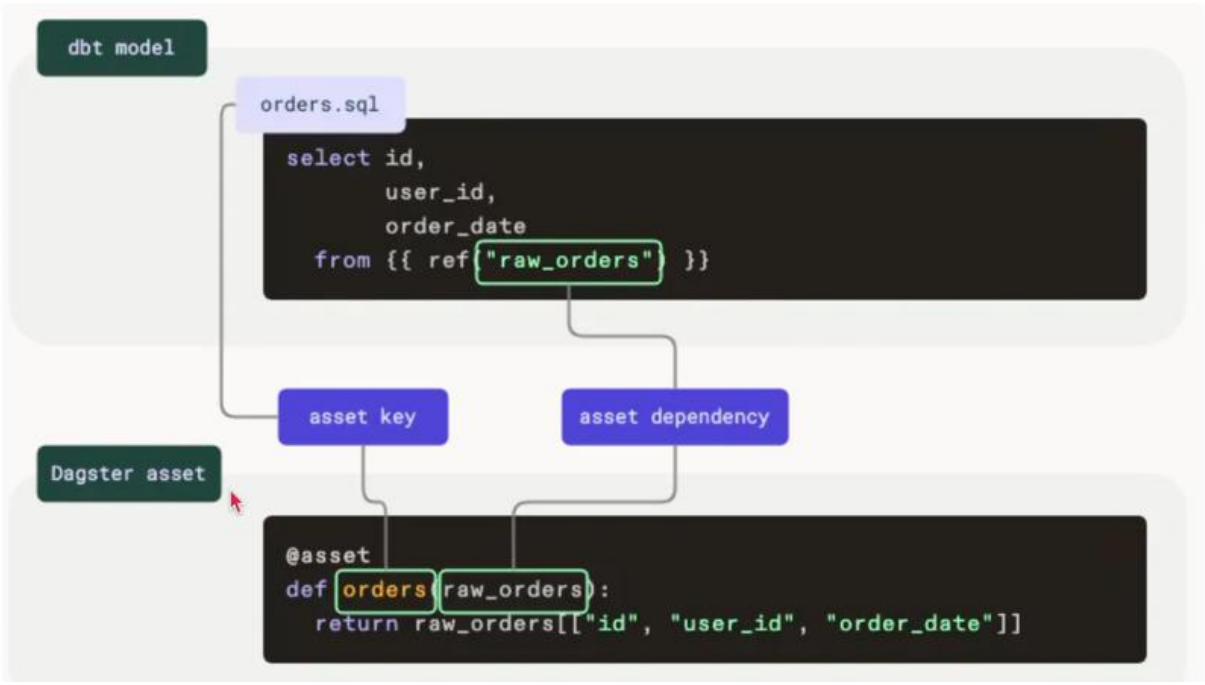
**ORCHESTRATION**

Orchestration in dbt means managing the order and timing of your data transformation tasks. It ensures that your models run in the correct sequence, so that each step has the data it needs from previous steps. This automation is important because it makes your data pipeline reliable, efficient, and easy to maintain

Dagster is often seen as a great partner for dbt because it's designed to understand and work with modern data transformation workflows. In simple terms:

- **Built for Data Pipelines**: Dagster treats pipelines as graphs where each step (like a dbt model) is a node. This makes it easier to manage dependencies and see how data flows through your transformations.
- **Seamless Integration**: Dagster has built-in support and integrations for dbt, which means less custom code and configuration to run dbt tasks as part of larger pipelines.
- **Observability and Testing**: It offers robust tools for monitoring, error handling, and testing, which align well with dbt's focus on data quality.
- **Developer Experience**: Compared to more generic orchestrators like Airflow, Data Factory, or Prefect, or even managed services like dbt Cloud, Dagster's modern, code-first approach fits naturally with dbt's philosophy making it simpler to build, debug, and maintain end-to-end pipelines.

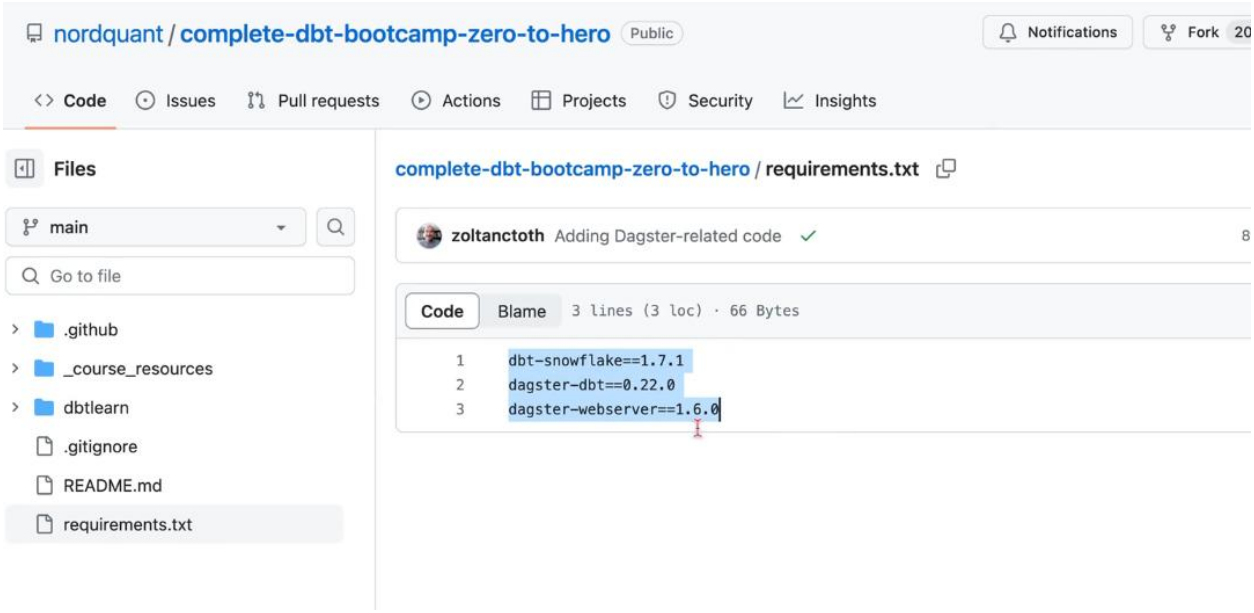
In short, Dagster's design and features complement dbt's strengths, making it easier to orchestrate, monitor, and manage your dbt models within a broader data pipeline.




A **dbt model** is essentially a SQL file that defines a transformation. When you run dbt, that file is compiled and executed to create a database object (typically a table or view) in your warehouse. It's version-controlled and focuses on transforming raw data into analytics-ready data using SQL.

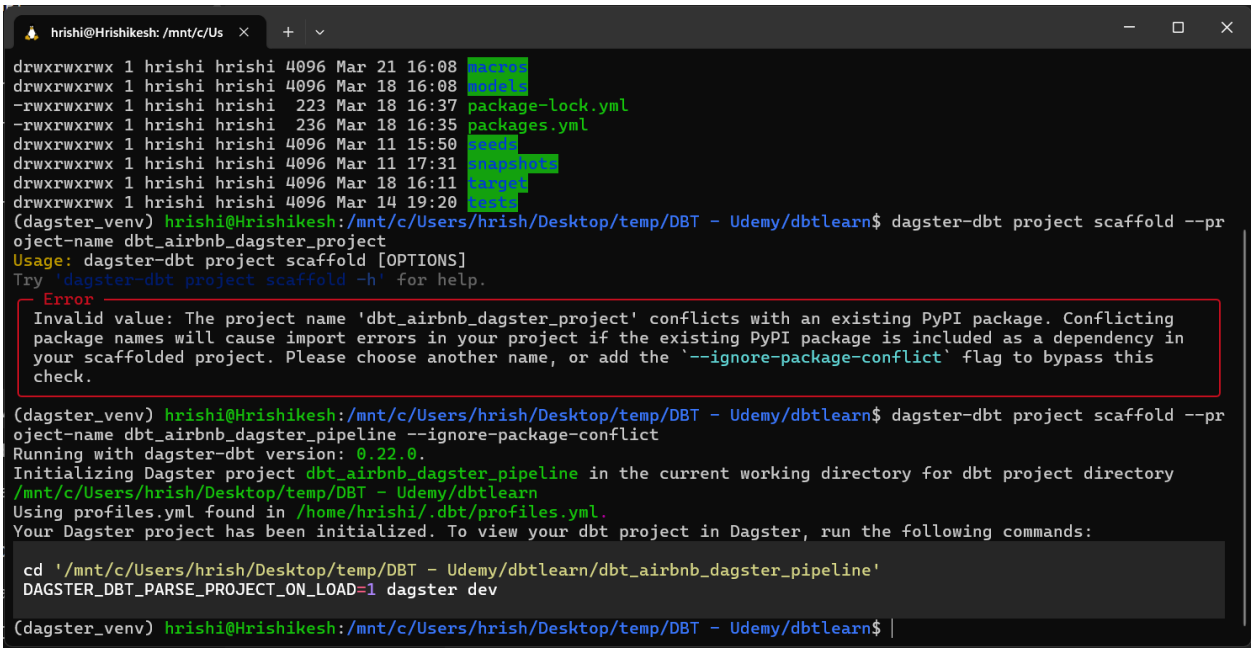
A **Dagster asset** is a broader concept. In Dagster, an asset represents any data product produced by your pipeline. It could be a table, a file, a dataset, or any computed output. Dagster assets come with built-in tracking of metadata, lineage, and dependencies, and they aren't limited to SQL transformations; they can be produced by Python code or other compute steps.

