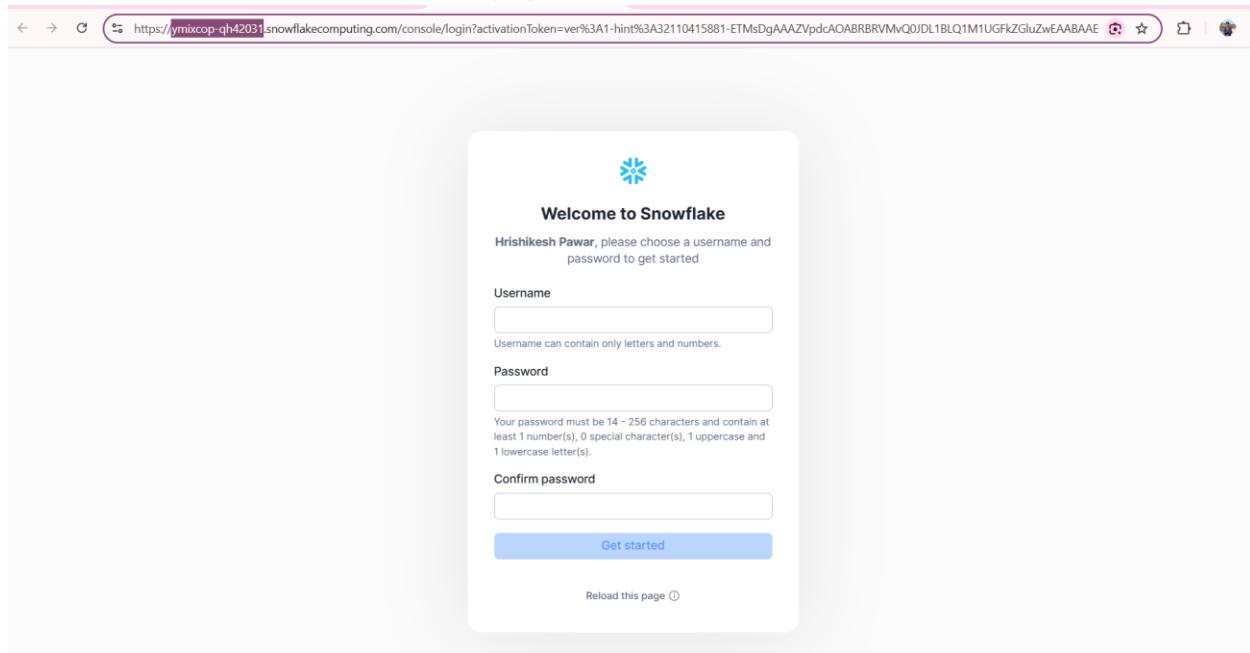


Activated snowflake account - ymixcop-qh42031



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 if 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 and Roles



✓ Importing Raw Tables



✓ Creating Reporter Role



Got Airbnb as database -

Databases

+ Database

3 Databases

Search Source All

NAME ↑	SOURCE	OWNER	CREATED	...
AIRBNB	Local	ACCOUNTADMIN	1 minute ago	...
SNOWFLAKE	Share	—	18 minutes ago	...
SNOWFLAKE_SA...	Share	ACCOUNTADMIN	18 minutes ago	...

The screenshot shows the Snowflake UI interface. At the top left is a search bar labeled "Search". To its right is a refresh icon. Below the search bar, the database tree is displayed under the "AIRBNB" schema. The "RAW" schema is currently selected, indicated by a blue background and a blue border around its name. Inside the "RAW" schema, the "Tables" node is also highlighted with a blue background and a blue border. Under "Tables", three tables are listed: "RAW_HOSTS", "RAW_LISTINGS", and "RAW_REVIEWS". Other nodes like "INFORMATION_SCHEMA" and "PUBLIC" are shown but are not selected. At the bottom of the tree, two additional schemas are visible: "SNOWFLAKE" and "SNOWFLAKE_SAMPLE_DATA".

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 -

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

```
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrishi/Desktop/temp/DBT - Udemy$ python --version
Python 3.11.0rc1
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrishi/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/hrishi/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 -

```
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
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): AIRBNB
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 validate the connection.
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrishi/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-5.15.15.1-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.

The screenshot shows the VS Code interface with the title bar "dbtlearn [WSL: Ubuntu]". The Explorer sidebar on the left shows a folder named "DBTLEARN (WSL: UBUNTU)" containing files like "analyses", "logs", "macros", "models", "seeds", "snapshots", ".gitignore", "dbt_project.yml", and "README.md". The main editor area displays the "dbt_project.yml" file with the following content:

```
profile: 'dbtlearn'

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

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

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

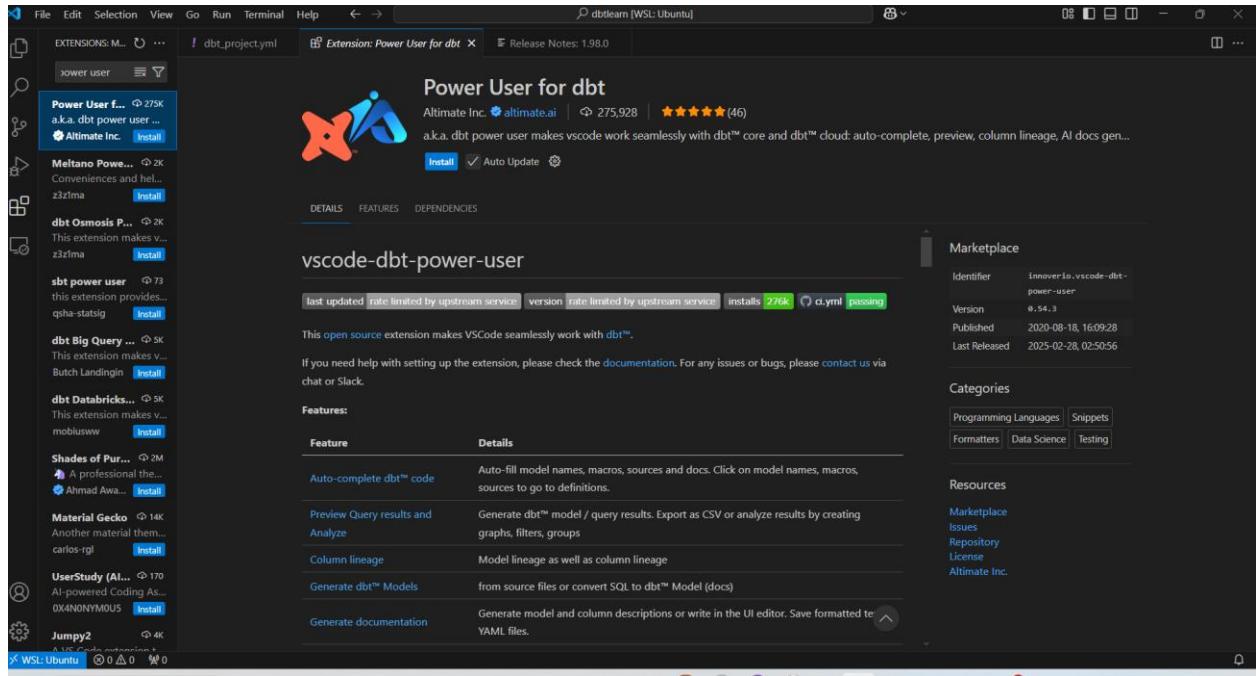
# In this example config, we tell dbt to build all models in the example/
# directory as views. These settings can be overridden in the individual model
# files using the `{{ config(...) }}` macro.

models:
  dbtlearn:
    # config indicated by + and applies to all files under models/example/
    example:
      +materialized: view
```

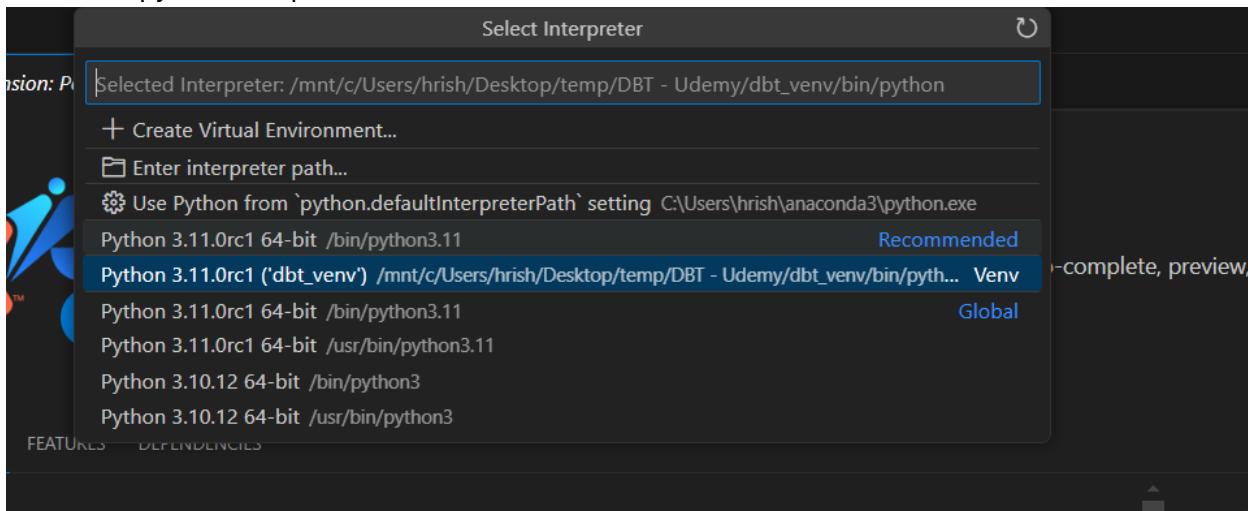
A status bar at the bottom right indicates "Source: WSL", "Ln 1, Col 1", "Spaces: 2", "UTF-8", "LF", "YAML", and a copy icon.

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 -

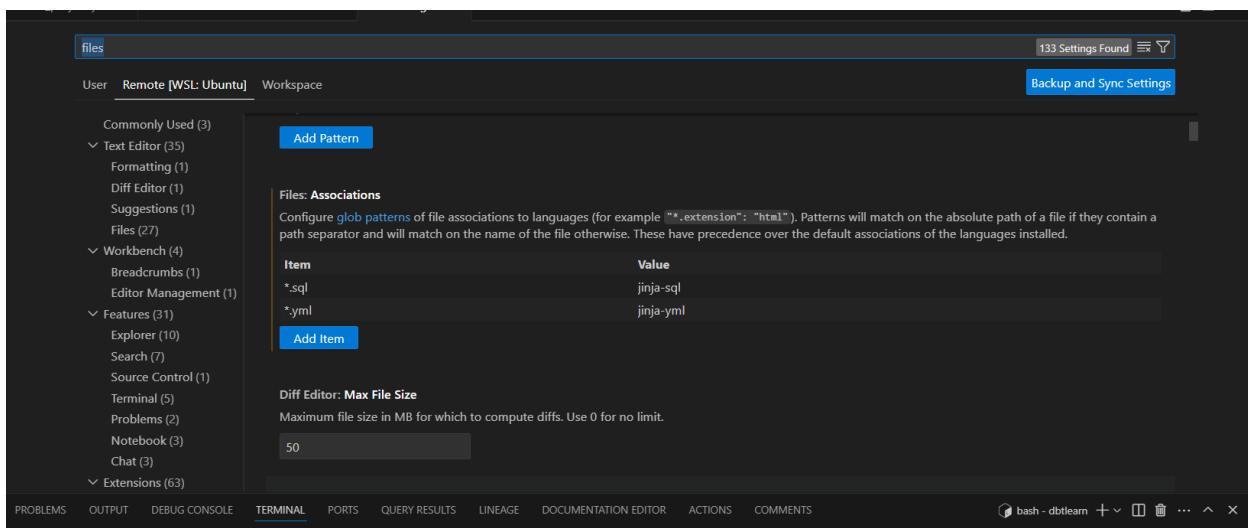


Or enter the file path manually (the venv is located in project folder)

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.

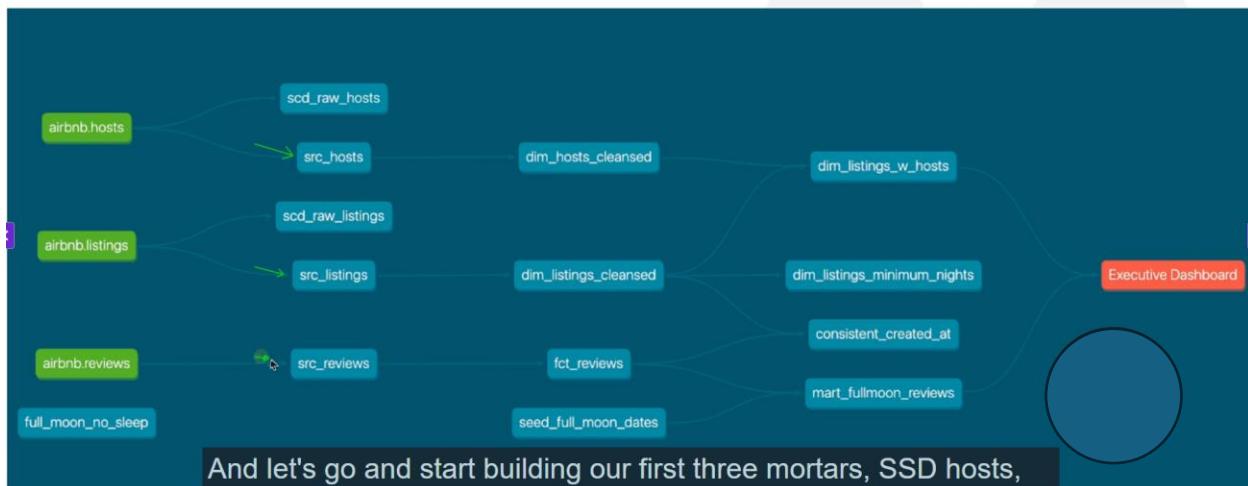
INPUT DATA MODEL

listings		reviews	
id	integer	listing_id	integer
listing_url	int	date	datetime
name	string	reviewer_name	string
room_type	string	comments	string
minimum_nights	integer	sentiment	string
host_id	integer		
price	string		
created_at	timestamp		
updated_at	timestamp		

hosts	
id	integer
name	integer
host_is_superhost	integer
created_at	timestamp
updated_at	timestamp

full_moon_dates	
full_moon_date	datetime

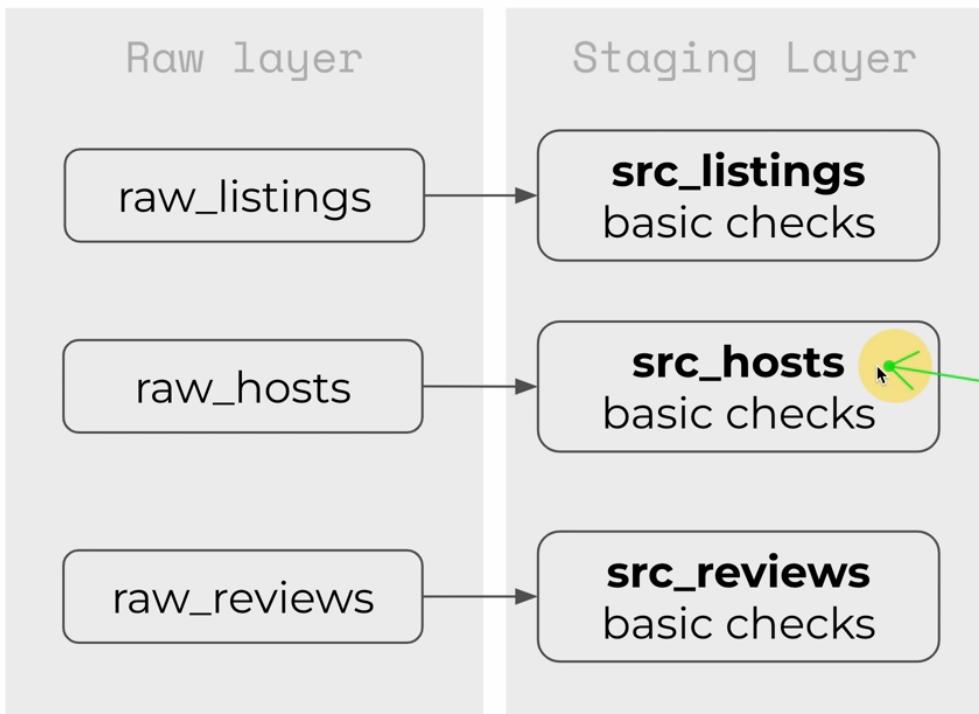
DATA FLOW OVERVIEW



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.

DATA FLOW PROGRESS



This is the raw_listings data that we have -

The screenshot shows a database interface with two main sections. The top section displays a table titled 'raw_listings' with 17.5K rows. The columns are: ID, LISTING_URL, NAME, ROOM_TYPE, and MINIMUM_NIGHTS. The bottom section shows the schema for the 'RAW_LISTINGS' table, which includes columns: ID, LISTING_URL, NAME, ROOM_TYPE, MINIMUM_NIGHTS, HOST_ID, PRICE, CREATED_AT, and UPDATED_AT.