Installed dagster with appropriate dependencies –




```
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy$ python3.10 -m venv dagster_venv
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy$ source dagster_venv/bin/activate
(dagster_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy$ pip install -r requirements.txt
Collecting dbt-snowflake==1.7.1
  Downloading dbt_snowflake-1.7.1-py3-none-any.whl (46 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 46.2/46.2 KB 1.4 MB/s eta 0:00:00
Collecting dagster-dbt==0.22.0
  Downloading dagster_dbt-0.22.0-py3-none-any.whl (81 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 81.9/81.9 KB 3.2 MB/s eta 0:00:00
Collecting dagster-webserver==1.6.0
  Downloading dagster_webserver-1.6.0-py3-none-any.whl (10.2 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 10.2/10.2 MB 11.3 MB/s eta 0:00:00
Collecting snowflake-connector-python[secure-local-storage]~=3.0
  Downloading snowflake_connector_python-3.14.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (2.5 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.5/2.5 MB 15.8 MB/s eta 0:00:00
Collecting dbt-core~=1.7.0
  Downloading dbt_core-1.7.19-py3-none-any.whl (1.0 MB)
```

Used these specific dependencies so that dagster runs smoothly on these stable versions as the newest versions are not yet stably supported. Also, do not forget to change the venv in vscode.



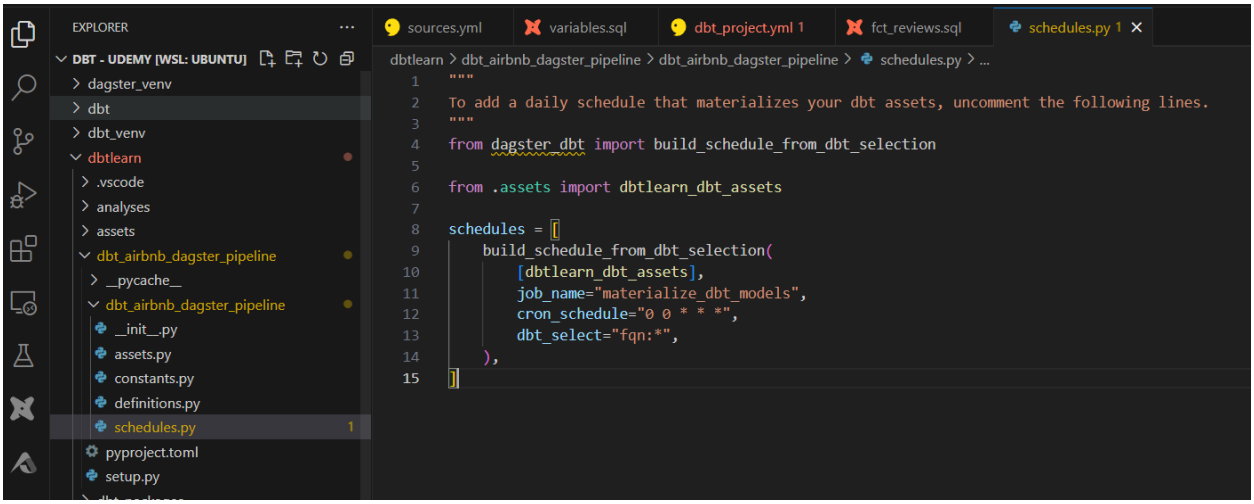

Encountered these errors even with unique names but for now its safe to bypass the error.

```
cd '/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn/dbt_airbnb_dagster_pipeline'
DAGSTER_DBT_PARSE_PROJECT_ON_LOAD=1 dagster dev
```

This command does two things:

1. **Changes Directory** to the newly created Dagster project folder (cd ...dbt_airbnb_dagster_pipeline).
2. **Sets an Environment Variable** (DAGSTER_DBT_PARSE_PROJECT_ON_LOAD=1) before running dagster dev. This tells Dagster to parse the dbt project as soon as the server starts, ensuring dbt models are recognized and available in the Dagster UI.

Now, the schedule in dagster needs to be explicitly enabled. We need to go and enable in python explicitly.

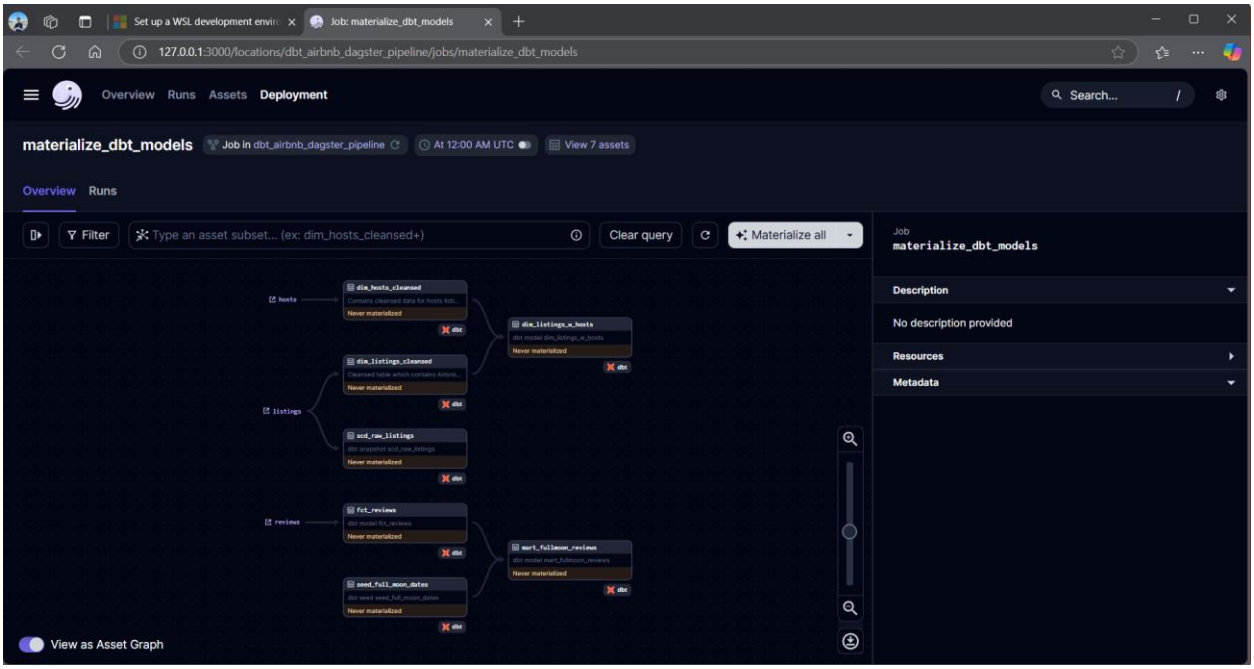



By uncommenting the schedules we have enabled schedules in dagster.

Now, its time to launch dagster.

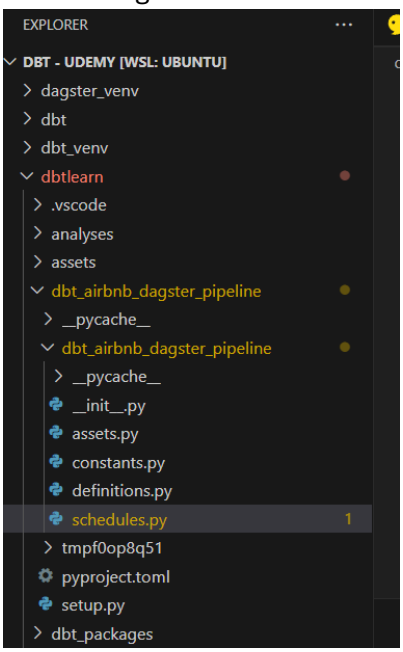
```
(dagster_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy$ ls
'DBT course.docx'   dbt          dbtlearn   requirements.txt  '~WRL2269.tmp'
 dagster_venv       dbt_venv     logs       '~$T course.docx'
(dagster_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy$ cd dbtlearn
(dagster_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$ ls
README.md  assets                dbt_packages      logs          models            packages.yml  snapshots  tests
analyses   dbt_airbnb_dagster_pipeline  dbt_project.yml  macros  package-lock.yml  seeds         target
(dagster_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$ cd dbt_airbnb_dagster_pipeline
(dagster_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn/dbt_airbnb_dagster_pipeline$ ls
__pycache__  dbt_airbnb_dagster_pipeline  pyproject.toml  setup.py
(dagster_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn/dbt_airbnb_dagster_pipeline$ DAGSTER_DBT_PARSE_PROJECT_ON_LOAD=1 dagst
er dev
2025-03-23 14:35:36 -0500 - dagster - INFO - Using temporary directory /mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn/dbt_airbnb_dagster_pipeline/tmpf
0op8q51 for storage. This will be removed when dagster dev exits.
2025-03-23 14:35:36 -0500 - dagster - INFO - To persist information across sessions, set the environment variable DAGSTER_HOME to a directory to use.
2025-03-23 14:36:16 -0500 - dagster - INFO - Launching Dagster services...

  Telemetry:

  As an open source project, we collect usage statistics to inform development priorities. For more
  information, read https://docs.dagster.io/getting-started/telemetry.

  We will not see or store solid definitions, pipeline definitions, modes, resources, context, or
  any data that is processed within solids and pipelines.

  To opt-out, add the following to $DAGSTER_HOME/dagster.yaml, creating that file if necessary:

    telemetry:
      enabled: false


  Welcome to Dagster!

  If you have any questions or would like to engage with the Dagster team, please join us on Slack
  (https://bit.ly/39dvSsF).

2025-03-23 14:38:21 -0500 - dagster.daemon - INFO - Instance is configured with the following daemons: ['AssetDaemon', 'BackfillDaemon', 'SchedulerDaemon',
'SensorDaemon']
2025-03-23 14:38:24 -0500 - dagster-webserver - INFO - Serving dagster-webserver on http://127.0.0.1:3000 in process 53686
```

I used "*dagster_dbt_parse_project_on_load=1 dagster dev*" command.
This command sets the DAGSTER_DBT_PARSE_PROJECT_ON_LOAD environment variable to 1 and starts the local Dagster development server with automatic dbt project parsing.