	ID	LISTING_URL	NAME	ROOM_TYPE	MINIMUM_NIGHTS
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	Georgeous 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	...
14			
15	# ID	NUMBER(38,0)	
16	A LISTING_URL	VARCHAR(16777216)	
	A NAME	VARCHAR(16777216)	
	A ROOM_TYPE	VARCHAR(16777216)	
	# MINIMUM_NIGHTS	NUMBER(38,0)	
	# HOST_ID	NUMBER(38,0)	
	A PRICE	VARCHAR(16777216)	
	@ CREATED_AT	TIMESTAMP_NTZ(9)	
	@ UPDATED_AT	TIMESTAMP_NTZ(9)	

Writing cte -

The screenshot shows a database interface with a code editor. The code is a Common Table Expression (CTE) named 'raw_listings'. It starts with a 'WITH' clause selecting all columns from the 'RAW_LISTINGS' table in the 'AIRBNB.RAW' schema. The CTE is then used in a 'SELECT' statement to list the columns: listing_id, listing_name, listing_url, room_type, minimum_nights, host_id, price_str, create_at, and updated_at. The code is numbered from 1 to 16.

```

WITH raw_listings AS (
  SELECT * FROM AIRBNB.RAW.RAW_LISTINGS
)
SELECT
  id AS listing_id,
  name AS listing_name,
  listing_url,
  room_type,
  minimum_nights,
  host_id,
  price AS price_str,
  create_at,
  updated_at
FROM
  raw_listings

```

The screenshot shows a database query results interface. On the left is a table with columns: LISTING_ID, LISTING_NAME, LISTING_URL, ROOM_TYPE, and MINIMUM_NIGHTS. The table contains 11 rows of data. On the right, there are several charts: a bar chart for LISTING_ID showing values 3176 and 52900404; a histogram for LISTING_NAME indicating 100% filled.

	LISTING_ID	LISTING_NAME	LISTING_URL	ROOM_TYPE	MINIMUM_NIGHTS
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	Georgeous 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

The screenshot shows the dbt IDE interface. The left sidebar shows the project structure under 'EXPLORER'. The main area shows the contents of 'src_listings.sql' which contains a WITH clause and a SELECT statement.

```

models > src_listings.sql
  1 | WITH raw_listings AS (
  2 |   |   SELECT * FROM AIRBNB.RAW.RAW_LISTINGS
  3 | )
  4 | ✓ SELECT
  5 |   |   id AS listing_id,
  6 |   |   name AS listing_name,
  7 |   |   listing_url,
  8 |   |   room_type,
  9 |   |   minimum_nights,
 10 |   |   host_id,
 11 |   |   price AS price_str,
 12 |   |   created_at,
 13 |   |   updated_at
 14 |   |   FROM
 15 |   |   raw_listings
 16 |

```

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

Error faced -

```
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrishi/Desktop/temp/DBT - Udemy/dbtlearn$ dbt run
18:40:00  Running with dbt=1.9.3
18:40:04  Registered adapter: snowflake=1.9.0
18:40:20  Found 1 model, 472 macros
18:40:20
18:40:20  Concurrency: 1 threads (target='dev')
18:40:20
18:40:24
18:40:24  Finished running in 0 hours 0 minutes and 4.33 seconds (4.33s).
18:40:24  Encountered an error:
Runtime Error
Database error while listing schemas in database "AIRNBN"
Database Error
002043 (02000): SQL compilation error:
Object does not exist, or operation cannot be performed.
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrishi/Desktop/temp/DBT - Udemy/dbtlearn$ ls
Engineering Study Gui
Ask anything
```

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/hrishi/.dbt/profiles.yml

This file stores your connection details (Snowflake account, database, schema, role, etc.).

dbt_project.yml at /mnt/c/Users/hrishi/Desktop/temp/DBT - Udemy/dbtlearn/dbt_project.yml

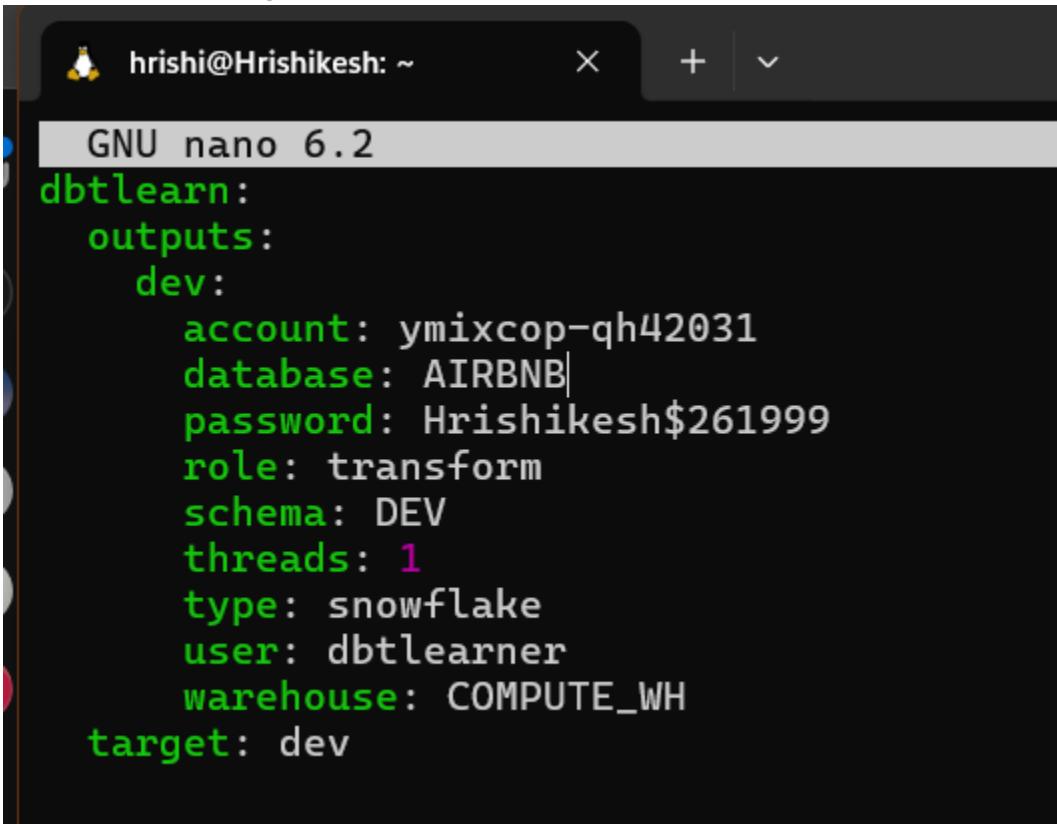
This file defines the name of your dbt project, the folder structure for models, the target schema name, and other project-specific settings.

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.

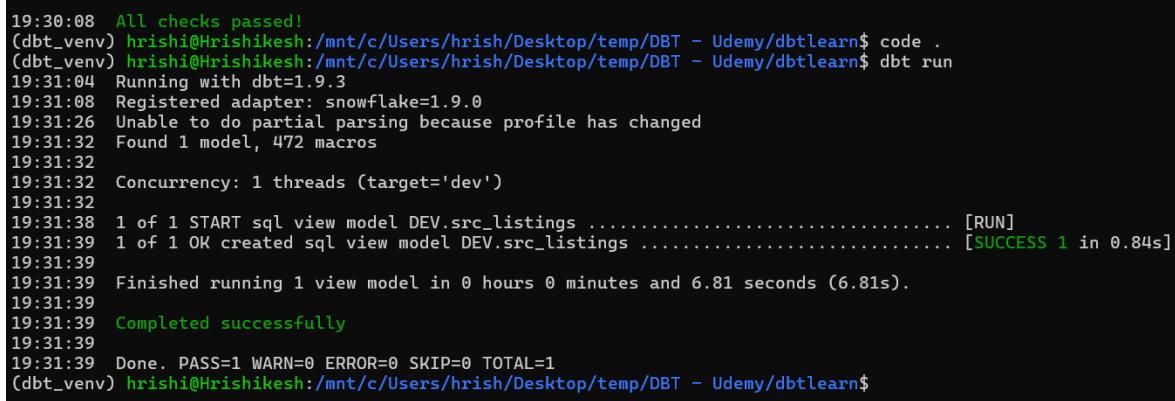
dbt_project.yml is a project-level config file, stored alongside your dbt project in your code folder.

Now, Made the changes -



The screenshot shows a terminal window titled "GNU nano 6.2". The content of the file is as follows:

```
hrishi@Hrishikesh: ~
GNU nano 6.2
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
```



The screenshot shows a terminal window displaying the output of a dbt run command. The log includes:

```
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 -

The screenshot shows the Snowflake interface. On the left, the sidebar includes options like Create, Home, Search, Projects, Data (selected), Databases, Data Products, AI & ML, Monitoring, Admin, and a trial credit reminder (\$399 credits left). The main area displays the schema structure for the AIRBNB database, specifically the DEV schema. Under Views, the SRC_LISTINGS view is selected and highlighted in blue. The view details page shows the following information:

AIRBNB / DEV / SRC_LISTINGS

View Details Columns Data Preview

9 Columns

NAME	TYPE	DESCRIPTION	NULLABLE	DEFAULT
CREATED_AT	Timestamp_N...		Yes	NULL
HOST_ID	# Number		Yes	NULL
LISTING_ID	# Number		Yes	NULL
LISTING_NAME	Varchar		Yes	NULL
LISTING_URL	Varchar		Yes	NULL
MINIMUM_NIGHTS	# Number		Yes	NULL
PRICE_STR	Varchar		Yes	NULL
ROOM_TYPE	Varchar		Yes	NULL
UPDATED_AT	Timestamp_N...		Yes	NULL

Applied the same step to create a raw_reviews

The screenshot shows the dbt CLI interface in a terminal window titled "dbtlearn [WSL: Ubuntu]". The left sidebar shows project structure under "DBTLEARN [WSL: UBUNTU]". The right pane shows the SQL code for the "src_reviews.sql" file:

```
models > src > src_reviews.sql
          Add documentation or tests | Datapilot Notebooks
1   WITH raw_reviews AS (
2     SELECT *
3     FROM
4       AIRBNB.RAW.RAW_REVIEWS
5   )
6   SELECT
7     listing_id,
8     date AS review_date,
9     reviewer_name,
10    comments AS review_text,
11    sentiment AS review_sentiment
12   FROM
13     raw_reviews
14
```

Dbt run -

```

19:51:59 Done. PASS=1 WARN=0 ERROR=0 SKIP=0 TOTAL=1
(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 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-

The screenshot shows the Snowflake UI interface. On the left, there is a sidebar with a search bar and a tree view of the database schema:

- AIRBNB** (selected)
 - DEV**
 - Views**
 - SRC_LISTINGS
 - SRC_REVIEWS** (highlighted in blue)
 - INFORMATION_SCHEMA
 - PUBLIC
 - RAW**
 - Tables**
 - RAW_HOSTS
 - RAW_LISTINGS
 - RAW_REVIEWS
 - SNOWFLAKE**
 - SNOWFLAKE_SAMPLE_DATA**

On the right, the details for the **AIRBNB / DEV / SRC_REVIEWS** table are displayed:

View Details | **Columns** (selected) | **Data Preview**

5 Columns

NAME ↑	TYPE	DESCRIPTION ⓘ	NULLABLE	DEFAULT
LISTING_ID	# Number		Yes	NULL
REVIEWER_NAME	VarChar		Yes	NULL
REVIEW_DATE	Timestamp_N...		Yes	NULL
REVIEW_SENTIMENT	VarChar		Yes	NULL
REVIEW_TEXT	VarChar		Yes	NULL

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 below screenshot -

The screenshot shows the dbt core interface running on WSL: Ubuntu. The Explorer sidebar on the left lists files like dbt_project.yml, README.md, and various source and model files. The main area displays a preview of 500 rows from the test_models.sql file. The results table has columns: LISTING_ID, LISTING_NAME, LISTING_URL, ROOM_TYPE, MINIMUM_NIGHTS, HOST_ID, PRICE_STR, CREATED_AT, and UPDATED_AT. The data includes various Airbnb listings with their details and pricing information.

LISTING_ID	LISTING_NAME	LISTING_URL	ROOM_TYPE	MINIMUM_NIGHTS	HOST_ID	PRICE_STR	CREATED_AT	UPDATED_AT
3176	Fabulous Flat in great Locat...	https://www.airbnb.com/roo...	Entire home/apt	62	3718	\$90.00	2009-06-05T21:34:42	2009-06-05T21:34:42
7071	BrightRoom with sunny gree...	https://www.airbnb.com/roo...	Private room	1	17391	\$33.00	2009-08-12T12:30:30	2009-08-12T12:30:30
9991	Georgeous flat - outstandi...	https://www.airbnb.com/roo...	Entire home/apt	0	33852	\$180.00	2015-07-30T05:08:52	2015-07-30T05:08:52
14325	Apartment in Prenzlauer Berg	https://www.airbnb.com/roo...	Entire home/apt	95	55531	\$70.00	2010-06-15T19:56:01	2010-06-15T19:56:01
16644	In the Heart of Berlin - Kreuz...	https://www.airbnb.com/roo...	Entire home/apt	60	64696	\$90.00	2010-05-30T12:11:33	2010-05-30T12:11:33
17904	Beautiful Kreuzberg studi...	https://www.airbnb.com/roo...	Entire home/apt	92	68997	\$47.00	2010-02-08T17:23:48	2010-02-08T17:23:48
20858	Designer Loft in Berlin Mitte	https://www.airbnb.com/roo...	Entire home/apt	3	71331	\$169.00	2012-09-24T21:03:01	2012-09-24T21:03:01
21869	Studio in the Heart of Kreuz...	https://www.airbnb.com/roo...	Entire home/apt	60	64696	\$70.00	2010-09-08T11:45:56	2010-09-08T11:45:56
22438	WOHNUNG IN BERLIN ★ MITTE	https://www.airbnb.com/roo...	Entire home/apt	90	86159	\$65.00	2011-03-31T17:17:58	2011-03-31T17:17:58
22677	Prenzel garden with leafy ter...	https://www.airbnb.com/roo...	Entire home/apt	2	87357	\$120.00	2010-10-08T08:59:12	2010-10-08T08:59:12
23834	Apartment in the heart of Be...	https://www.airbnb.com/roo...	Entire home/apt	185	94918	\$65.00	2010-05-26T11:58:00	2010-05-26T11:58:00
24569	Sunny & Wheelchair access...	https://www.airbnb.com/roo...	Entire home/apt	5	99662	\$200.00	2010-09-18T04:06:39	2010-09-18T04:06:39
26543	Helmholtzplatz Bright&Spac...	https://www.airbnb.com/roo...	Entire home/apt	60	112675	\$208.00	2010-08-03T02:49:12	2010-08-03T02:49:12
28156	Beautiful apartment in Prenz...	https://www.airbnb.com/roo...	Entire home/apt	95	55531	\$80.00	2010-11-09T05:01:07	2010-11-09T05:01:07

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

The screenshot shows the dbt core interface running on WSL: Ubuntu. The Explorer sidebar on the left lists files like dbt_project.yml, README.md, and various source and model files. The main area displays a preview of 500 rows from the test_models.sql file. The results table has columns: LISTING_ID, LISTING_NAME, LISTING_URL, ROOM_TYPE, MINIMUM_NIGHTS, HOST_ID, PRICE_STR, CREATED_AT, and UPDATED_AT. A PerspectiveDatagridJSONV view is open on the right, showing a sidebar with data manipulation options: Group By, Split By, Order By, Where, and Columns. The columns listed are # LISTING_ID, # LISTING_NAME, and # LISTING_URL.

LISTING_ID	LISTING_NAME	LISTING_URL	ROOM_TYPE	MINIMUM_NIGHTS	HOST_ID	PRICE_STR	CREATED_AT	UPDATED_AT
3176	Fabulous Flat in great Locat...	https://www.airbnb.com/roo...	Entire home/apt	62	3718	\$90.00	2009-06-05T21:34:42	2009-06-05T21:34:42
7071	BrightRoom with sunny gree...	https://www.airbnb.com/roo...	Private room	1	17391	\$33.00	2009-08-12T12:30:30	2009-08-12T12:30:30
9991	Georgeous flat - outstandi...	https://www.airbnb.com/roo...	Entire home/apt	0	33852	\$180.00	2015-07-30T05:08:52	2015-07-30T05:08:52
14325	Apartment in Prenzlauer Berg	https://www.airbnb.com/roo...	Entire home/apt	95	55531	\$70.00	2010-06-15T19:56:01	2010-06-15T19:56:01
16644	In the Heart of Berlin - Kreuz...	https://www.airbnb.com/roo...	Entire home/apt	60	64696	\$90.00	2010-05-30T12:11:33	2010-05-30T12:11:33
17904	Beautiful Kreuzberg studi...	https://www.airbnb.com/roo...	Entire home/apt	92	68997	\$47.00	2010-02-08T17:23:48	2010-02-08T17:23:48
20858	Designer Loft in Berlin Mitte	https://www.airbnb.com/roo...	Entire home/apt	3	71331	\$169.00	2012-09-24T21:03:01	2012-09-24T21:03:01
21869	Studio in the Heart of Kreuz...	https://www.airbnb.com/roo...	Entire home/apt	60	64696	\$70.00	2010-09-08T11:45:56	2010-09-08T11:45:56
22438	WOHNUNG IN BERLIN ★ MITTE	https://www.airbnb.com/roo...	Entire home/apt	90	86159	\$65.00	2011-03-31T17:17:58	2011-03-31T17:17:58
22677	Prenzel garden with leafy ter...	https://www.airbnb.com/roo...	Entire home/apt	2	87357	\$120.00	2010-10-08T08:59:12	2010-10-08T08:59:12
23834	Apartment in the heart of Be...	https://www.airbnb.com/roo...	Entire home/apt	185	94918	\$65.00	2010-05-26T11:58:00	2010-05-26T11:58:00
24569	Sunny & Wheelchair access...	https://www.airbnb.com/roo...	Entire home/apt	5	99662	\$200.00	2010-09-18T04:06:39	2010-09-18T04:06:39
26543	Helmholtzplatz Bright&Spac...	https://www.airbnb.com/roo...	Entire home/apt	60	112675	\$208.00	2010-08-03T02:49:12	2010-08-03T02:49:12

Creating another view called – src_hosts.sql –

Screenshot of a terminal window showing a DBT project structure and a SQL script editor.

File Explorer:

- DBLEARN [WSL: UBUNTU]
 - .vscode
 - analyses
 - dbt_packages
 - logs
 - macros
 - models
 - src
 - src_listings.sql
 - src_reviews.sql
 - test_models.sql
 - src_hosts.sql
 - seeds
 - snapshots
 - target
 - tests
 - .gitignore
 - dbt_project.yml
 - README.md

SQL Editor:

```

models > src_hosts.sql
      Add documentation or tests | Datapilot Notebooks
      WITH raw_hosts AS (
        SELECT *
        FROM AIRBNB.RAW.RAW_HOSTS
      )
      SELECT
        id AS host_id,
        NAME AS host_name,
        is_superhost,
        created_at,
        updated_at
      FROM raw_hosts
  
```

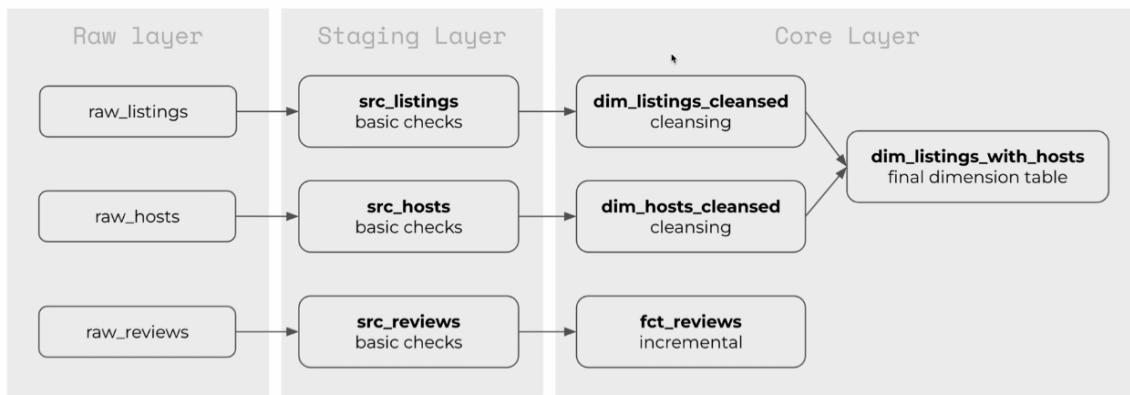
```

(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

DATA FLOW PROGRESS



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.

```
File Edit Selection View Go Run Terminal Help < > dbtlearn [WSL: Ubuntu] ⌂
```

EXPLORER	...	dbt_project.yml	src_listings.sql	src_reviews.sql	test_models.sql	dim_listings_cleansed.sql	src_hosts.sql
DBTLEARN [WSL: UBUNTU]		models > dim > dim_listings_cleansed.sql	Add documentation or tests Datapilot Notebooks				
> .vscode		WITH src_listings AS (
> analyses		SELECT					
> dbt_packages		*					
> logs		FROM					
> macros		{{ ref('src_listings') }}					
models)					
dim		SELECT					
dim_listings_cleansed.sql		listing_id,					
src		listing_name,					
src_hosts.sql		room_type,					
src_listings.sql		CASE					
src_reviews.sql		WHEN minimum_nights = 0 THEN 1					
test_models.sql		ELSE minimum_nights					
seeds		END AS minimum_nights,					
snapshots		host_id,					
target		REPLACE(
tests		price_str,					
.gitignore		'\$'					
dbt_project.yml) :: NUMBER(
README.md		10,					
		2,					
) AS price,					
		created_at,					
		updated_at					
		FROM					
		src_listings					

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.

```
01:27:10 Done. PASS=4 WARN=0 ERROR=0 SKIP=0 TOTAL=4
(dbt_venv) hrishi@HrishiKesh:/mnt/c/Users/hrishi/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 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/hrishi/Desktop/temp/DBT - Udemy/dbtlearn$
```

Verifying in snowflake –

The screenshot shows the Snowflake interface. On the left, the sidebar includes options like Home, Search, Projects, Data (selected), Databases, Data Products, AI & ML, Monitoring, and Admin. It also displays \$399 credits left and a Trial ends in 27 days message. The main panel shows the schema structure for the AIRBNB database, specifically the DEV schema. Under RAW, there is a table named DIM_LISTINGS_CLEANSED which contains 8 columns: CREATED_AT, HOST_ID, LISTING_ID, LISTING_NAME, MINIMUM_NIGHTS, PRICE, ROOM_TYPE, and UPDATED_AT. Each column has its type (e.g., Timestamp_NTZ, Number, Varchar), description, and nullable status.

NAME	TYPE	DESCRIPTION	NULLABLE	DEFAULT
CREATED_AT	Timestamp_NTZ		Yes	NULL
HOST_ID	Number		Yes	NULL
LISTING_ID	Number		Yes	NULL
LISTING_NAME	Varchar		Yes	NULL
MINIMUM_NIGHTS	Number		Yes	NULL
PRICE	Number		Yes	NULL
ROOM_TYPE	Varchar		Yes	NULL
UPDATED_AT	Timestamp_NTZ		Yes	NULL

Dim_hosts_cleaned.sql

The screenshot shows the DBT IDE interface with multiple tabs open: dbt_project.yml, src_listings.sql, src_reviews.sql, dim_listings_cleaned.sql, dim_hosts_cleaned.sql (active tab), and src_hosts.sql. The Explorer sidebar shows the project structure under DBTLEARN [WSL: UBUNTU]. The active tab, dim_hosts_cleaned.sql, contains the following SQL code:

```

{{ config(
    materialized = 'view'
) }

WITH src_hosts AS (
    SELECT
        *
    FROM
        {{ ref('src_hosts') }}
)
SELECT
    host_id,
    NVL(
        host_name,
        'Anonymous'
    ) AS host_name,
    is_superhost,
    created_at,
    updated_at
FROM
    src_hosts

```

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

The screenshot shows the VS Code interface with the following details:

- EXPLORER**: Shows the project structure under "DBTLEARN [WSL: UBUNTU]". It includes:
 - `.vscode`
 - analyses
 - dbt_packages
 - logs
 - macros
 - models
 - dim
 - dim_hosts_cleanse...
 - dim_listings_clean...
 - src
 - src_hosts.sql
 - src_listings.sql
 - src_reviews.sql
 - seeds
 - snapshots
 - target
 - tests
 - .gitignore
- dbt_project.yml**: The active file in the editor. Its content is as follows:


```

version: 2
snapshot_paths: [ snapshots ]
clean_targets:
  - "target"
  - "dbt_packages"

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

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

models:
  dbtlearn:
    # Config indicated by + and applies to all files under models/example/
    +materialized: view # here hashtag means i do not want a subfolder named materialized and sets this materialized to default as 'view'
    dim:
      +materialized: table
      
```
- README.md**: A file listed in the Explorer.

```
(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_cleanse ..... [RUN]
03:33:44 4 of 5 OK created sql view model DEV.dim_hosts_cleanse ..... [SUCCESS 1 in 0.37s]
03:33:44 5 of 5 START sql table model DEV.dim_listings_cleanse ..... [RUN]
03:33:45 5 of 5 OK created sql table model DEV.dim_listings_cleanse ..... [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 –

The screenshot shows a code editor with the following details:

- models > dim > dim_hosts_cleanse.sql**: The active file in the editor. It contains the following configuration:


```

{{ config(
  materialized = 'view'
)}}
      
```
- Add documentation or tests | Datapilot Notebooks**: A tooltip or sidebar indicating options for the current file.

Here earlier I set the type of materialization explicitly for dim_hosts_cleanse.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.

```

EXPLORER [DBTLEARN [WSL: UBUNTU]] dbt_project.yml dim_hosts_cleanse... src_reviews.sql fct_reviews.sql
models > fct > fct_reviews.sql
1  {{ config(
2    materialized = 'incremental',
3    on_schema_change='fail'
4  )}}
5  }}
6  -- this is model level configuration
7  -- 'this is added so that if theres any changes in the schema in the upstream data then this should fail and necessary actions should be taken'
8  WITH src_reviews AS (
9    SELECT * FROM {{ ref('src_reviews') }}
10   )
11  SELECT * FROM src_reviews
12 WHERE review_text is not null
13
14  -- now i need to mention how dbt/jinja would know if this is a new record or not.
15
16  {{ if is_incremental() }}
17    AND review_date > (select max(review_date) from {{ this }})
18  {{ endif }}
19

```

```
(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 Concurrency: 1 threads (target='dev')
03:55:30
03:55:35 1 of 6 START sql view model DEV.src_hosts .....
03:55:35 1 of 6 OK created sql view model DEV.src_hosts .....
03:55:35 2 of 6 START sql view model DEV.src_listings .....
03:55:36 2 of 6 OK created sql view model DEV.src_listings .....
03:55:36 3 of 6 START sql view model DEV.src_reviews .....
03:55:36 3 of 6 OK created sql view model DEV.src_reviews .....
03:55:36 4 of 6 START sql table model DEV.dim_hosts_cleansed .....
03:55:37 4 of 6 OK created sql table model DEV.dim_hosts_cleansed .....
03:55:37 5 of 6 START sql table model DEV.dim_listings_cleansed .....
03:55:38 5 of 6 OK created sql table model DEV.dim_listings_cleansed .....
03:55:38 6 of 6 START sql incremental model DEV.fct_reviews .....
03:55:41 6 of 6 OK created sql incremental model DEV.fct_reviews .....
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$ |
```

AIRBNB / DEV / FCT_REVIEWS

Table Details Columns Data Preview Copy History

5 Columns

NAME ↑	TYPE	DESCRIPTION	NULLABLE	DEFAULT
LISTING_ID	# Number		Yes	NULL
REVIEWER_NAME	VarChar		Yes	NULL
REVIEW_DATE	Timestamp_N...		Yes	NULL
REVIEW_SENTIMENT	VarChar		Yes	NULL
REVIEW_TEXT	VarChar		Yes	NULL

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

The screenshot shows the BigQuery interface. On the left, the sidebar displays the CTE section, Databases (AIRBNB.RAW), Worksheets, and various tables like DIM_HOSTS_CLEANSED, DIM_LISTINGS_CLEANSED, and FCT_REVIEWS. The FCT_REVIEWS table is selected. The main area shows a query editor with the following code:

```
AIRBNB.RAW
SELECT * FROM "AIRBNB"."DEV"."FCT_REVIEWS" WHERE listing_id=3176;
```

The results pane shows the following data:

#	LISTING_ID	REVIEW_DATE	REVIEWER_NAME	REVIEW_TEXT
1	3176	2009-06-20 00:00:00.000	Milan	excellent stay, i would highly recommend it. a nice flat in a very nice area.
2	3176	2010-11-07 00:00:00.000	George	Brittas apartment in Berlin is in a great area. There are numerous fantas
3	3176	2010-11-24 00:00:00.000	Patricia	Fantastic, large place in good location. Only a short tram ride to Alexander
4	3176	2010-12-21 00:00:00.000	Benedetta	Lappartamento di Britta è molto largo carino confortabile è in un quartier
5	3176	2011-01-04 00:00:00.000	Aude	We went in Berlin for the new year eve. The appartement of Britta was rea
6	3176	2011-01-12 00:00:00.000	David + Tilla	Location, location, location Brittas place was just what we needed when
7	3176	2011-01-17 00:00:00.000	Andy	We were there to find out what life is like in the city so staying in someo
8	3176	2011-02-28 00:00:00.000	Björn	Recommended Nice, informative and easy to deal with. Beautiful flat in
9	3176	2011-03-07 00:00:00.000	Emmanuel	Brittas flat is very nice and in a wonderful area of Berlin. Organisation h
10	3176	2011-03-22 00:00:00.000	Aurélie	Brittas flat was perfect for our week end in Berlin. It was nice, located in

On the right, the Query Details panel shows: Query duration 170ms, Rows 147, and Query ID 01bae80e-0002-862d-... . Below it are two charts: one for LISTING_ID (100% filled) and one for REVIEW_DATE.

Now we'll, insert a data in it –

The screenshot shows the BigQuery interface. The query editor contains the following code:

```
AIRBNB.RAW
INSERT INTO "AIRBNB"."RAW"."RAW_REVIEWS"
VALUES (3176, CURRENT_TIMESTAMP(), 'Hrishikesh', 'excellent stay!', 'positive');
```

The results pane shows the following data:

#	number of rows inserted
1	1

On the right, the Query Details panel shows: Query duration 599ms, Rows 1, and Query ID 01bae811-0002-862e-0... . Below it is a chart for Rows.

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  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.

The screenshot shows a database interface with the following details:

- Account and Compute:** ACCOUNTADMIN • COMPUTE_WH (X-Small)
- Code Versions:** Code Versions
- Query:**

```
1 | SELECT * FROM "AIRBNB"."DEV"."FCT_REVIEWS" WHERE listing_id=3176;
```
- Results:**

#	LISTING_ID	REVIEW_DATE	REVIEWER_NAME	REVIEW_TEXT
1	3176	2025-03-09 21:01:23.326	Hrishikesh	excellent stay!
2	3176	2020-06-20 00:00:00.000	Milano	excellent stay! I would highly recommend it. a nice flat in a very nice area.
- Query Details:**
 - Query duration: 223ms

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 `listings` table. Also, we are now treating `dim_listings_cleansed` and `dim_hosts_cleansed` as view instead of tables. And `src_hosts`, `src_listings` and `src_reviews` will be CTEs as they wont be much accessed.