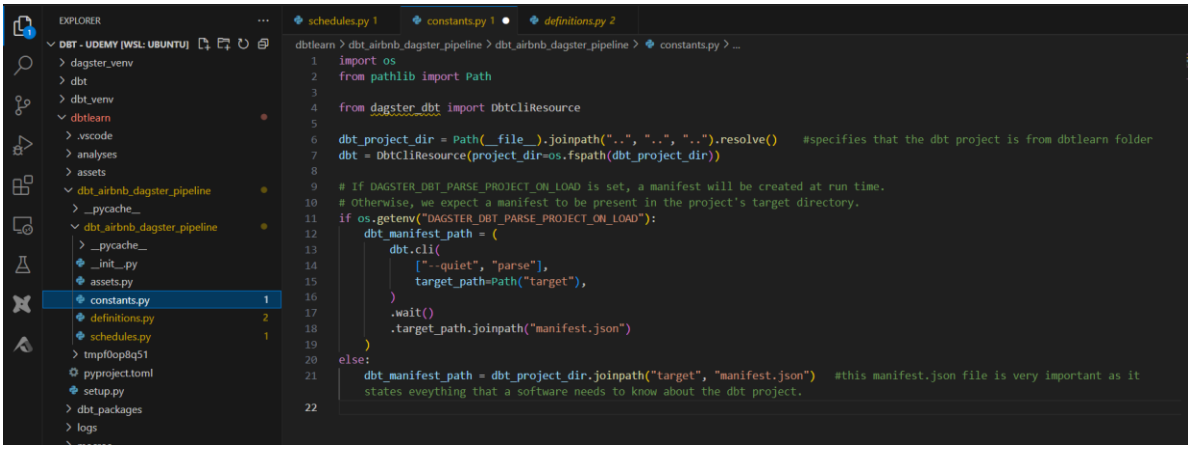



this is how the dagster UI looks.

Before we get started lets understand project structure of dagster –




1. constants.py




**File Purpose**

constants.py holds **shared configuration** and **paths** that tell Dagster how to locate and interact with your dbt project.

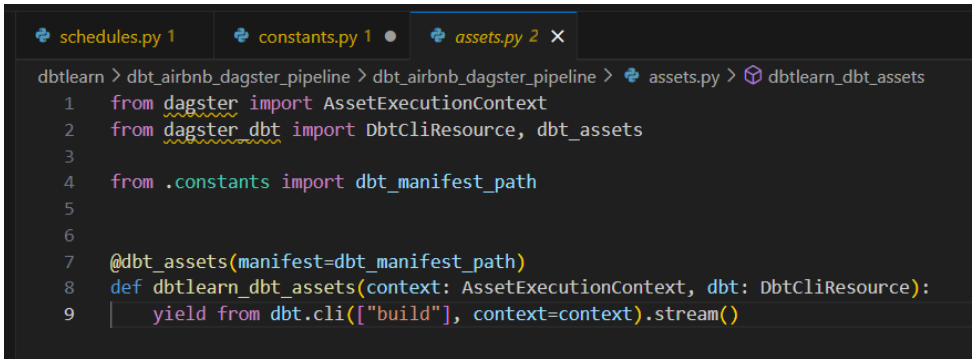
---

**Key Points**

- **dbt_project_dir**: Sets the path to your dbt project (so Dagster knows where your dbt_project.yml lives).
- **dbt (DbtCliResource)**: Connects Dagster to the dbt CLI, letting you run dbt commands from Dagster.
- **Environment Variable Check**: If DAGSTER_DBT_PARSE_PROJECT_ON_LOAD is set, dbt will parse the project on startup. Otherwise, the code points to an existing manifest.json file, which dbt uses to understand your project's structure.

Overall, this file makes sure other parts of your Dagster code know **where the dbt project is** and **how to run dbt** within your data pipelines.

2. assets.py




**File Purpose**

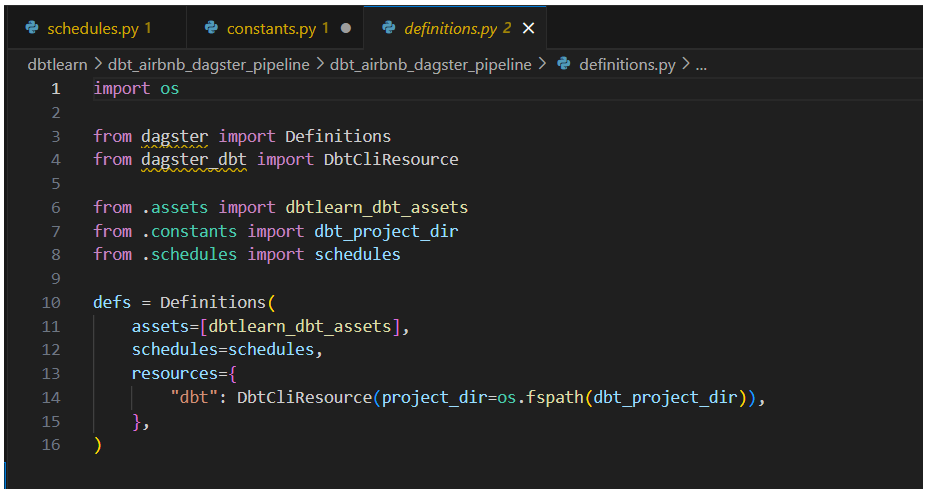
assets.py defines **Dagster assets** that trigger dbt commands. This lets you integrate dbt models as part of your Dagster pipeline.

---

**Key Points**

- **@dbt_assets(manifest=dbt_manifest_path)**: Tells Dagster that these assets come from the specified dbt manifest, so Dagster knows how to map dbt models to assets.
- **dbtlearn_dbt_assets**: This function runs dbt build using the DbtCliResource, allowing you to **build** (i.e., compile and execute) all dbt models.
- **Streaming the Output**: yield from dbt.cli("build", context=context).stream() streams the command's logs to Dagster, so you can see real-time progress.

3. definitions.py




**File Purpose**

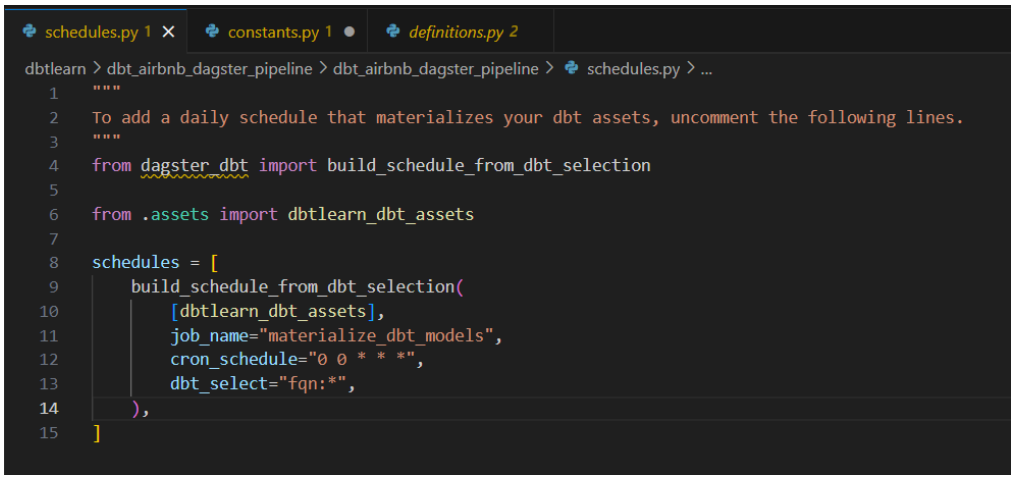
definitions.py serves as the **central registry** for your Dagster project. It ties together your assets, schedules, and resources so Dagster knows what to load and run.

**Key Points**

- **Definitions(...)**: Creates a Dagster definitions object that includes:
    - **Assets** (dbtlearn_dbt_assets): The dbt assets you defined in assets.py.
    - **Schedules**: A list of schedules to automate asset runs.
    - **Resources** (dbt): Configures the DbtCliResource with the path to your dbt project, enabling dbt commands within Dagster.

Everything is organized here so Dagster can discover and orchestrate your data pipeline.

4. schedules.py




**File Purpose**

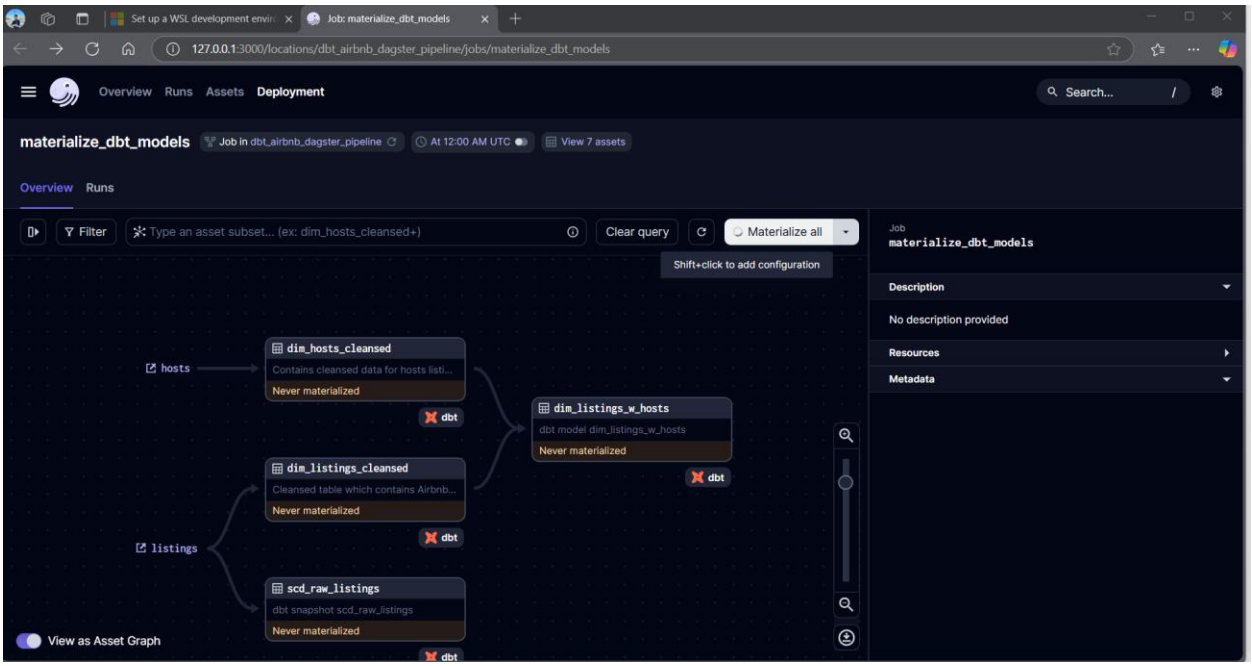
schedules.py defines how often and which dbt assets should be run automatically.