The screenshot shows the dbt IDE interface with the following details:

- EXPLORER:** DBTLEARN [WSL: UBUNTU]
- FILES:**
 - `dbt_project.yml`
 - `dim_listings_w_hosts.sql` (selected)
 - `dim_hosts_cleansed.sql`
 - `src_reviews.sql`
 - `fct_reviews.sql`
 - `dim_listings_cleansed.sql`
 - `src_hosts.sql`
 - `src_listings.sql`
 - `src_reviews.sql`
- Code Editor:**

```
models > dim > dim_listings_w_hosts.sql
WITH
  l AS (
    SELECT *
    FROM {{ ref('dim_listings_cleansed') }}
  ),
  h AS (
    SELECT *
    FROM {{ ref('dim_hosts_cleansed') }}
  )
SELECT
  l.listing_id,
  l.listing_name,
  l.room_type,
  l.minimum_nights,
  l.price,
  l.host_id,
  h.host_name,
  h.is_superhost as host_is_superhost,
  l.created_at,
  GREATEST(l.updated_at, h.updated_at) as updated_at
FROM l
LEFT JOIN h ON (h.host_id = l.host_id)
```

Ran **dbt run** command

	LISTING_ID	LISTING_NAME	ROOM_TYPE	MINIMUM_NIGHTS	PRICE	HOST_ID	HOST_NAME
1	3176	Fabulous Flat in great Location	Entire home/apt	62	90.00	3718	Britta
2	7071	BrightRoom with sunny greenview!	Private room	1	33.00	17391	BrightRoom
3	9991	Gorgeous flat - outstanding views	Entire home/apt	1	180.00	33852	Philipp
4	14325	Apartment in Prenzlauer Berg	Entire home/apt	95	70.00	55531	Chris + Oliv
5	16644	In the Heart of Berlin - Kreuzberg	Entire home/apt	60	90.00	64696	Rene
6	17904	Beautiful Kreuzberg studio - 3 months minimum	Entire home/apt	92	47.00	68997	Matthias
7	20858	Designer Loft in Berlin Mitte	Entire home/apt	3	169.00	71331	Marc
8	21869	Studio in the Heart of Kreuzberg	Entire home/apt	60	70.00	64696	Rene
9	22438	WOHNUNG IN BERLIN ★ MITTE	Entire home/apt	90	65.00	86159	Javier
10	22677	Prenzel garden with leafy terrace (quiet Guests)	Entire home/apt	2	120.00	87357	Ramfis
11	23834	Apartment in the heart of Berlin	Entire home/apt	185	65.00	94918	Tanja

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.

```

File Edit Selection View Go Run Terminal Help ⌘ ⌘ dbtlearn [WSL: Ubuntu]
EXPLORER
  ✓ DBTLEARN [WSL: UBUNTU]
    > .vscode
    > analyses
    > dbt_packages
    > logs
    > macros
    ✓ models
      ✓ dim
        ✘ dim_hosts_cleanse...
        ✘ dim_listings_cleanse...
        ✘ dim_listings_w_hosts...
      ✓ fct
        ✘ fct_reviews.sql
      ✓ src
        ✘ src_hosts.sql
        ✘ src_listings.sql
        ✘ src_reviews.sql
      > seeds
      > snapshots
      > target
      > tests
      & .gitignore
    & dbt_project.yml
    README.md

dbt_project.yml
  19  snapshots:
  20  21  clean-targets:          # directories to be removed by `dbt clean`
  22  23  - "target"
  24  25  - "dbt_packages"
  26  # Configuring models
  27  # Full documentation: https://docs.getdbt.com/docs/configuring-models
  28
  29  # In this example config, we tell dbt to build all models in the example/
  30  # directory as views. These settings can be overridden in the individual model
  31  # files using the `{{ config(...)}}` macro.
  32  models:
  33    dbtLearn:
  34      # Config indicated by + and applies to all files under models/example/
  35      +materialized: view # here hashtag means i do not want a subfolder named materialized and sets this materialized to default
  36      dim:
  37        +materialized: table
  38      src:
  39        +materialized: ephemeral

```

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 -

```
31 # files using the `{{ config(...) }}` macro.
32 models:
33   dbtlearn:
34     # Config indicated by + and applies to all
35     +materialized: view # here hastag means i
36     dim:
37       +materialized: table
38     src:
39       +materialized: ephemeral
```

now explicitly mentioning both the dim models-

```
EXPLORER      ...
DBTLEARN [WSL: UBUNTU]
  .vscode
  analyses
  dbt_packages
  logs
  macros
  models
    dim
      dim_hosts_cleanse...
      dim_listings_cleanse...
      dim_listings_w_hos...
models > dim > dim_listings_cleanse.sql
  Add documentation or tests | Datapilot Notebooks
  1  {{ config(
  2    materialized = 'view'
  3  )
  4  }}
  5  }
  6  |
  7  WITH src_listings AS (
  8    SELECT
  9      *
 10     FROM
 11      {{ ref('src_listings') }}
```

```
(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_cleanse..... [RUN]
03:51:19  1 of 4 OK created sql view model DEV.dim_hosts_cleanse..... [SUCCESS 1 in 0.58s]
03:51:19  2 of 4 START sql view model DEV.dim_listings_cleanse..... [RUN]
03:51:20  2 of 4 OK created sql view model DEV.dim_listings_cleanse..... [SUCCESS 1 in 0.63s]
03:51:20  3 of 4 START sql incremental model DEV.fct_reviews..... [RUN]
03:51:20  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 C

AIRBNB

DEV

Tables

DIM_LISTINGS_W_HOSTS

FCT_REVIEWS

Views

DIM_HOSTS_CLEANSED

DIM_LISTINGS_CLEANSED

INFORMATION_SCHEMA

PUBLIC

RAW

SNOWFLAKE

SNOWFLAKE_SAMPLE_DATA

The screenshot shows a database navigation interface with the following structure:

- Databases** tab is selected.
- AIRBNB** database is expanded.
- DEV** schema is expanded.
- Tables** section contains:
 - DIM_LISTINGS_W_HOSTS** (highlighted with a blue box)
 - FCT_REVIEWS**
- Views** section contains:
 - DIM_HOSTS_CLEANSED**
 - DIM_LISTINGS_CLEANSED**
- INFORMATION_SCHEMA**, **PUBLIC**, and **RAW** schemas are collapsed.
- SNOWFLAKE** and **SNOWFLAKE_SAMPLE_DATA** are collapsed.

SEEDS AND SOURCES –

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

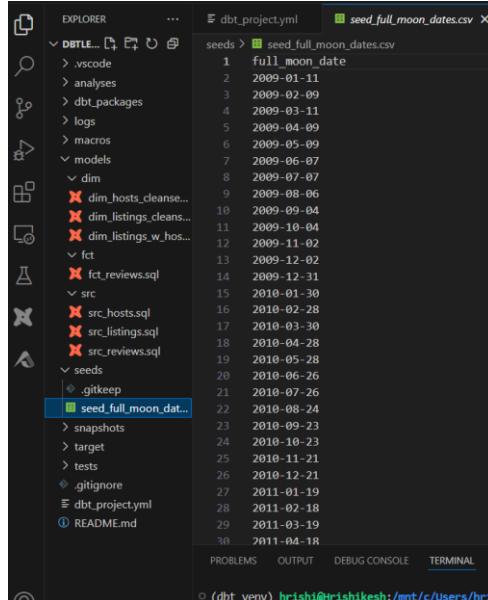
- **Sources** are your raw, existing data that's already in your data warehouse. Think of them as the starting ingredients that you want to clean up or combine.
- **Seeds** are CSV files you include in your dbt project. They're like small, ready-made datasets you commit to version control, and dbt can 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/hrishi/Desktop/temp/DBT - Udemy/dbtlearn$ curl https://dbtlearn.s3.us-east-2.amazonaws.com/seed_full_moon_dates.csv -o seeds/seed_full_moon_dates.csv
% Total    % Received % Xferd  Average Speed   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/hrishi/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/hrishi/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')
20:52:42 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 Finished running 1 seed in 0 hours 0 minutes and 6.86 seconds (6.86s).
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/hrishi/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.

```

models > mart > mart_fullmoon_reviews.sql
Add documentation or tests | Datapilot Notebooks
1 + {{ config(
2   materialized = 'table',
3   )} } {# Materializing this as a table#}
4
5 WITH fct_reviews AS (
6   SELECT * FROM {{ ref('fct_reviews') }}
7 ),
8   full_moon_dates AS (
9   SELECT * FROM {{ ref('seed_full_moon_dates') }}
10 )
11
12 SELECT
13   r.*,
14   CASE
15     WHEN fm.full_moon_date IS NULL THEN 'not full moon'
16     ELSE 'full moon'
17   END AS is_full_moon
18 FROM
19   fct_reviews
20   r
21   LEFT JOIN full_moon_dates
22   fm
23   ON (TO_DATE(r.review_date) = DATEADD(DAY, 1, fm.full_moon_date)) {#here we are joining every full moon date with the upcoming/same date in the reviews#}

```

```
(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.

The screenshot shows the Snowflake web interface. On the left, the navigation sidebar includes options like '+ Create', 'Home', 'Search', 'Projects', 'Data' (selected), 'Databases' (selected), 'Data Products', 'AI & ML', 'Monitoring', and 'Admin'. A message at the bottom left says '\$383 credits left' and 'Trial ends in 25 days'. On the right, the main area displays the 'AIRBNB' database structure under the 'DEV' schema. The 'Tables' section shows 'DIM_LISTINGS_W_HOSTS', 'FCT_REVIEWS', 'MART_FULLMOON_REVIEWS' (which is highlighted in blue), and 'SEED_FULL_MOON_DATES'. Below it, 'Views' include 'DIM_HOSTS_CLEANSED' and 'DIM_LISTINGS_CLEANSED'. Under 'INFORMATION_SCHEMA', there's a 'PUBLIC' schema. Under 'RAW', there are 'RAW_HOSTS', 'RAW_LISTINGS', and 'RAW_REVIEWS'. Under 'SNOWFLAKE', there's 'SNOWFLAKE_SAMPLE_DATA'. The 'MART_FULLMOON_REVIEWS' table is selected, showing its details: 'Table' (selected), 'TRANSFORM' (disabled), 'just now', 409.7K rows, 42.7MB size. The 'Data Preview' tab is active, showing 100 of 409.7K Rows updated just now. The preview table has columns 'REVIEW_SENTIMENT' and 'IS_FULL_MOON'. The data rows are as follows:

	REVIEW_SENTIMENT	IS_FULL_MOON
1	positive	not full moon
2	positive	not full moon
3	neutral	not full moon
4	neutral	not full moon
5	positive	not full moon
6	positive	not full moon
7	neutral	not full moon
8	positive	not full moon
9	positive	not full moon
10	neutral	not full moon
11	neutral	not full moon
12	positive	not full moon
13	neutral	not full moon
14	positive	not full moon
15	neutral	not full moon

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.

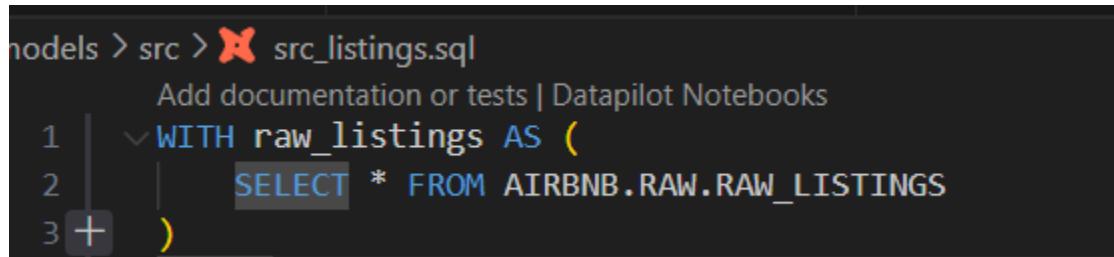
4. Better Data Lineage:

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.



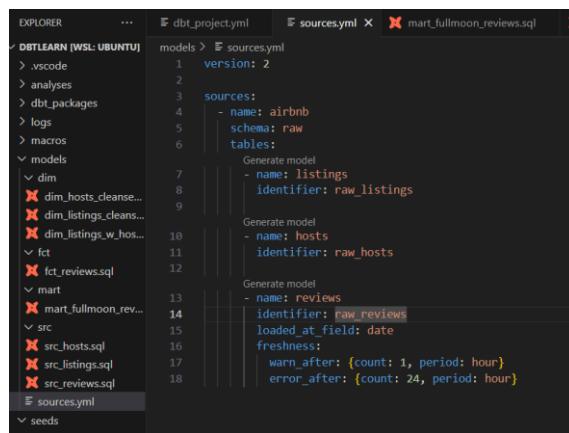
The screenshot shows a code editor with a dark theme. The file is named 'src_listings.sql'. The code is as follows:

```
models > src > ✘ src_listings.sql
          Add documentation or tests | Datapilot Notebooks
1   |   ✓ WITH raw_listings AS (
2   |   |   SELECT * FROM AIRBNB.RAW.RAW_LISTINGS
3   |   |   )

```

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

With sources –



The screenshot shows a code editor with a dark theme. The file is named 'sources.yml'. The content is as follows:

```
EXPLORER ... dbt_project.yml sources.yml ✘ mart_fullmoon_reviews.sql
DBLEARN (WSL: UBUNTU)
models > sources.yml
  1 version: 2
  2
  3 sources:
  4   - name: airbnb
  5     schema: raw
  6     tables:
  7       Generate model
  8       - name: listings
  9         identifier: raw_listings
 10      Generate model
 11      - name: hosts
 12        identifier: raw_hosts
 13      Generate model
 14      - name: reviews
 15        identifier: raw_reviews
 16        loaded_at_field: date
 17        freshness:
 18          warn_after: {count: 1, period: hour}
 19          error_after: {count: 24, period: hour}
```

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
1   -- import raw listings
2   WITH raw_listings AS (
3       SELECT * FROM {{ source('airbnb', 'listings') }}
4   )
5   SELECT
6       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

The screenshot shows the DBT IDE interface. On the left is the EXPLORER sidebar with project structure. In the center is the code editor with the dbt_project.yml file open at the top. Below it is the scd_raw_listings.sql file, which contains the following code:

```
{% snapshot scd_raw_listings %}
  {{
    config(
      target_schema='DEV',
      unique_key='id',
      strategy='timestamp',
      updated_at='updated_at',
      invalidate_hard_deletes=True
    )
  }}
  select * FROM {{ source('airbnb', 'listings') }}
{% endsnapshot %}
```

```
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$ dbt snapshot
22:34:08  Running with dbt=1.9.3
22:34:11  Registered adapter: snowflake=1.9.0
22:34:27  Found 8 models, 1 seed, 1 snapshot, 3 sources, 472 macros
22:34:27
22:34:27  Concurrency: 1 threads (target='dev')
22:34:27
22:34:31  1 of 1 START snapshot DEV.scd_raw_listings ..... [RUN]
22:34:33  1 of 1 OK snapshotted DEV.scd_raw_listings ..... [SUCCESS 1 in 1.88s]
22:34:33
22:34:33  Finished running 1 snapshot in 0 hours 0 minutes and 6.40 seconds (6.40s).
22:34:33
22:34:33  Completed successfully
22:34:33
22:34:33  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 will create a snapshot table in dev as mentioned in the .sql file above.

AIRBNB / DEV / SCD_RAW_LISTINGS

Table Details Columns Data Preview Copy History

13 Columns

NAME	TYPE	DESCRIPTION	NULLABLE	DEFAULT
CREATED_AT	Timestamp		Yes	NULL
DBT_SCD_ID	Varchar		Yes	NULL
DBT_UPDATED_AT	Timestamp		Yes	NULL
DBT_VALID_FROM	Timestamp		Yes	NULL
DBT_VALID_TO	Timestamp		Yes	NULL
HOST_ID	Number		Yes	NULL
ID	Number		Yes	NULL
LISTING_URL	Varchar		Yes	NULL
MINIMUM_NIGHTS	Number		Yes	NULL
NAME	Varchar		Yes	NULL
PRICE	Varchar		Yes	NULL

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

```

1 UPDATE AIRBNB.RAW.RAW_LISTINGS SET MINIMUM_NIGHTS=30,
2     updated_at=CURRENT_TIMESTAMP() WHERE ID=3176;
3
4 SELECT * FROM AIRBNB.DEV.SCD_RAW_LISTINGS WHERE ID=3176;

```

Results

ID	LISTING_URL	NAME	ROOM_TYPE	MINIMUM_NIGHTS	HOST_ID	PRICE	CREATED_AT
3176	https://www.airbnb.com	Fabulous Flat in great Location	Entire home/apt	62	3718	\$90.00	2009-06-05 21:34:42.000

Query Details

- Query duration: 326ms
- Rows: 1
- Query ID: 01baf225-0002-88c9-0...

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

Results

LISTING_URL	NAME	ROOM_TYPE	MINIMUM_NIGHTS	HOST_ID	PRICE	CREATED_AT
www.airbnb.com	Fabulous Flat in great Location	Entire home/apt	30	3718	\$90.00	2009-06-05 21:34:42.000

Query Details

- Query duration: 298ms
- Rows: 1

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

Results

UPDATED_AT	DBT_SCD_ID	DBT_UPDATED_AT	DBT_VALID_FROM	DBT_VALID_TO
2009-06-05 21:34:42.000	c9e3bc0b5eb3a808ee31530eccdfa503	2009-06-05 21:34:42.000	2009-06-05 21:34:42.000	null

Query Details

- Query duration: 42ms
- Rows: 1
- Query ID: 01baf22b-0002-88c9-0...

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

```
(dbt_venv) hrishi@HrishiKesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$ dbt snapshot
23:08:17 Running with dbt=1.9.3
23:08:23 Registered adapter: snowflake=1.9.0
23:08:47 Found 8 models, 1 seed, 1 snapshot, 3 sources, 472 macros
23:08:47 Concurrency: 1 threads (target='dev')
23:08:47
23:08:53 1 of 1 START snapshot DEV.scd_raw_listings ..... [RUN]
23:09:09 1 of 1 OK snapshotted DEV.scd_raw_listings ..... [SUCCESS 2 in 15.72s]
23:09:09
23:09:09 Finished running 1 snapshot in 0 hours 0 minutes and 21.90 seconds (21.90s).
23:09:09
23:09:09 Completed successfully
23:09:09
23:09:09 Done. PASS=1 WARN=0 ERROR=0 SKIP=0 TOTAL=1
(dbt_venv) hrishi@HrishiKesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$ |
```

```
4
5 | SELECT * FROM AIRBNB.DEV.SCD_RAW_LISTINGS WHERE ID=3176;
```

Results

	DBT_SCD_ID	DBT_UPDATED_AT	DBT_VALID_FROM	DBT_VALID_TO
1	82a0daeaa83c76f8b12ef5bd356341995	2025-03-11 16:03:08.101	2025-03-11 16:03:08.101	null
2	c9e3bc0b5eb3a808ee31530eccdfa503	2009-06-05 21:34:42.000	2009-06-05 21:34:42.000	2025-03-11 16:03:08.101

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.
 - o 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.
 - o 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.
 - o If it’s set to null, that means this version is currently the active one.
 - o 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 –

The screenshot shows the VS Code interface with the DBT Explorer extension open. The Explorer sidebar on the left lists project files: dbt_project.yml, scd_raw_listings.sql, schema.yml (which is currently selected), sources.yml, seeds, snapshots, target, tests, .gitignore, dbt_project.yml, and README.md. The main editor area displays the schema.yml file, which defines a model named 'dim_listings_cleansed' with columns 'listing_id' and 'host_id'. Tests for 'listing_id' include 'unique' and 'not_null'. Tests for 'host_id' include 'not_null' and 'relationships' (to 'dim_hosts_cleansed'). The 'room_type' column has accepted values: 'Entire home/apt', 'Private room', 'Shared room', and 'Hotel room'.

```

version: 2
models:
  models:
    Add documentation or tests
    - name: dim_listings_cleansed
      columns:
        - name: listing_id
          tests:
            - unique
            - not_null
        - name: host_id
          tests:
            - not_null
            - relationships:
                to: ref('dim_hosts_cleansed')
                field: host_id
        - name: room_type
          tests:
            - accepted_values:
                values: ['Entire home/apt',
                          'Private room',
                          'Shared room',
                          'Hotel room']

```

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

```
(dbt_venv) hrishi@HrishiKesh:/mnt/c/Users/hrishi/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_apartment__Private_room__Shared_room__Hotel_room [RUN]
00:28:47 1 of 5 PASS accepted_values_dim_listings_cleansed_room_type__Entire_home_apartment__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/hrishi/Desktop/temp/DBT - Udemy/dbtlearn$ |

```

The 'pass' shows that all the tests were successfully made.

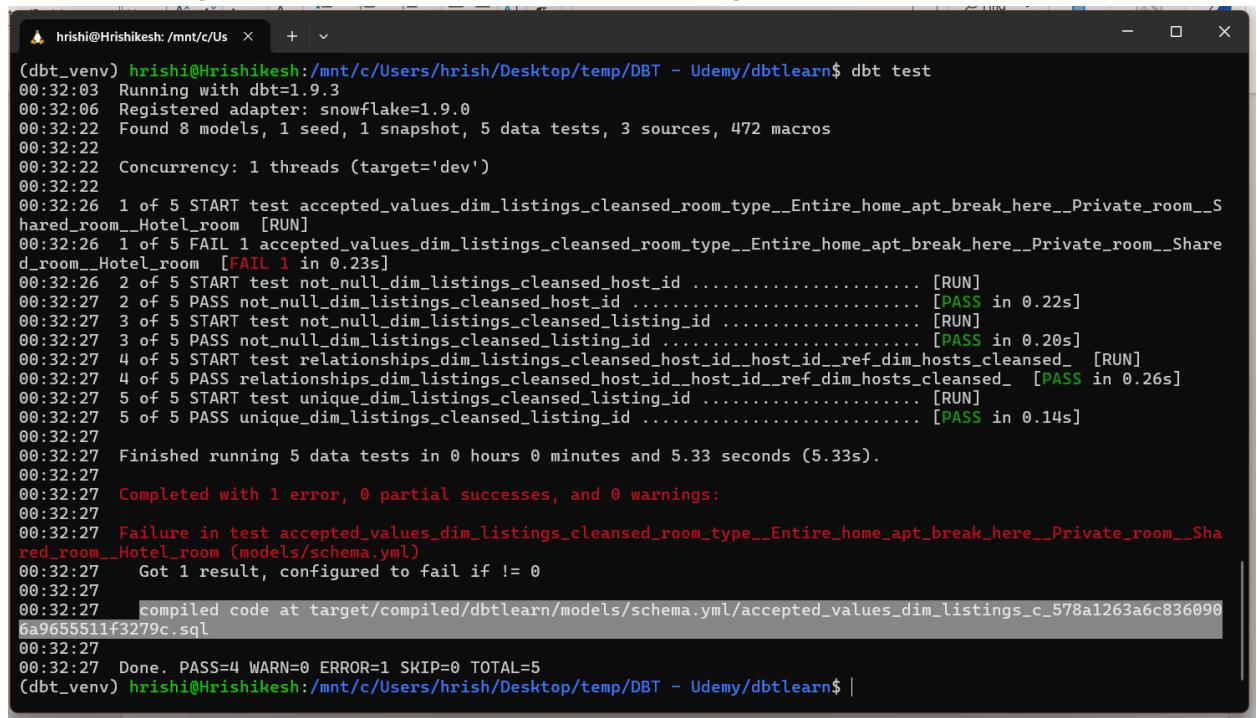
NOW debugging if there's any test that fails.

```

19     - name: room_type
20       tests:
21         - accepted_values:
22           values: ['Entire home/apt - break here',
23                     'Private room',
24                     'Shared room',
25                     'Hotel room']

```

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



```

hrishi@HrishiKesh: /mnt/c/Us x + | ~
(dbt_venv) hrishi@HrishiKesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$ dbt test
00:32:03 Running with dbt=1.9.3
00:32:06 Registered adapter: snowflake=1.9.0
00:32:22 Found 8 models, 1 seed, 1 snapshot, 5 data tests, 3 sources, 472 macros
00:32:22
00:32:22 Concurrency: 1 threads (target='dev')
00:32:22
00:32:26 1 of 5 START test accepted_values_dim_listings_cleansed_room_type__Entire_home_apt_break_here__Private_room__S
hared_room__Hotel_room [RUN]
00:32:26 1 of 5 FAIL 1 accepted_values_dim_listings_cleansed_room_type__Entire_home_apt_break_here__Private_room__Share
d_room__Hotel_room [FAIL 1 in 0.23s]
00:32:26 2 of 5 START test not_null_dim_listings_cleansed_host_id ..... [RUN]
00:32:27 2 of 5 PASS not_null_dim_listings_cleansed_host_id ..... [PASS in 0.22s]
00:32:27 3 of 5 START test not_null_dim_listings_cleansed_listing_id ..... [RUN]
00:32:27 3 of 5 PASS not_null_dim_listings_cleansed_listing_id ..... [PASS in 0.20s]
00:32:27 4 of 5 START test relationships_dim_listings_cleansed_host_id__host_id__ref_dim_hosts_cleansed_ [RUN]
00:32:27 4 of 5 PASS relationships_dim_listings_cleansed_host_id__host_id__ref_dim_hosts_cleansed_ [PASS in 0.26s]
00:32:27 5 of 5 START test unique_dim_listings_cleansed_listing_id ..... [RUN]
00:32:27 5 of 5 PASS unique_dim_listings_cleansed_listing_id ..... [PASS in 0.14s]
00:32:27
00:32:27 Finished running 5 data tests in 0 hours 0 minutes and 5.33 seconds (5.33s).
00:32:27
00:32:27 Completed with 1 error, 0 partial successes, and 0 warnings:
00:32:27
00:32:27 Failure in test accepted_values_dim_listings_cleansed_room_type__Entire_home_apt_break_here__Private_room__Sha
red_room__Hotel_room (models/schema.yml)
00:32:27     Got 1 result, configured to fail if != 0
00:32:27
00:32:27     compiled code at target/compiled/dbtlearn/models/schema.yml/accepted_values_dim_listings_c_578a1263a6c836090
6a9655511f3279c.sql
00:32:27
00:32:27 Done. PASS=4 WARN=0 ERROR=1 SKIP=0 TOTAL=5
(dbt_venv) hrishi@HrishiKesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$ |

```

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

```

hrishi@Hrishikesh: /mnt/c/Us + | v
00:32:27 Failure in test accepted_values_dim_listings_cleansed_room_type__Entire_home_apt_break_here__Private_room__Sha
red_room__Hotel_room (models/schema.yml)
00:32:27 Got 1 result, configured to fail if != 0
00:32:27
00:32:27 compiled code at target/compiled/dbtlearn/models/schema.yml/accepted_values_dim_listings_c_578a1263a6c836090
6a9655511f3279c.sql
00:32:27
00:32:27 Done. PASS=4 WARN=0 ERROR=1 SKIP=0 TOTAL=5
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$ cat target/compiled/dbtlearn/models/s
chema.yml/accepted_values_dim_listings_c_578a1263a6c8360906a9655511f3279c.sql

with all_values as (
    select
        room_type as value_field,
        count(*) as n_records
    from AIRBNB.DEV.dim_listings_cleansed
    group by room_type
)
select *
from all_values
where value_field not in (
    'Entire home/apt - break here', 'Private room', 'Shared room', 'Hotel room'
)

(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$ |

```

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.

The screenshot shows the DBT IDE interface. On the left, the Explorer sidebar displays the project structure:

- DBTLE... (selected)
- .vscode
- analyses
- dbt_packages
- logs
- macros
- models
 - dim
 - fct
 - mart
 - src
- schema.yml
- sources.yml
- seeds
 - .gitkeep
 - seed_full_moon_dat...
- snapshots
 - .gitkeep
 - scd_raw_listings.sql
- target
- tests
 - .gitkeep
 - dim_listings_minimu...
 - .gitignore
- dbt_project.yml
- README.md

In the center, the code editor shows a SQL file named `dim_listings_minimum_nights.sql`:

```

1 | SELECT *
2 | FROM
3 |   {{ ref('dim_listings_cleansed') }}
4 | WHERE minimum_nights < 1
5 |
6 | LIMIT 10
7 |

```

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_cleanse... [RUN]
00:54:36  5 of 6 PASS relationships_dim_listings_cleansed_host_id__host_id_ref_dim_hosts_cleanse... [PASS in 0.01s]
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_cleaned
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 Concurrency: 1 threads (target='dev')
01:09:54
01:09:58 1 of 6 START test accepted_values_dim_listings_cleaned_room_type__Entire_home_apartment__Private_room__Shared_room__Hotel_room [RUN]
01:09:58 1 of 6 PASS accepted_values_dim_listings_cleaned_room_type__Entire_home_apartment__Private_room__Shared_room__Hotel_room [PASS 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_cleaned_host_id ..... [RUN]
01:09:58 3 of 6 PASS not_null_dim_listings_cleaned_host_id ..... [PASS in 0.14s]
01:09:58 4 of 6 START test not_null_dim_listings_cleaned_listing_id ..... [RUN]
01:09:58 4 of 6 PASS not_null_dim_listings_cleaned_listing_id ..... [PASS in 0.22s]
01:09:58 5 of 6 START test relationships_dim_listings_cleaned_host_id__host_id_ref_dim_hosts_cleaned_ [RUN]
01:09:59 5 of 6 PASS relationships_dim_listings_cleaned_host_id__host_id_ref_dim_hosts_cleaned_ [PASS in 0.13s]
01:09:59 6 of 6 START test unique_dim_listings_cleaned_listing_id ..... [RUN]
01:09:59 6 of 6 PASS unique_dim_listings_cleaned_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_cleaned.

The screenshot shows the Datapilot IDE interface. On the left is the Explorer sidebar with the DBTLEARN [WSL: UBUNTU] workspace selected. It lists various files and folders: dbt_project.yml, scd_raw_listings.sql, schema.yml, dim_listings_minimum_nights.sql, consistent_created_at.sql (which is currently open), .vscode, analyses, dbt_packages, logs, macros, models (dim, fct, mart), src (src_hosts.sql, src_listings.sql, src_reviews.sql), schema.yml, sources.yml, seeds (.gitkeep, seed_full_moon.dat), snapshots (.gitkeep, scd_raw_listings.sql), target, tests (.gitkeep, consistent_created_at.sql), and .gitignore. The main area shows the code for consistent_created_at.sql:

```
tests > consistent_created_at.sql
          Add documentation or tests | Datapilot Notebooks
1  SELECT * FROM {{ ref('dim_listings_cleaned') }} l
2  INNER JOIN {{ ref('fct_reviews') }} r
3  USING (listing_id)
4  WHERE l.created_at >= r.review_date
```

At the bottom, the terminal tab is active, showing the command dbt test --select path:tests/consistent_created_at.sql.

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 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

MACROS, CUSTOM TESTS AND PACKAGES

- Macros are jinja templates created in the *macros* folder
- There are many built-in macros in DBT
- You can use macros in model definitions and tests
- A special macro, called *test*, can be used for implementing your own generic tests
- dbt packages can be installed easily to get access to a plethora of macros and tests

Now, we create a macro under macro folder -

```
lbt_project.yml  ✘ scd_raw_listings.sql  🌐 schema.yml  ✘ dim_listings_minimu
macros > ✘ no_nulls_in_columns.sql
          Add documentation or tests | Datapilot Notebooks
1  {% macro no_nulls_in_columns(model) %}
2    SELECT * FROM {{ model }} WHERE
3    {% for col in adapter.get_columns_in_relation(model) -%}
4      {{ col.column }} IS NULL OR
5    {% endfor %}
6    FALSE
7  [% endmacro %]
```

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`.

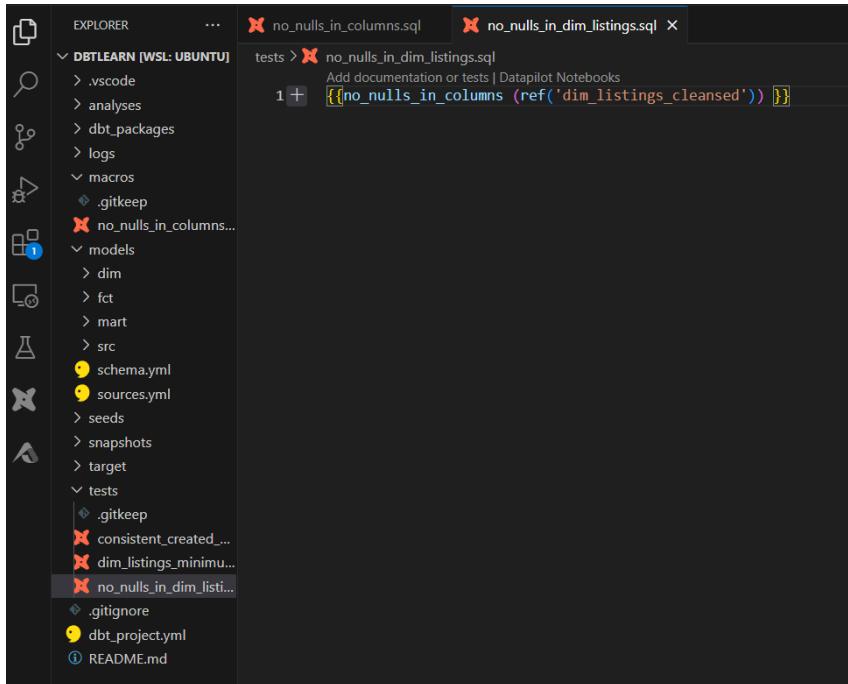
- This makes sure the SQL statement is valid by ending the WHERE clause logically (so you don't end with an extra OR).

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 –



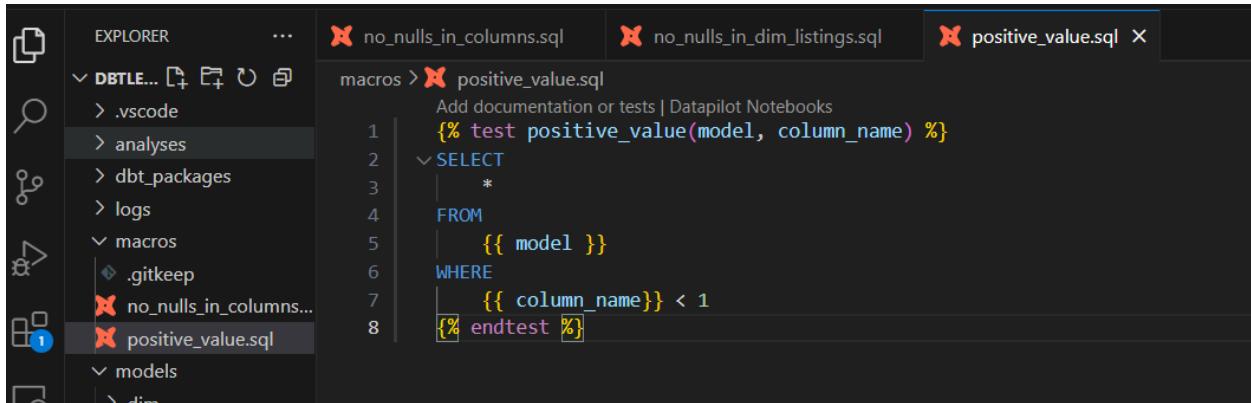
As you know, we can call a particular test on a specific model for which the test is to be runned as shown below. The highlighted text shows that the test has been passed successfully.