**Key Points**

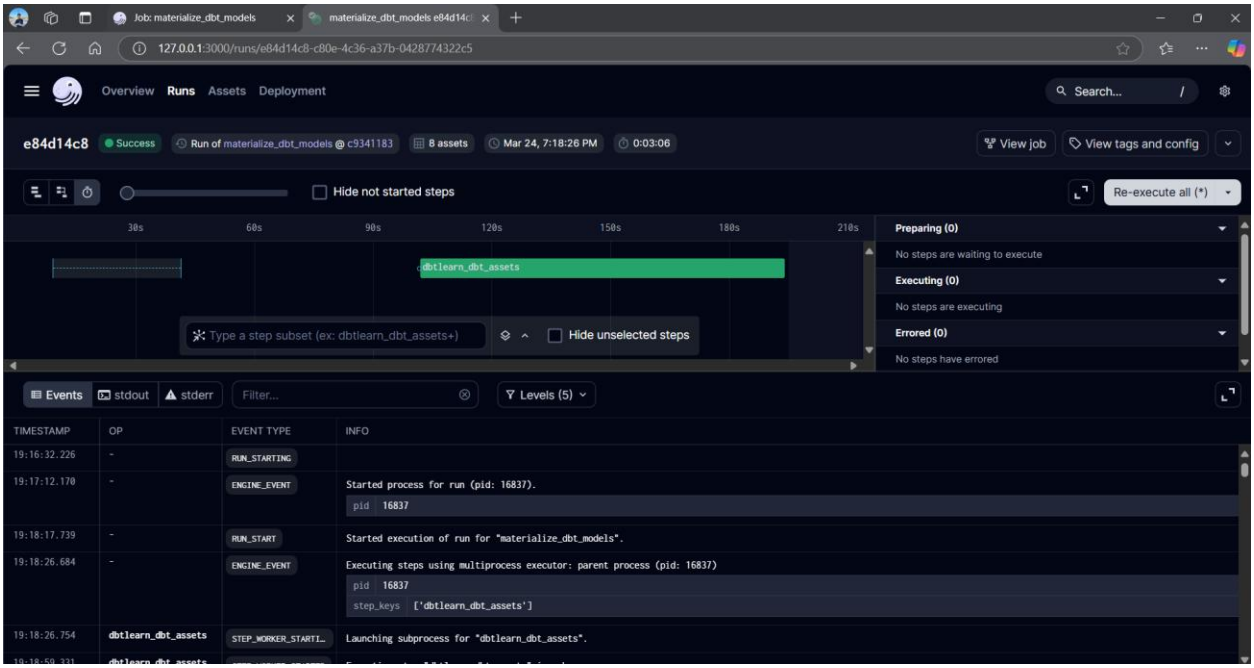
- **build_schedule_from_dbt_selection(...)**: Creates a Dagster schedule that runs your dbt models on a set cron schedule.
- **cron_schedule="0 0 * * *"**: Runs every day at midnight (example timing).
- **dbt_select="fqn:*"**: Tells the schedule to select all dbt assets (the "fully qualified name" selector).

When uncommented, this file automates your dbt pipeline execution on a fixed schedule.

Now, in the dagster UI you can use materialize all option to materialize everything ~ to ***dbt run*** command




this was done manually by clicking materialize option and not by scheduling option.




this shows that the materialization was successful.

**Dagster Run Window Overview**

This window displays the details of a specific pipeline/job execution—in this case, the **materialize_dbt_models** job.

**Top Section:**

- **Run ID (e84d14c8)**: Unique identifier for the run.
- **Status (Success)**: Indicates that the job executed without errors.

- **Assets (8)**: Total number of dbt models/assets involved in this run.
- **Run Time**: Shows start time and total duration (03:06).
- **Timeline Chart**: Visual execution timeline—dbtlearn_dbt_assets is the main step run here.

**Re-execution Options:**

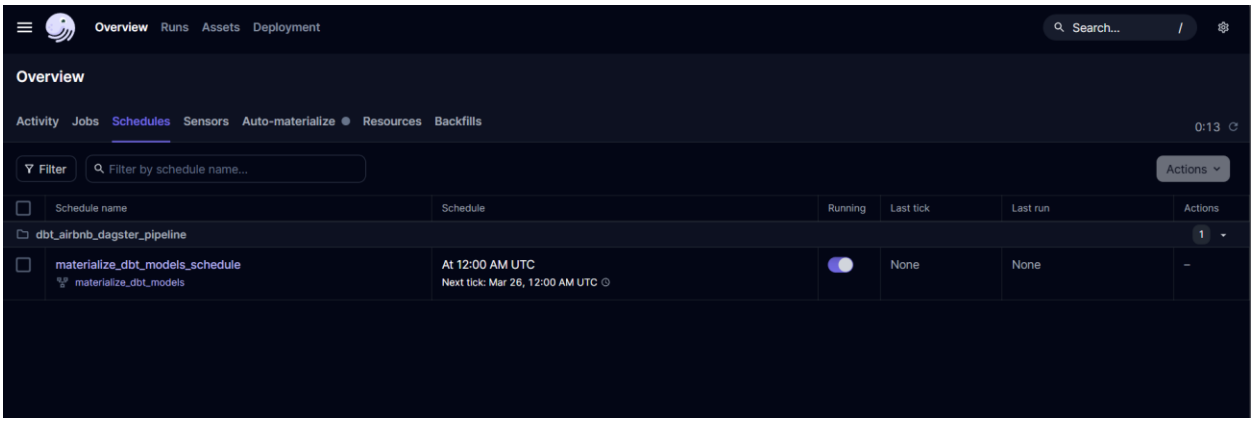
- **Re-execute all**: Allows re-running all steps from this job.
- **Step selection box**: Lets you filter and run only selected steps.

**Events Panel (Bottom):**

- **Timestamps**: Precise logs of execution events.
- **Event Types**:
    - RUN_START, RUN_STARTING: Marks the beginning of the run.
    - ENGINE_EVENT: Internal Dagster events (e.g., launching process).
    - STEP_WORKER_STARTING: Execution start for a step (e.g., dbt assets).
- **INFO column**: Human-readable logs, e.g., launching subprocess, step keys executed.

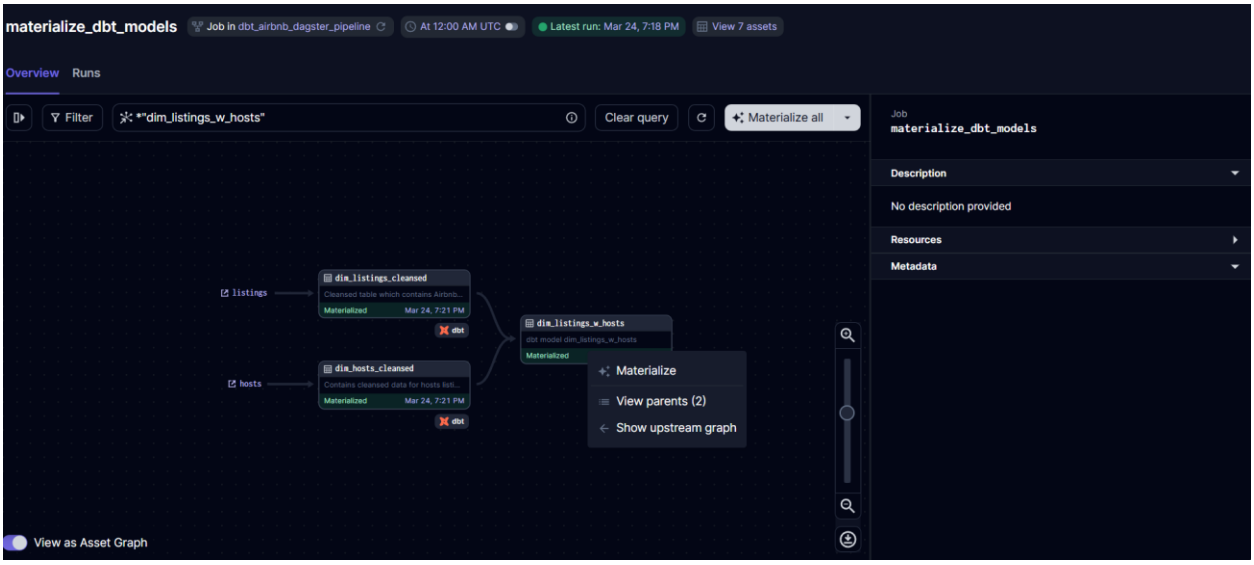
---

This interface provides **full visibility into the execution flow**, helpful for debugging and auditing the dbt + Dagster pipeline.