```
(dbt_venv) hrishi@HrishiKesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$ dbt test --select dim_listings_cleanse
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_apartment__Private_room__Shared_room_
_Hotel_room [RUN]
00:24:47 1 of 8 PASS accepted_values_dim_listings_cleansed_room_type__Entire_home_apartment__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_cleanse_ [RUN]
00:24:51 7 of 8 PASS relationships_dim_listings_cleansed_host_id__host_id__ref_dim_hosts_cleanse_ [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 your 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. -



The screenshot shows the VS Code interface with the DBT Explorer sidebar on the left. The sidebar lists project files: .vscode, analyses, dbt_packages, logs, macros (which contains .gitkeep), no_nulls_in_columns.sql, positive_value.sql, models, and dim. The code editor on the right displays a SQL script named positive_value.sql under the macros directory. The script contains a single test macro:

```
% test positive_value(model, column_name) %}  
SELECT *  
FROM {{ model }}  
WHERE {{ column_name }} < 1  
% endtest %}
```

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 macros 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.

The screenshot shows the VS Code interface with the following details:

- EXPLORER** view: Shows the project structure under "DBTLEARN [WSL: UBUNTU]". It includes:
 - Analyses
 - dbt_packages
 - Logs
 - Macros (with .gitkeep)
 - Models (with no_nuls_in_columns.sql and positive_value.sql)
 - Sources (with schema.yml, sources.yml, and seeds)
 - Snapshots
 - Target
 - Tests (with .gitkeep, consistent_created_at, dim_listings_minimum_nights, no_nuls_in_dim_listings, and .gitignore)
 - Src
 - README.md
- SCHEMA.YML** file content (selected in the Explorer):


```

version: 2
models:
  - name: dim_listings_cleansed
    columns:
      - name: listing_id
        tests:
          - unique
          - not_null
      - name: host_id
        tests:
          - not_null
          - relationships:
              to: ref('dim_hosts_cleansed')
              field: host_id
      - name: room_type
        tests:
          - accepted_values:
              values: ['Entire home/apt',
                        'Private room',
                        'Shared room',
                        'Hotel room']
      - name: minimum_nights
        tests:
          - positive_value
      
```
- OTHER FILES** (visible in the top bar):
 - no_nuls_in_columns.sql
 - no_nuls_in_dim_listings.sql
 - schema.yml (highlighted)
 - positive_value.sql

name: minimum_nights = column name

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

```

hrishi@HrishiKesh: /mnt/c/Us  ~  +  -
01:01:16
01:01:16 Concurrency: 1 threads (target='dev')
01:01:16
01:01:20 1 of 9 START test accepted_values_dim_listings_cleansed_room_type__Entire_home_apartment__Private_room__Shared_room__Hotel_room [RUN]
01:01:21 1 of 9 PASS accepted_values_dim_listings_cleansed_room_type__Entire_home_apartment__Private_room__Shared_room__Hotel_room [PASS in 0.17s]
01:01:21 2 of 9 START test consistent_created_at ..... [RUN]
01:01:21 2 of 9 PASS consistent_created_at ..... [PASS in 0.21s]
01:01:21 3 of 9 START test dim_listings_minimum_nights ..... [RUN]
01:01:21 3 of 9 PASS dim_listings_minimum_nights ..... [PASS in 0.14s]
01:01:21 4 of 9 START test no_nuls_in_dim_listings ..... [RUN]
01:01:21 4 of 9 PASS no_nuls_in_dim_listings ..... [PASS in 0.24s]
01:01:21 5 of 9 START test not_null_dim_listings_cleansed_host_id ..... [RUN]
01:01:21 5 of 9 PASS not_null_dim_listings_cleansed_host_id ..... [PASS in 0.12s]
01:01:21 6 of 9 START test not_null_dim_listings_cleansed_listing_id ..... [RUN]
01:01:21 6 of 9 PASS not_null_dim_listings_cleansed_listing_id ..... [PASS in 0.16s]
01:01:21 7 of 9 START test positive_value_dim_listings_cleansed_minimum_nights ..... [RUN]
01:01:22 7 of 9 PASS positive_value_dim_listings_cleansed_minimum_nights ..... [PASS in 0.34s]
01:01:22 8 of 9 START test relationships_dim_listings_cleansed_host_id__host_id__ref_dim_hosts_cleansed_ [RUN]
01:01:22 8 of 9 PASS relationships_dim_listings_cleansed_host_id__host_id__ref_dim_hosts_cleansed_ [PASS in 0.18s]
01:01:22 9 of 9 START test unique_dim_listings_cleansed_listing_id ..... [RUN]
01:01:22 9 of 9 PASS unique_dim_listings_cleansed_listing_id ..... [PASS in 0.16s]
01:01:22
01:01:22 Finished running 9 data tests in 0 hours 0 minutes and 6.37 seconds (6.37s).
01:01:22
01:01:22 Completed successfully
01:01:22

```

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.

The screenshot shows the VS Code interface with the following details:

- EXPLORER**: Shows the project structure under "DBTLEARN [WSL: UBUNTU]".
- CODE EDITOR**: Displays two files:
 - `packages.yml`: Contains the following YAML code:

```
1 packages:
2   - package: dbt-labs/dbt_utils
3     version: 1.3.0
```
 - `no_nulls_in_columns.sql`: A SQL script.
- TERMINAL**: Shows the command `dbt deps` being run in a terminal window.

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 –

The screenshot shows the dbt CLI interface with the following details:

- File Menu:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Terminal:** dbtlearn [WSL: Ubuntu].
- Explorer:** Shows the project structure:
 - DBTLEARN [WSL: UBUNTU]
 - > vsource
 - > analyses
 - > dbt_packages
 - > logs
 - > macros
 - .gitkeep
 - > no_nulls_in_columns...
 - > positive_value.sql
 - models > fct > fct_reviews.sql
- Code Editor:** The fct_reviews.sql file is open, showing dbt configuration and SQL code for generating a surrogate key from review_text.
- Bottom Navigation:** PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, PORTS 1, QUERY RESULTS, LINEAGE, DOCUMENTATION EDITOR, ACTIONS, COMMENTS.
- Bottom Right:** bash - dbtlearn +

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 an error –

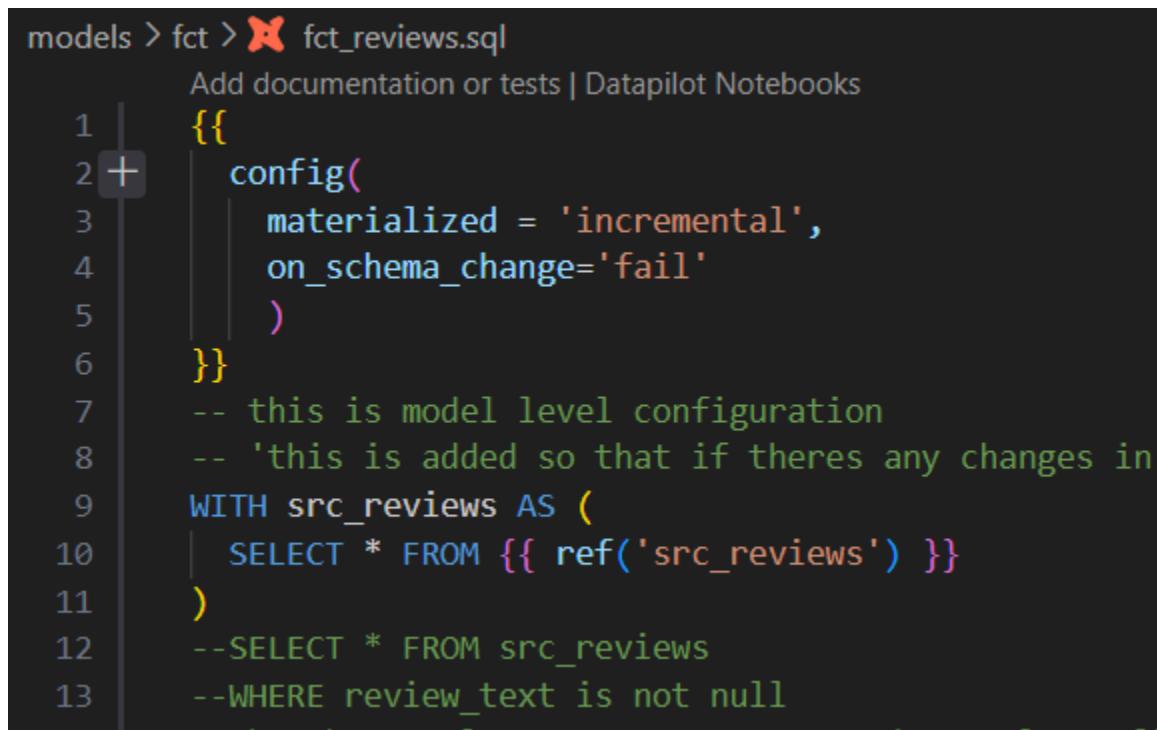
```
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrishi/Desktop/temp/DBT - Udemy/dbtlearn$ dbt run --select fct_reviews
01:58:27 Running with dbt=1.9.3
01:58:33 Registered adapter: snowflake=1.9.0
01:59:01 Unable to do partial parsing because a project dependency has been added
01:59:16 Found 8 models, 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 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 Done. PASS=0 WARN=0 ERROR=1 SKIP=0 TOTAL=1
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrishi/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 –

The screenshot shows the Snowflake Data Preview interface for the AIRBNB / DEV / FCT_REVIEWS table. The left sidebar shows the schema structure under the AIRBNB database, with the DEV schema expanded to show Tables (DIM_LISTINGS_W_HOSTS, FCT_REVIEWS, MART_FULLMOON_REVIEWS, SCD_RAW_LISTINGS, SEED_FULL_MOON_DATES) and Views. The FCT_REVIEWS table is selected and highlighted with a blue background.

The main preview area displays the following information:

- Table Details:** COMPUTE_WH, 100 of 409.7K Rows, Updated just now.
- Columns:** REVIEW_ID, LISTING_ID, REVIEW_DATE, REVIEWER_NAME.
- Data Preview:** A grid of 15 rows of sample data.

	REVIEW_ID	LISTING_ID	REVIEW_DATE	REVIEWER_NAME
1	01747a5d79f6f0bdb4ed9524490a1e8	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	4cf102287c4b5b6f49f5c0ae0f078e30d	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	647d52970727be06f0b0ff1f2089b10ba	14285959	2018-12-27T00:00:00Z	Buse
9	4ce00b0ca19c8c82591e76cb0f76bc38	14285959	2018-12-30T00:00:00Z	Vanessa
10	11946d2e1b4c99b8a260339535357dbf	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.

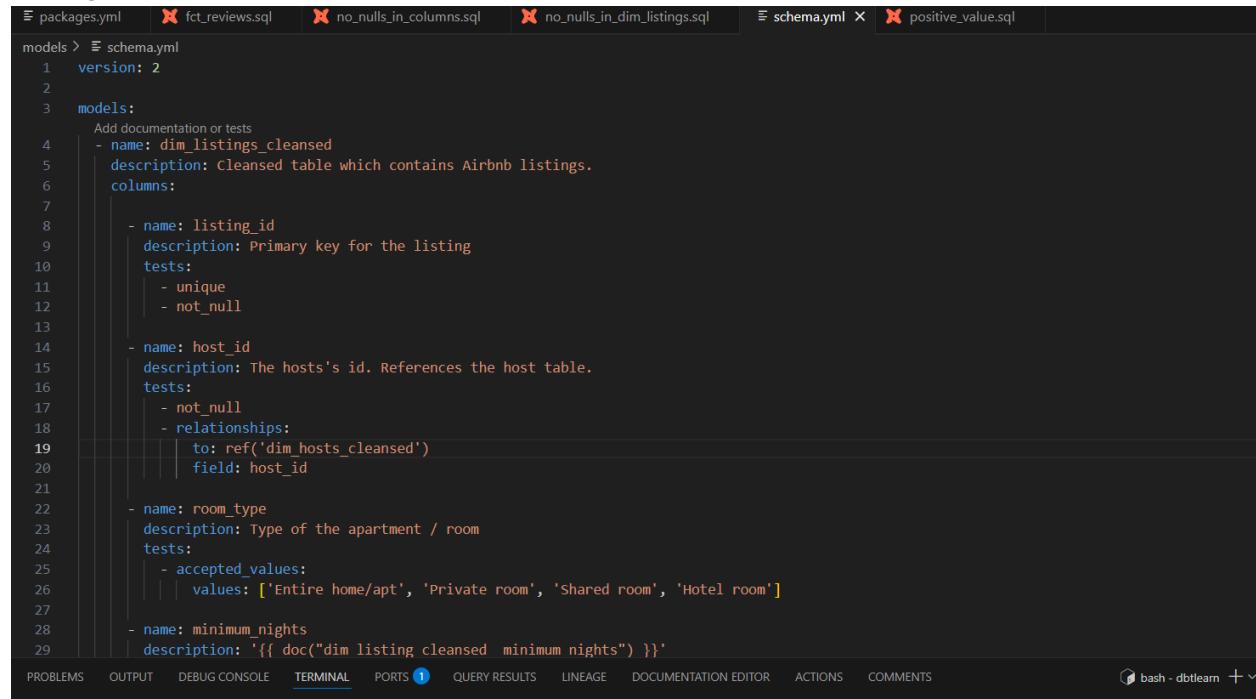
DOCUMENTATION

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

Documentation overview

- Documentations can be defined 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 -



```
models > schema.yml
1 version: 2
2
3 models:
  Add documentation or tests
4   - name: dim_listings_cleansed
5     description: Cleansed table which contains Airbnb listings.
6     columns:
7       - name: listing_id
8         description: Primary key for the listing
9         tests:
10           - unique
11           - not_null
12
13       - name: host_id
14         description: The host's id. References the host table.
15         tests:
16           - not_null
17           - relationships:
18             - to: ref('dim_hosts_cleanse')
19               field: host_id
20
21       - name: room_type
22         description: Type of the apartment / room
23         tests:
24           - accepted_values:
25             - values: ['Entire home/apt', 'Private room', 'Shared room', 'Hotel room']
26
27       - name: minimum_nights
28         description: '{{ doc("dim_listing_cleanse minimum_nights") }}'
```

Dbt doc generate is the command to generate docs.

```
(dbt_venv) hrishi@HrishiKesh:/mnt/c/Users/hrishi/Desktop/temp/DBT - Udemy/dbtlearn$ dbt docs generate
1 05:29:56 Running with dbt=1.9.3
2 05:30:00 Registered adapter: snowflake=1.9.0
3 05:30:21 Found 8 models, 1 snapshot, 16 data tests, 1 seed, 3 sources, 590 macros
4 05:30:21 Concurrency: 1 threads (target='dev')
5 05:30:21
6 05:30:27 Building catalog
7 05:30:32 Catalog written to /mnt/c/Users/hrishi/Desktop/temp/DBT - Udemy/dbtlearn/target/catalog.json
8 (dbt_venv) hrishi@HrishiKesh:/mnt/c/Users/hrishi/Desktop/temp/DBT - Udemy/dbtlearn$ |
```

Shows the target 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.

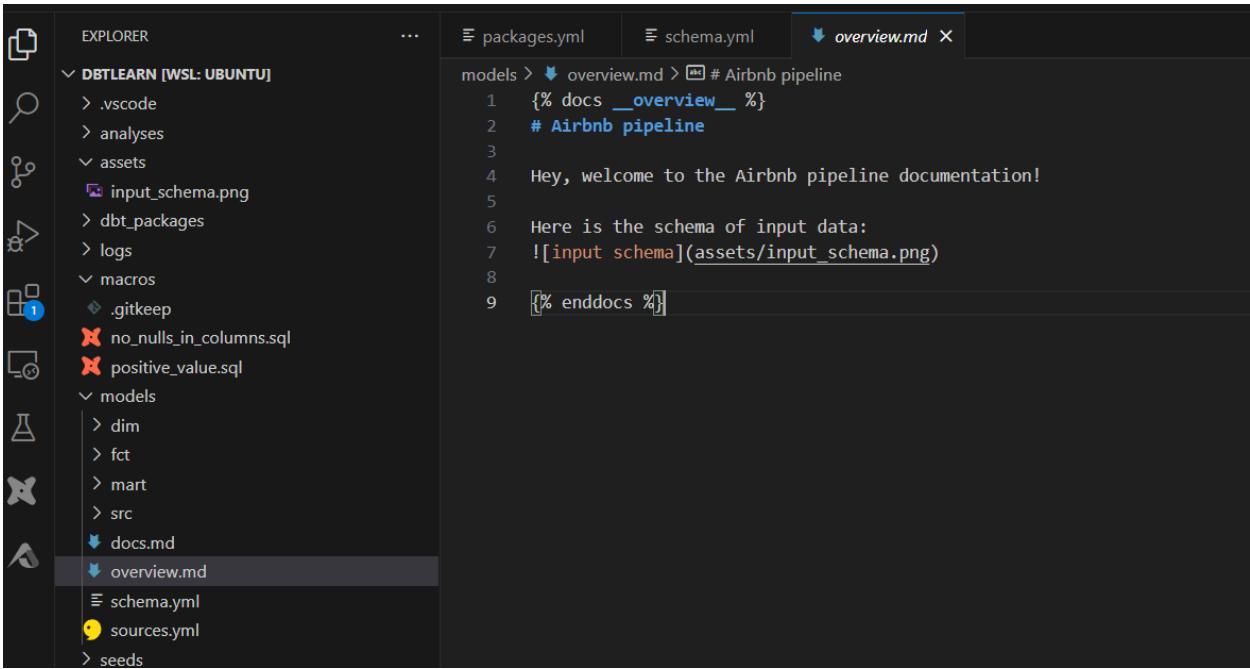
Use dbt docs serve to launch the documentation in html