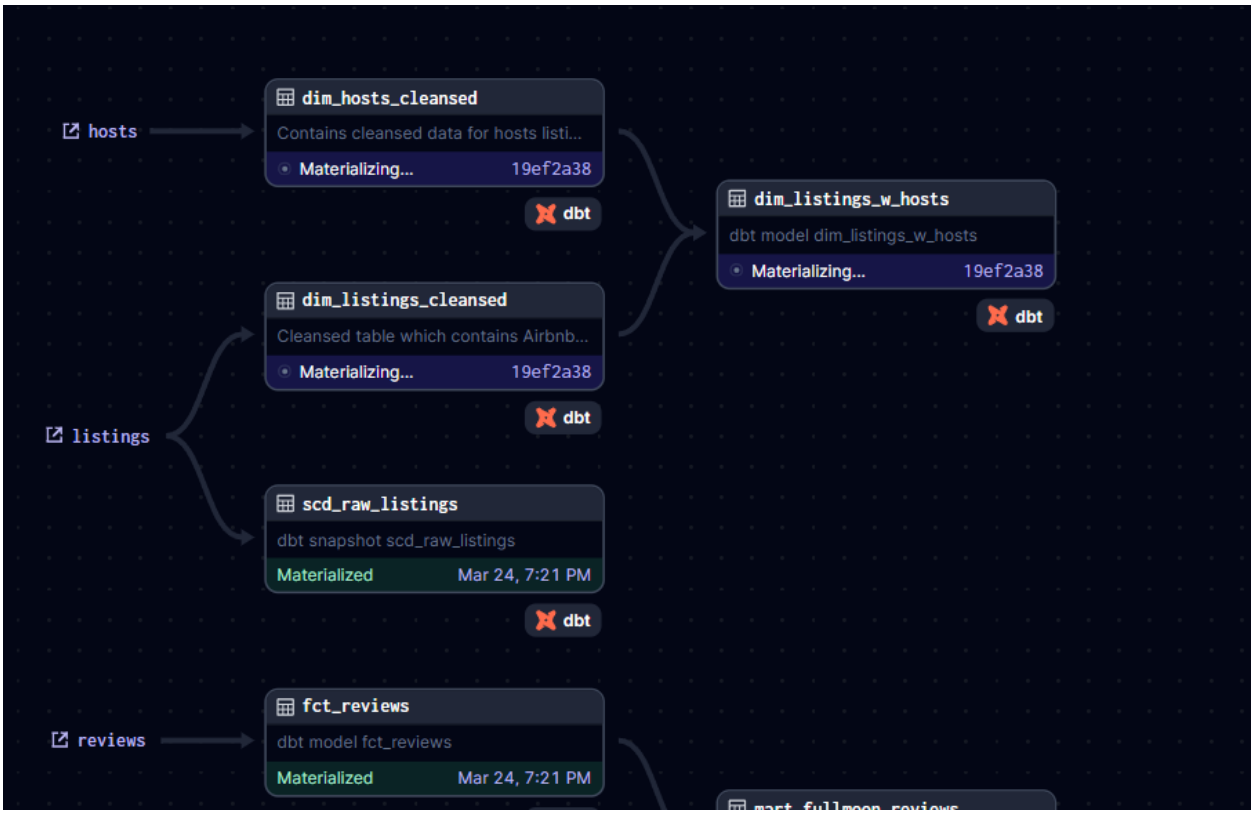

We also have a scheduled job set to run at midnight which will materialize the models as per the set schedule in schedules.py

Now,




here I have used 'show upstream graph' option to select all the models that contribute to dim_listings_w_hosts and since my scope of view is with respect to dim_listings_w_hosts then if I select materialize all option then everything will be rematerialized (the viewable models in the window).

I clicked on materialize all option –



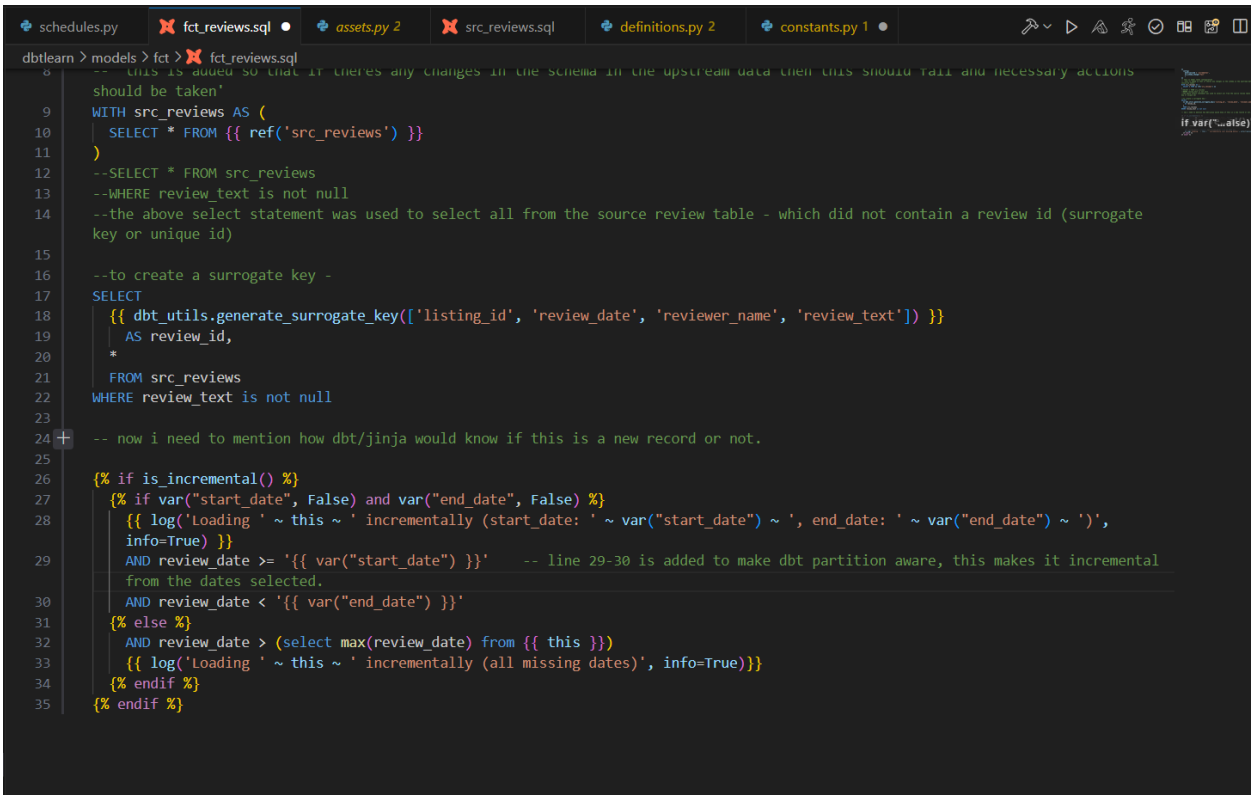

As you can see only the models that were in scope were being materialized.

- Using **dbt with Dagster** combines the strengths of both tools to build reliable, modular, and scalable data workflows. **dbt** focuses on data transformations, building and testing SQL-based

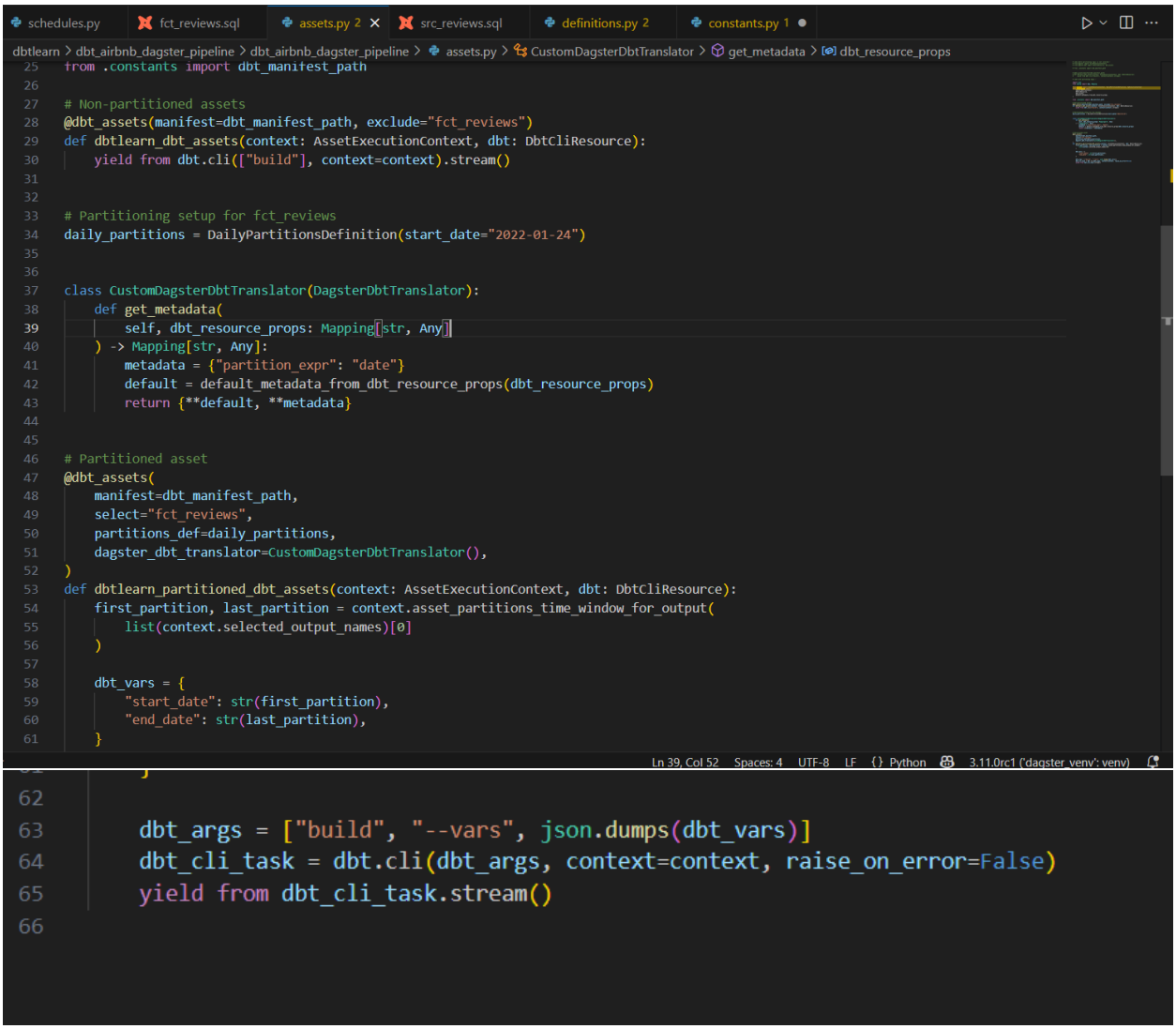
models in a data warehouse like Snowflake, while **Dagster** acts as an orchestration layer that manages the execution, scheduling, monitoring, and lineage tracking of these transformations.

- While dbt provides scheduling and testing features, Dagster enhances them with advanced capabilities like Python-based orchestration, retries, dependency management, and the ability to integrate dbt with other systems like APIs, ML pipelines, and data ingestion tools. Together, they enable full pipeline visibility, robust data asset tracking, and production-grade orchestration for modern data platforms.

Now, new need incremental models so that the as the load of data increases we do not have to process the entire model again and again. For eg- fct_reviews.sql is our incremental model with the logic -




This tells dbt to perform partition processing in fct_reviews with the mentioned logic.
**But for dagster to behave as incrementally we need to make changes in assets.py with the portioning logic –**
**Assets.py –**




```
    dbt_args = ["build", "--vars", json.dumps(dbt_vars)]
    dbt_cli_task = dbt.cli(dbt_args, context=context, raise_on_error=False)
    yield from dbt_cli_task.stream()
```

**Key Additions for Partitioning**

1. **DailyPartitionsDefinition**

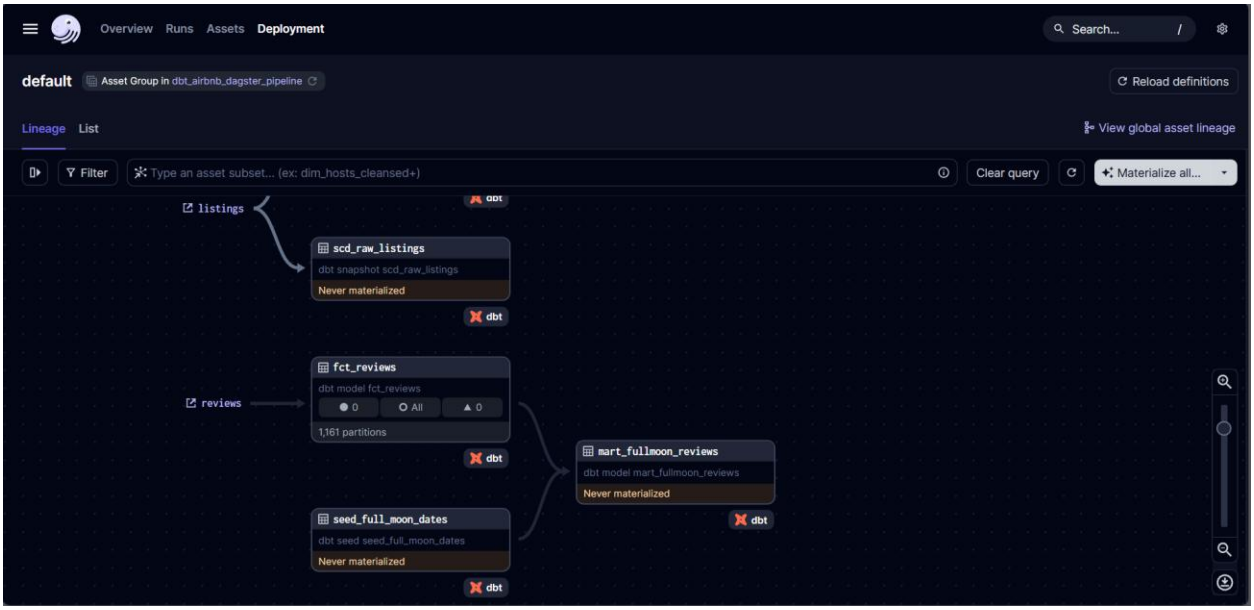
   *daily_partitions = DailyPartitionsDefinition(start_date="2022-01-24")*
   - Defines partitions by date, starting from the given date.
   - Each run will process data for a specific partition (e.g., one day).
2. **Custom Translator Class**

   *class CustomDagsterDbtTranslator(DagsterDbtTranslator):*
   *def get_metadata(...):*
   *metadata = {"partition_expr": "date"}*
   *...*
   - Tells Dagster how to link partitions to dbt, using the date column in your models.
   - This is essential for incremental models that depend on dates.
3. **Partitioned Asset Function**

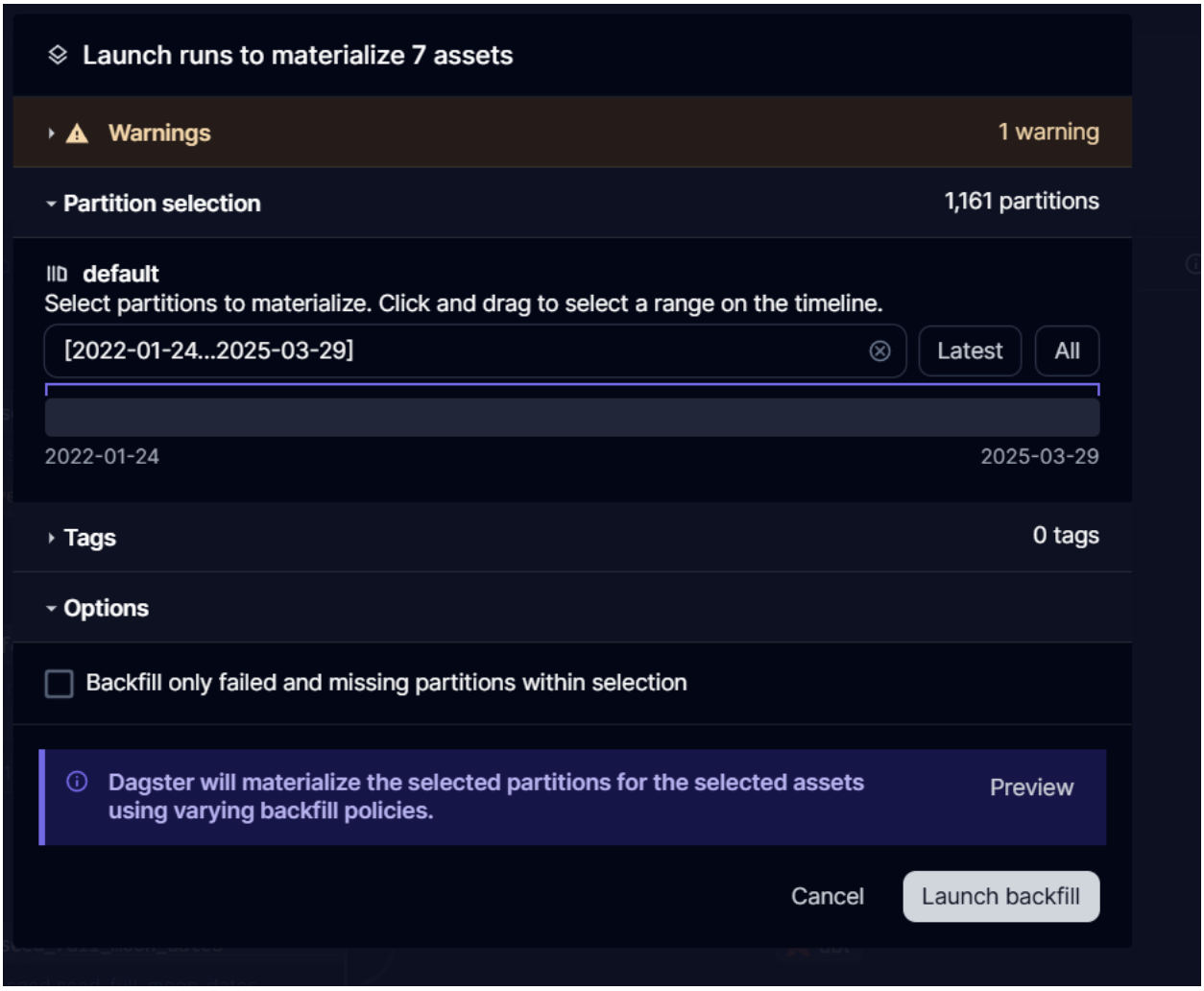
```
@dbt_assets(
  manifest=dbt_manifest_path,
  select="fct_reviews",
  partitions_def=daily_partitions,
  dagster_dbt_translator=CustomDagsterDbtTranslator()
)
def dbtlearn_partitioned_dbt_assets(...):
  ...
```