```
(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 -
|
```

these are the files it generated as documentation

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

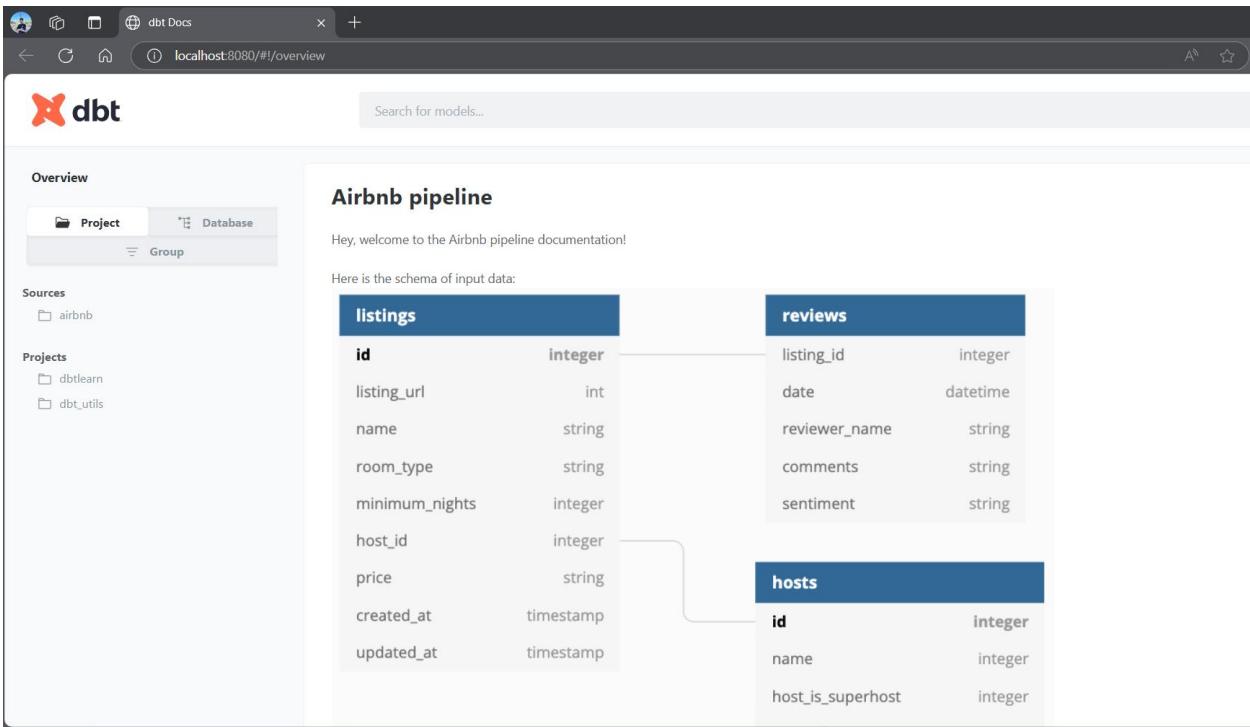


The screenshot shows the VS Code interface with the 'EXPLORER' view open. The left sidebar lists the project structure under 'DBLEARN [WSL: UBUNTU]'. The 'models' folder contains 'dim', 'fct', 'mart', 'src', 'docs.md', and 'overview.md'. The 'overview.md' file is currently selected. The right panel shows the contents of 'overview.md':

```
models > overview.md > # Airbnb pipeline
1  {% docs __overview__ %}
2  # Airbnb pipeline
3
4 Hey, welcome to the Airbnb pipeline documentation!
5
6 Here is the schema of input data:
7 ![[input schema](assets/input_schema.png)]
8
9 [% enddocs %]
```

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

This is how it looks -



The screenshot shows a web browser displaying the dbt Docs interface at localhost:8080/#/overview. The page title is 'dbt Docs'. The main content area is titled 'Airbnb pipeline' and displays the following text:

Hey, welcome to the Airbnb pipeline documentation!

Here is the schema of input data:

The schema is represented as three tables: 'listings', 'reviews', and 'hosts'. A diagram shows relationships between them:

- listings**

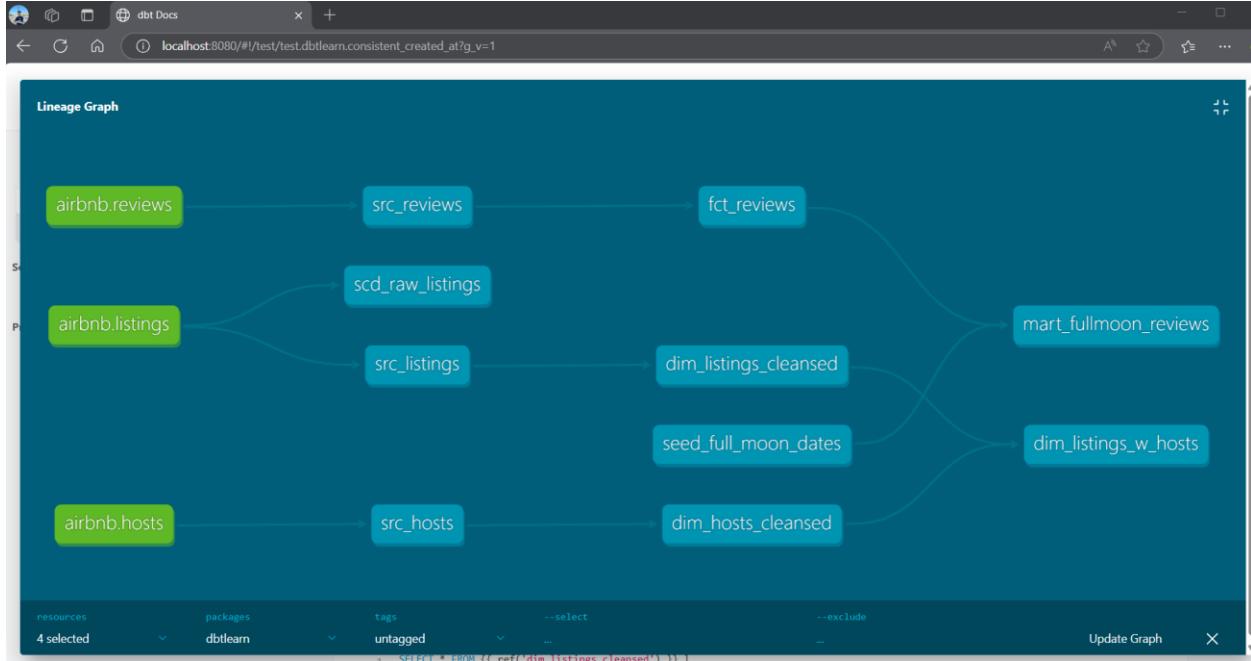
id	integer
listing_url	int
name	string
room_type	string
minimum_nights	integer
host_id	integer
price	string
created_at	timestamp
updated_at	timestamp
- reviews**

listing_id	integer
date	datetime
reviewer_name	string
comments	string
sentiment	string
- hosts**

id	integer
name	string
host_is_superhost	integer

A curved arrow points from the 'listing_id' column in the 'reviews' table to the 'host_id' column in the 'hosts' table, indicating a many-to-one relationship.

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.

The screenshot shows the dbt CLI interface with the title bar 'dbtlearn [WSL: Ubuntu]'. The left sidebar shows the project structure under 'EXPLORER': DBLEARN [WSL: UBUNTU] (with '.vscode', 'analyses', '.gitkeep', and 'full_moon_no_sleep.sql' selected), 'assets' (with 'input_schema.png'), 'logs', 'macros' (with '.gitkeep', 'no_nulls_in_columns.sql', and 'positive_value.sql'), 'models' (with 'dim', 'fct' (containing 'fct_reviews.sql'), and 'mart'). The right pane shows the contents of 'full_moon_no_sleep.sql' (highlighted in the code block below). The status bar at the bottom shows '4 selected' packages and 'dbtlearn'.

```

SELECT * FROM {{ ref('dim_listings_cleansed') }}
)
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
  
```

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/analyses/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 -

IS_FULL_MOON	REVIEW_SENTIMENT	REVIEWS
full moon	negative	1070
full moon	neutral	4903
full moon	positive	7877
not full moon	negative	28818
not full moon	neutral	142352
not full moon	positive	224678

Query Details

- Query duration: 882ms
- Rows: 6
- Query ID: 01bb14cb-0002-8f92-0...

IS_FULL_MOON

IS_FULL_MOON	Count
full moon	3
not full moon	3

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 ->

```

USE ROLE ACCOUNTADMIN;
CREATE ROLE IF NOT EXISTS REPORTER;
CREATE USER IF NOT EXISTS PRESET
    PASSWORD='presetPassword123'
    LOGIN_NAME='preset'
    MUST_CHANGE_PASSWORD=FALSE
    DEFAULT_WAREHOUSE='COMPUTE_WH'
    DEFAULT_ROLE=REPORTER
    DEFAULT_NAMESPACE='AIRBNB.DEV'
    COMMENT='Preset user for creating reports';

GRANT ROLE REPORTER TO USER PRESET;
GRANT ROLE REPORTER TO ROLE ACCOUNTADMIN;
GRANT ALL ON WAREHOUSE COMPUTE_WH TO ROLE REPORTER;
GRANT USAGE ON DATABASE AIRBNB TO ROLE REPORTER;
GRANT USAGE ON SCHEMA AIRBNB.DEV TO ROLE REPORTER;

```

Results

status
1 Statement executed successfully.

Query Details

- Query duration
- Rows
- Query ID 01bb14

select reporter role -

Databases Worksheets

AIRBNB.RAW Settings

```

USE ROLE ACCOUNTADMIN;
CREATE ROLE IF NOT EXISTS REPORTER;
CREATE USER IF NOT EXISTS PRESET
    PASSWORD='presetPassword123'
    LOGIN_NAME='preset'
    MUST_CHANGE_PASSWORD=FALSE
    DEFAULT_WAREHOUSE='COMPUTE_WH'
    DEFAULT_ROLE=REPORTER
    DEFAULT_NAMESPACE='AIRBNB.DEV'
    COMMENT='Preset user for creating reports';

GRANT ROLE REPORTER TO USER PRESET;
GRANT ROLE REPORTER TO ROLE ACCOUNTADMIN;
GRANT ALL ON WAREHOUSE COMPUTE_WH TO ROLE REPORTER;
GRANT USAGE ON DATABASE AIRBNB TO ROLE REPORTER;
GRANT USAGE ON SCHEMA AIRBNB.DEV TO ROLE REPORTER;

```

Run as role...

- REPORTER
- ACCOUNTADMIN
- ORGADMIN
- PUBLIC
- SECURITYADMIN
- SYSADMIN
- TRANSFORM
- USERADMIN

Loading Warehouses

Code View

Results

status
1 Statement executed successfully.

Query Details

- Query duration

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

```

1 USE ROLE ACCOUNTADMIN;
2 CREATE ROLE IF NOT EXISTS REPORTER;
3 CREATE USER IF NOT EXISTS PRESET
4     PASSWORD='presetPassword123'
5     LOGIN_NAME='preset'
6     MUST_CHANGE_PASSWORD=FALSE
7     DEFAULT_WAREHOUSE='COMPUTE_WH'
8     DEFAULT_ROLE=REPORTER
9     DEFAULT_NAMESPACE='AIRBNB.DEV'
10    COMMENT='Preset user for creating reports';
11
12 GRANT ROLE REPORTER TO USER PRESET;
13 GRANT ROLE REPORTER TO ROLE ACCOUNTADMIN;
14 GRANT ALL ON WAREHOUSE COMPUTE_WH TO ROLE REPORTER;
15 GRANT USAGE ON DATABASE AIRBNB TO ROLE REPORTER;
16 GRANT USAGE ON SCHEMA AIRBNB.DEV TO ROLE REPORTER;
17

```

Statement executed successfully.

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

```

# Configuring models
# Full documentation: https://docs.getdbt.com/docs/configuring\_models
# In this example config, we tell dbt to build all models in the example directory as views. These settings can be overridden in the individual files using the `{{ config(...) }}` macro.
models:
  dbtlearn:
    # Config indicated by + and applies to all files under models/example
    +materialized: view # here hashtag means i do not want a subfolder
    +post-hook:
      ... "GRANT SELECT ON {{this}} TO ROLE REPORTER"
    dim:
      +materialized: table
    src:
      +materialized: ephemeral

```

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.

```
hrishi@Hrishikesh:/mnt/c/Us x + v
review_sentiment
ORDER BY
    is_full_moon,
    review_sentiment(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$ (dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$ dbt run
03:32:57 Running with dbt=1.9.3
03:33:04 Registered adapter: snowflake=1.9.0
03:33:40 Found 8 models, 1 snapshot, 1 analysis, 16 data tests, 1 seed, 3 sources, 590 macros
03:33:40
03:33:40 Concurrency: 1 threads (target='dev')
03:33:40
03:33:50 1 of 5 START sql view model DEV.dim_hosts_cleansed ..... [RUN]
03:33:51 1 of 5 OK created sql view model DEV.dim_hosts_cleansed ..... [SUCCESS 1 in 1.03s]
03:33:51 2 of 5 START sql view model DEV.dim_listings_cleansed ..... [RUN]
03:33:52 2 of 5 OK created sql view model DEV.dim_listings_cleansed ..... [SUCCESS 1 in 0.74s]
03:33:52 3 of 5 START sql incremental model DEV.fct_reviews ..... [RUN]
03:33:56 3 of 5 OK created sql incremental model DEV.fct_reviews ..... [SUCCESS 0 in 3.90s]
03:33:56 4 of 5 START sql table model DEV.dim_listings_w_hosts ..... [RUN]
03:33:59 4 of 5 OK created sql table model DEV.dim_listings_w_hosts ..... [SUCCESS 1 in 2.64s]
03:33:59 5 of 5 START sql table model DEV.mart_fullmoon_reviews ..... [RUN]
03:34:02 5 of 5 OK created sql table model DEV.mart_fullmoon_reviews ..... [SUCCESS 1 in 3.57s]
03:34:02
03:34:03 Finished running 1 incremental model, 2 table models, 2 view models in 0 hours 0 minutes and 22.65 seconds (22.65s).
03:34:03
03:34:03 Completed successfully
03:34:03
03:34:03 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 screenshot shows the Snowflake UI interface. On the left, the sidebar displays the database structure under the AIRBNB database, including the DEV schema which contains Tables like DIM_LISTINGS_W_HOSTS, FCT_REVIEWS, and MART_FULLMOON_REVIEWS, and Views like DIM_HOSTS_CLEANSED and DIM_LISTINGS_CLEANSED. On the right, the main pane shows a SQL script for creating a user:

```
1 USE ROLE ACCOUNTADMIN;
2 CREATE ROLE IF NOT EXISTS REPORTER;
3 CREATE USER IF NOT EXISTS PRESET
4 PASSWORD='presetPassword123'
5 LOGIN_NAME='preset'
6 MUST_CHANGE_PASSWORD=FALSE
7 DEFAULT_WAREHOUSE='COMPUTE_WH'
8 DEFAULT_ROLE=REPORTER
9 DEFAULT_NAMESPACE='AIRBNB.DEV'
10 COMMENT='Preset user for creating reports';
11
12 GRANT ROLE REPORTER TO USER PRESET;
13 GRANT ROLE REPORTER TO ROLE ACCOUNTADMIN;
14 GRANT ALL ON WAREHOUSE COMPUTE_WH TO ROLE REPORTER;
15 GRANT USAGE ON DATABASE AIRBNB TO ROLE REPORTER;
16 GRANT USAGE ON SCHEMA AIRBNB.DEV TO ROLE REPORTER;
17
```

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

SETTING UP PRESET

The screenshot shows a web-based configuration interface for setting up a database preset. The top navigation bar includes tabs for 'CTE - Snowflake' and 'Preset Manager'. The main URL is 'manage.app.preset.io/app/onboarding'. A status bar at the top right indicates '10 Minu'. Below the header, the IP address '44.193.153.196 / 52.70.123.52 / 54.83.88.93' is displayed.

Database name *
AIRBNB

Username
preset

Password
.....