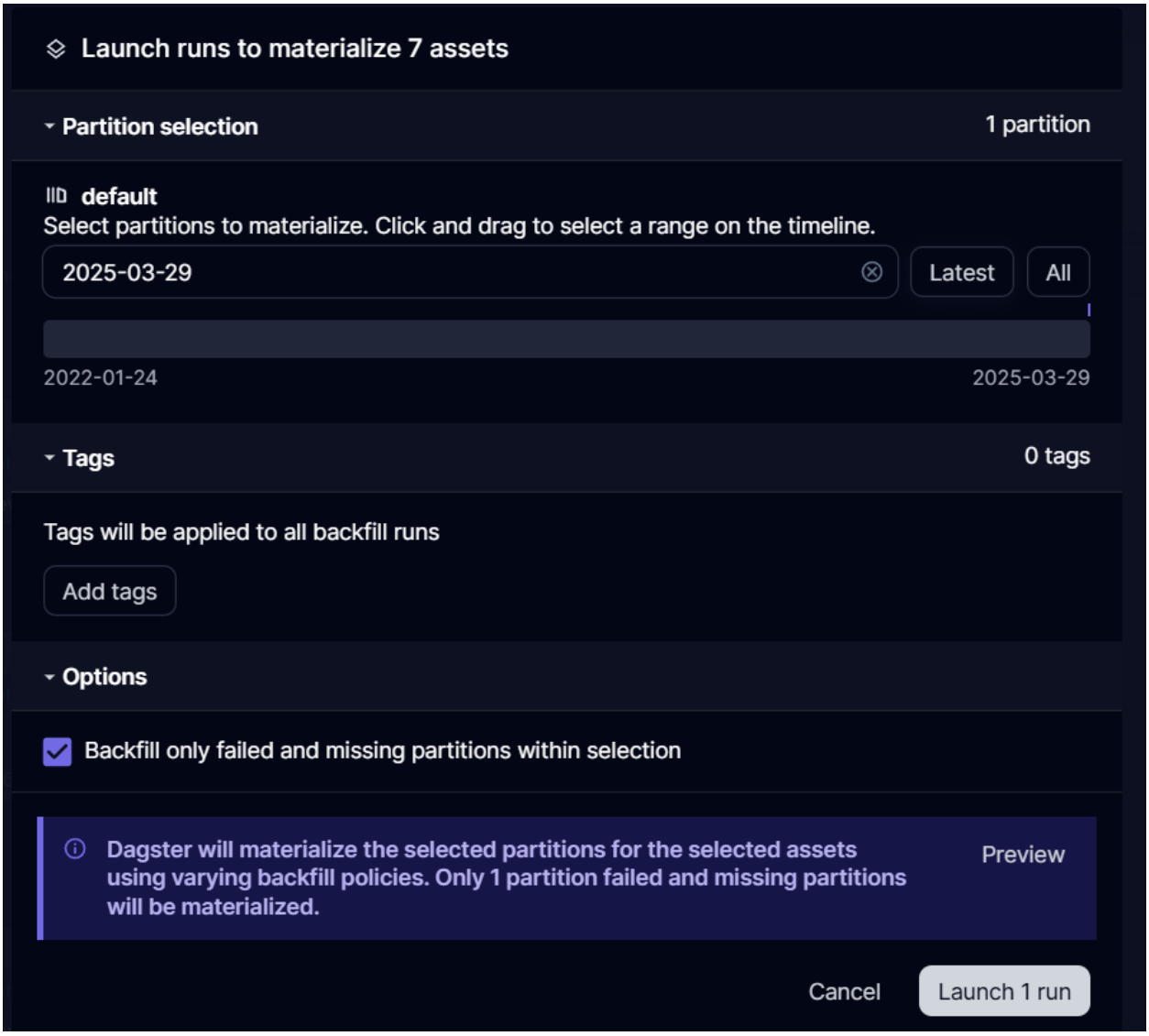
- This defines a partitioned dbt asset just for fct_reviews.
- It builds with --vars containing start_date and end_date, so dbt knows which data to process.

In Simple Terms: Only run fct_reviews for one day at a time, starting from Jan 24, 2022 — and use the date column to know which rows to process.

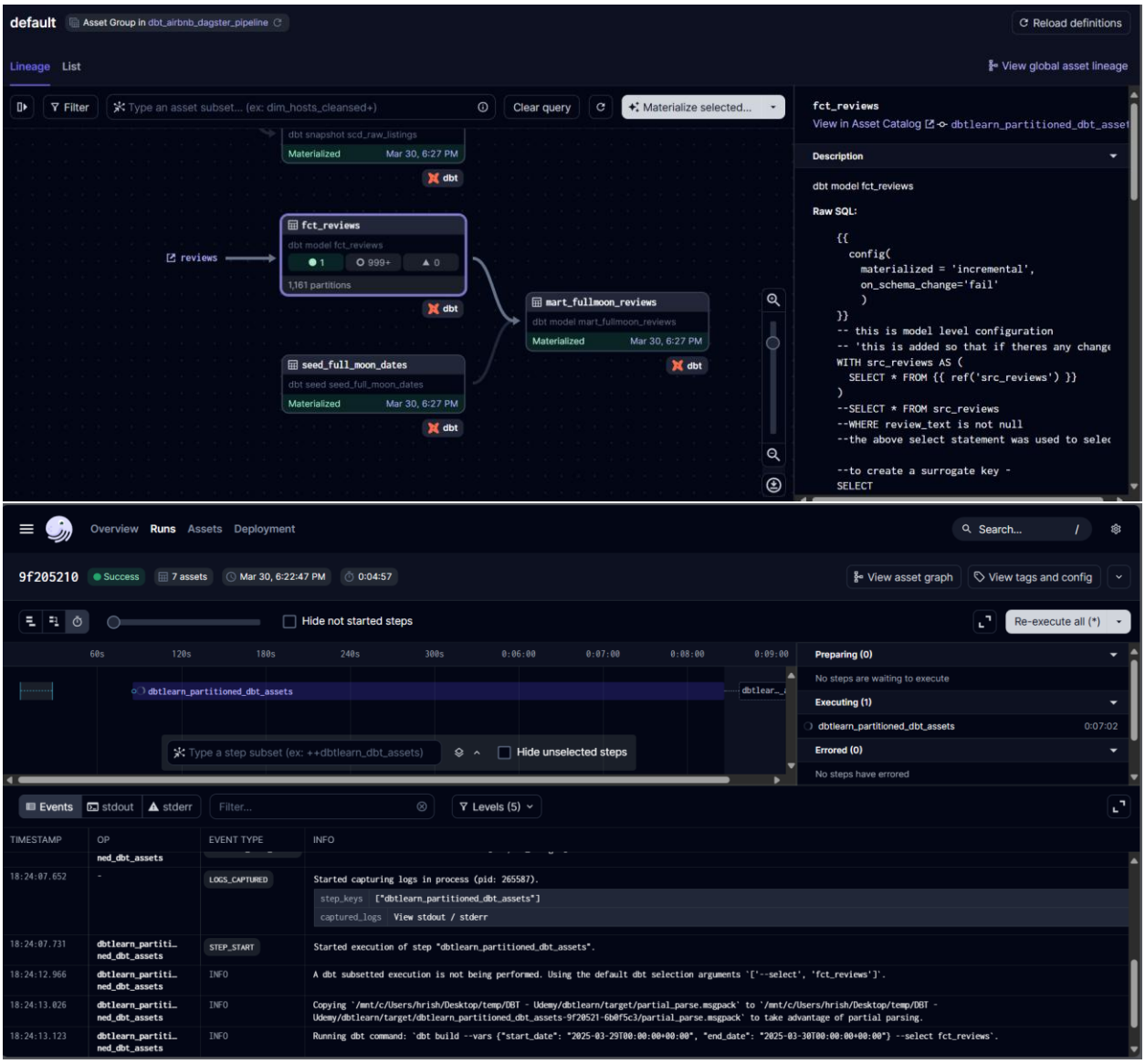



Can select the partitioning selection

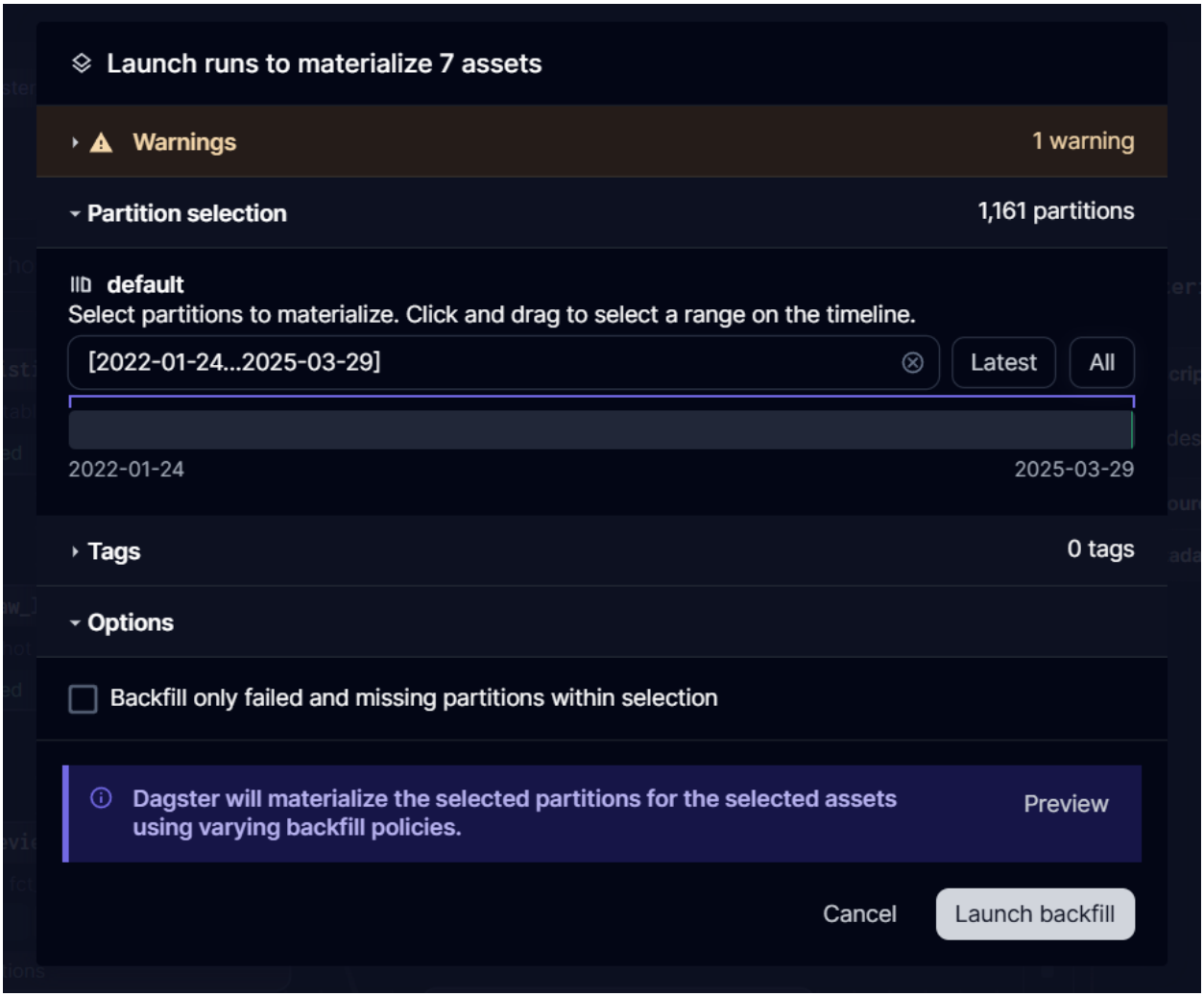

Launch runs to materialize 7 assets
Warnings
1 warning
Partition selection
1,161 partitions
default
Select partitions to materialize. Click and drag to select a range on the timeline.
[2022-01-24...2025-03-29]
Latest
All
2022-01-24
2025-03-29
Tags
0 tags
Options
Backfill only failed and missing partitions within selection
Dagster will materialize the selected partitions for the selected assets using varying backfill policies.
Preview
Cancel
Launch backfill



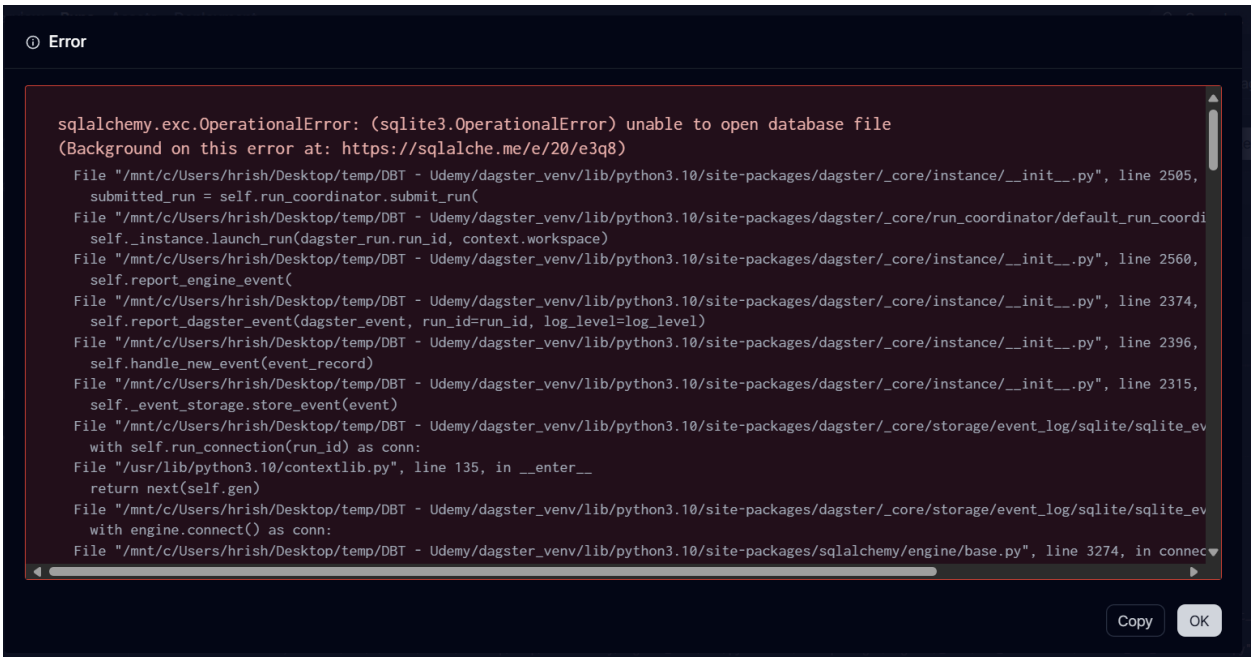

after materialization –




it got successfully partitioned based on the selected data and we have the dagster aware incremental model.
To compensate those missing materialization we can select materialize all so that it materializes all the partitions from 2022-01-24 i.e. backfill data-




Got this error –




This is due to the large load on sqllite as it was difficult to process 1161 partitions at a time. To solve this either change the sql engine to postgre, increase the timeout or tell dagster to process in batch instead of processing everything at one through cli command –

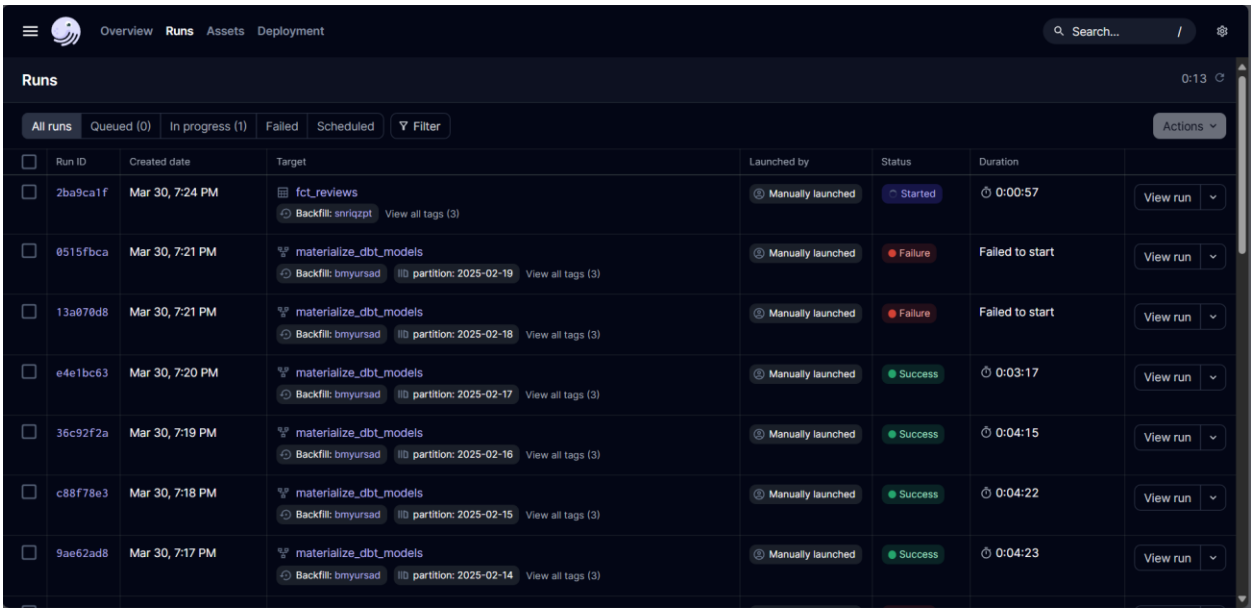
# In CLI:

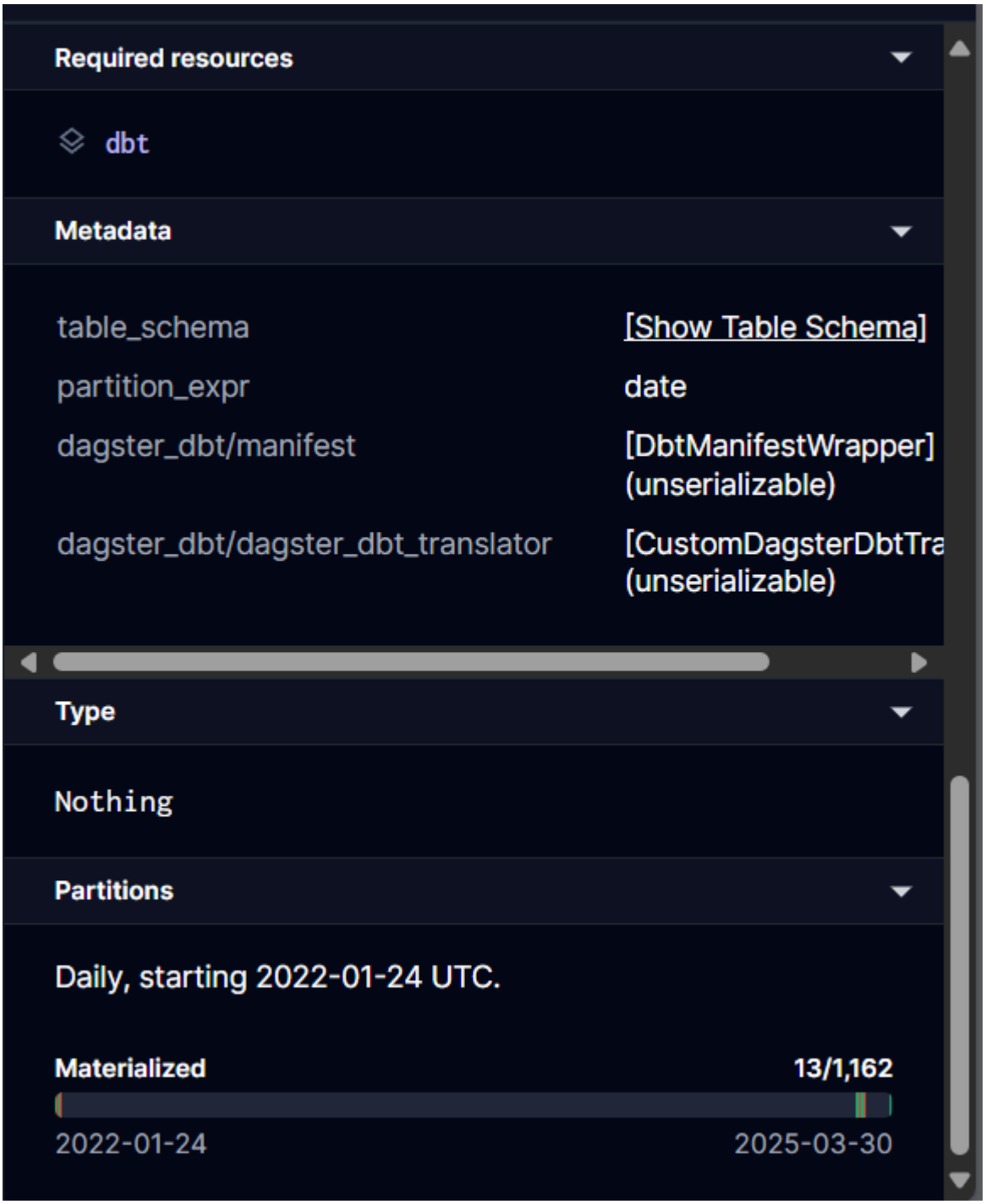
dagster job launch --partition-range start_date:end_date

or in UI we can select a smaller date range for now 7-30 days.

through UI –

For the 49 partitions selected dagster will create a separate run log for each partition running -





partition status can also be seen in the resource requirement section. (its still executing the runs)

This is the result after some time -


**Total 98 materialized partitions**. (because I clicked for 49 partitions and then the remaining ones again) with 8 errors.