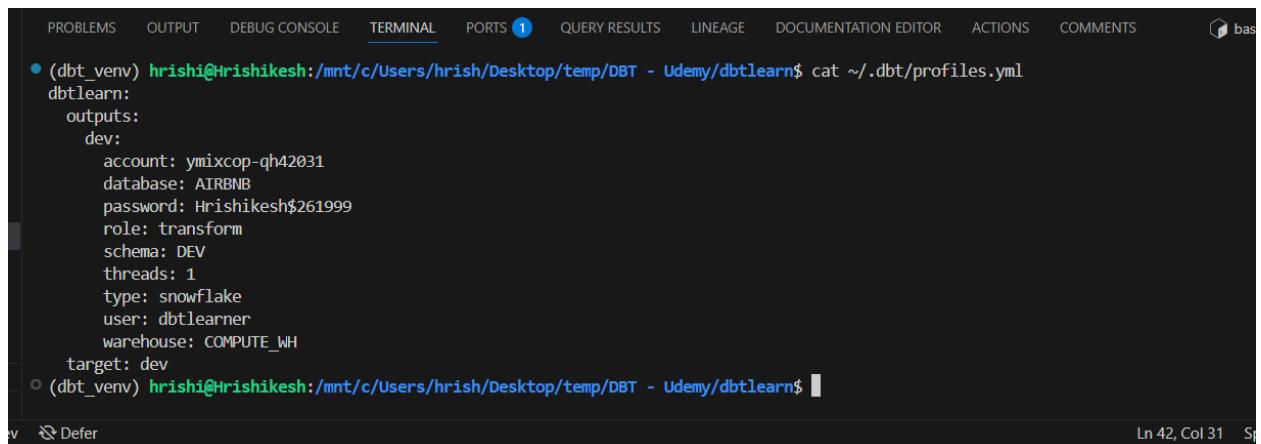
Display Name *
preset

Account *
ymixcop-qh42031

Warehouse *
COMPUTE_WH

Role
REPORTER

Connect this database with a SQLAlchemy URI instead



```

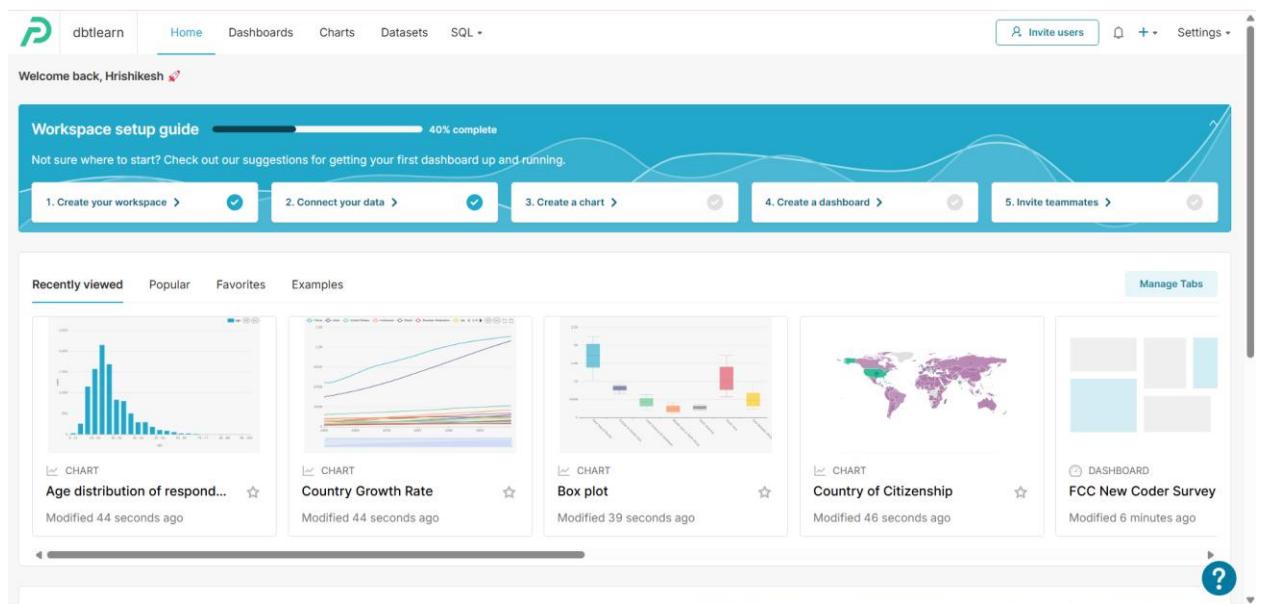
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS 1 QUERY RESULTS LINEAGE DOCUMENTATION EDITOR ACTIONS COMMENTS
● (dbt_venv) hrishi@Hrishikesh:~/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$ cat ~/.dbt/profiles.yml
dbtLearn:
  outputs:
    dev:
      account: ymixcop-qh42031
      database: AIRBNB
      password: Hrishikesh$261999
      role: transform
      schema: DEV
      threads: 1
      type: snowflake
      user: dbtlearner
      warehouse: COMPUTE_WH
  target: dev
○ (dbt_venv) hrishi@Hrishikesh:~/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$ 

```

Ln 42, Col 31 Sp

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

And this will be successful home screen display –



Welcome back, Hrishikesh 🌟

Workspace setup guide 40% complete

Not sure where to start? Check out our suggestions for getting your first dashboard up and running.

1. Create your workspace > ✓
2. Connect your data > ✓
3. Create a chart > ⓘ
4. Create a dashboard > ⓘ
5. Invite teammates > ⓘ

Recently viewed Popular Favorites Examples

- CHART Age distribution of respond... ★ Modified 44 seconds ago
- CHART Country Growth Rate ★ Modified 44 seconds ago
- CHART Box plot ★ Modified 39 seconds ago
- CHART Country of Citizenship ★ Modified 46 seconds ago
- DASHBOARD FCC New Coder Survey Modified 6 minutes ago

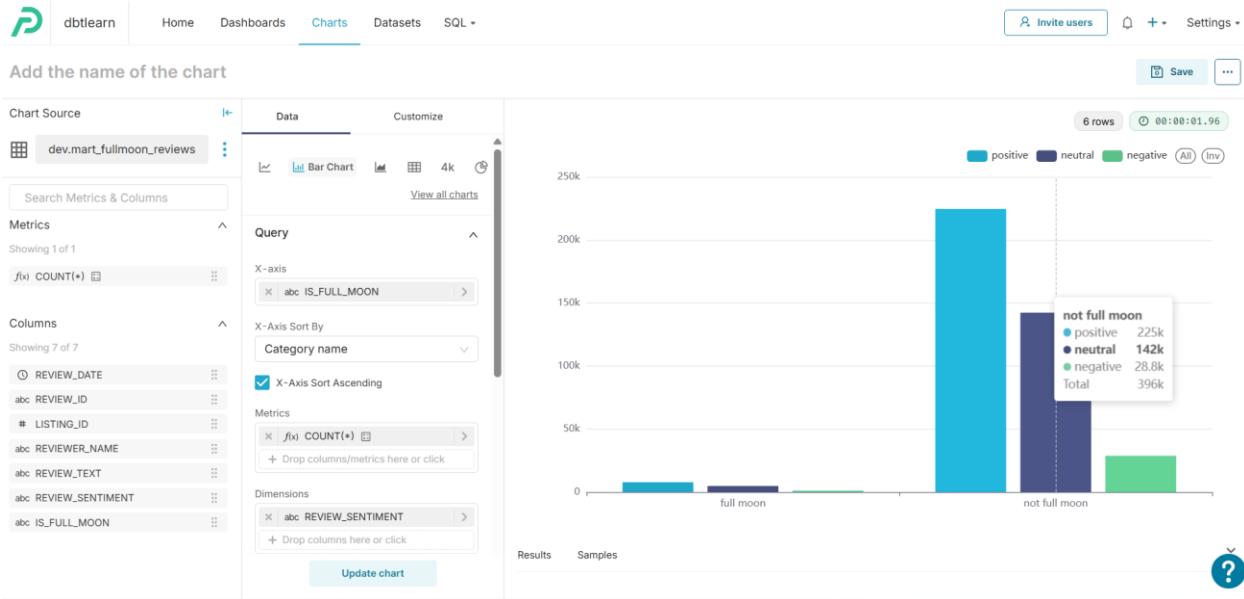
Now, connect the dataset of Airbnb which is connected through snowflake as we performed data ingestion and make a visual representation for bookings with full moon and no full moon.

Connected to the database – Airbnb

The screenshot shows the dbtlearn interface connected to a Snowflake database. The top navigation bar includes links for Home, Dashboards, Charts, Datasets, and SQL. On the right, there are buttons for Invite users, a search icon, a plus sign for creating new datasets, and Settings. The main area displays the schema for the 'mart_fullmoon_reviews' table. The schema details seven columns: review_id (VARCHAR), listing_id (DECIMAL), review_date (TIMESTAMP), reviewer_name (VARCHAR), review_text (VARCHAR), review_sentiment (VARCHAR), and is_full_moon (VARCHAR). The 'review_id' column is highlighted with a blue border.

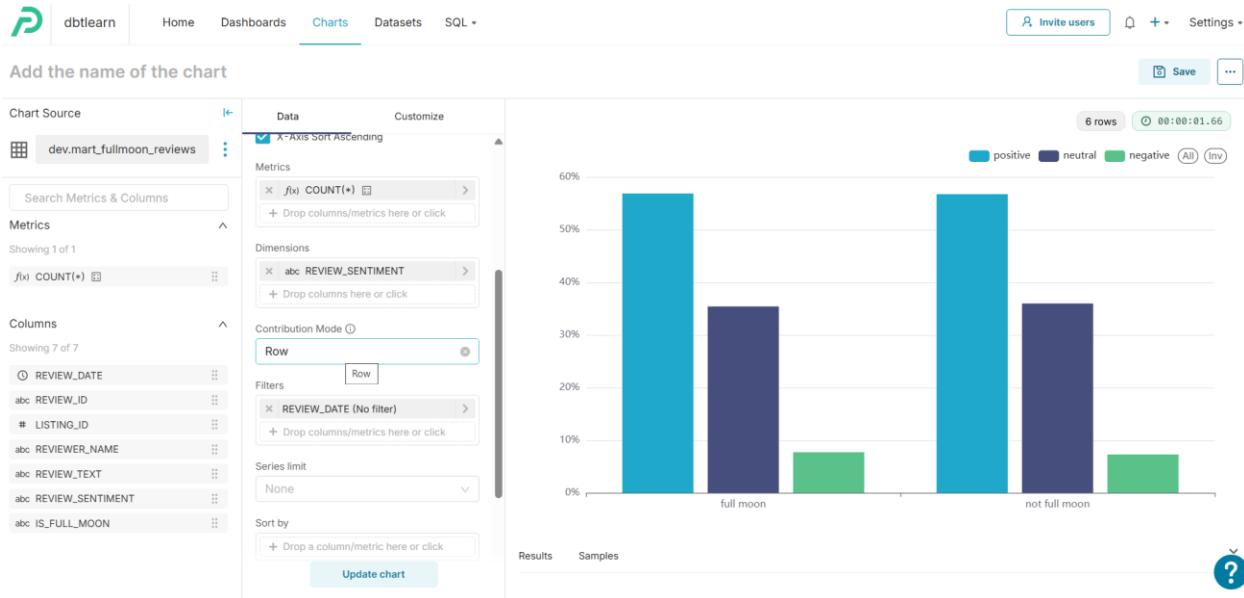
Create new chart –

The screenshot shows the 'Create a new chart' wizard in dbtlearn. Step 1, 'Choose a dataset', has a checked checkbox and shows 'mart_fullmoon_reviews' selected. A link to 'Add a dataset or view instructions' is also present. Step 2, 'Choose chart type', shows a sidebar with categories like Correlation, Distribution, Evolution, Flow, KPI, Map, Part of a Whole, Ranking, Table, and Other. The main area displays a grid of chart thumbnails, including Area Chart, Bar Chart, Big Number, Big Number with Trendline, Box Plot, Bubble Chart, Funnel Chart, Gauge Chart, Graph Chart, Heatmap, Interactive Table, Line Chart, Mixed Chart, Pie Chart, Pivot Table, and Radar Chart. A search bar at the top of the chart grid allows for searching all charts. A message at the bottom states 'Please select both a Dataset and a Chart type to proceed'. A 'Create new chart' button is located at the bottom right, and a help icon is in the bottom right corner of the main area.

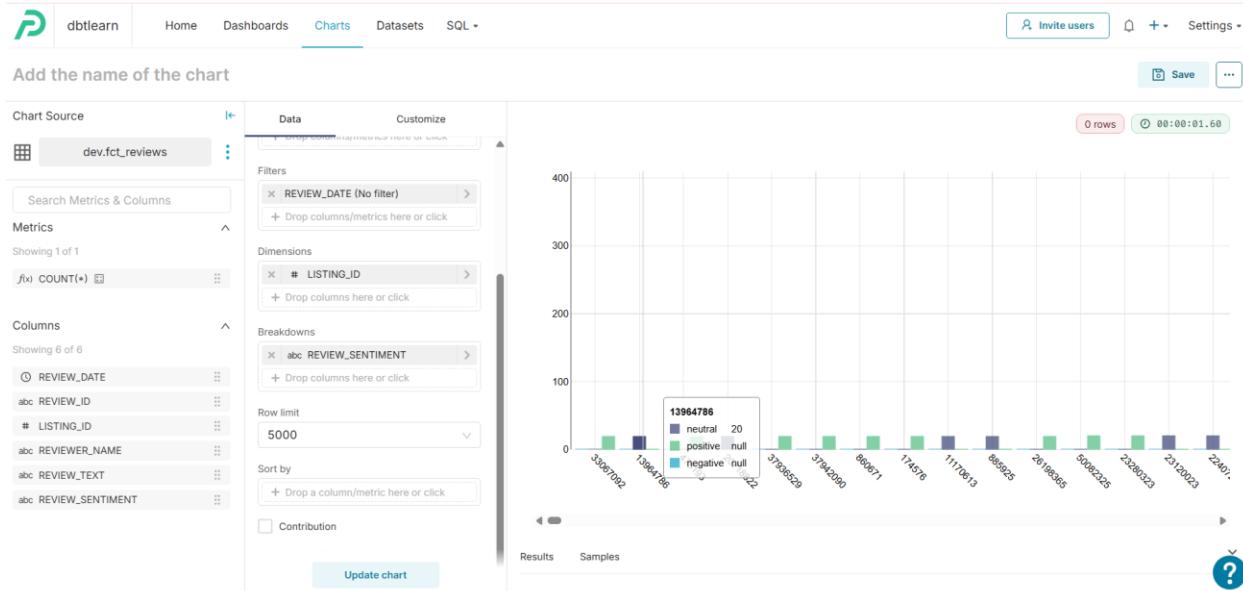


But as we can see there's more data with not full moon and it seems the dataset is quite skewed.

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 ← → dbteam [WSL: Ubuntu] dashboardsym x mart_fullmoon_reviews.sql overview.md

EXPLORER

DBTEAM [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

```

models > dashboardsyml
  1   version: 2
  2
  3   exposures:
  4     - name: airbnb_dashboard
  5       label: AIRBNB Dashboard
  6       type: dashboard
  7       maturity: low
  8       url: https://469e14e3.us2a.app.preset.io/superset/dashboard/8/?native_filters_key=qH0HzKQNifAN5gQ8kBPsfsoDMajK_SSPyXAdoJ3D1x0dt
  9       description: Executive Dashboard about Airbnb listings and hosts
 10
 11   depends_on:
 12     - ref('dim_listings_w_hosts')
 13     - ref('mart_fullmoon_reviews')
 14
 15   owner:
 16     name: hrishikesh
 17
 18
  
```

localhost:8080/#/exposure/exposure.dbteam.airbnb_dashboard

dbt

Search for models...

Overview

AIRBNB Dashboard exposure

View this exposure

Project Database Group

Sources

- airbnb

Exposures

- Dashboard
- AIRBNB Dashboard

Projects

- dbteam
- dbt_utils

Details

TAGS	PACKAGE	CONTRACT	MATURITY	OWNER	EXPOSURE NAME
untagged	dbteam	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

The lineage graph illustrates the data flow from raw sources to the final dashboard. Key nodes include:

- Raw Sources: airbnb.hosts, airbnb.listings, airbnb.reviews
- Intermediate Models: src_hosts, scd_raw_listings, src_listings, src_reviews, dim_hosts_cleaned, dim_listings_cleaned, fct_reviews, dim_listings_minimum_nights, consistent_created_at, no_nulls_in_dim_listings, seed_full_moon_dates, dim_listings_w_hosts, mart_fullmoon_reviews
- Final Model: full_moon_no_sleep
- Dashboard: AIRBNB Dashboard

Filters at the bottom of the graph interface:

- resources: All selected
- packages: dbteam
- tags: untagged
- select: ...
- exclude: ...
- Update Graph X

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.

The screenshot shows a terminal window with several tabs at the top: dashboards.yml, packages.yml (which is the active tab), mart_fullmoon_reviews.sql, and overview.md.

The contents of the packages.yml file are:

```
packages:
  - package: dbt-labs/dbt_utils
    version: 1.3.0
  - 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|
```

The terminal output shows the execution of dbt commands:

```
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrishi/Desktop/temp/DBT - Udemy/dbtlearn$ ls
 README.md assets dbt_project.yml macros package-lock.yml seeds target
 analysis dbt_packages logs models packages.yml snapshots tests
 (dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrishi/Desktop/temp/DBT - Udemy/dbtlearn$ dbt deps
 21:37:06 Running with dbt=1.9.3
 21:37:06 [WARNING]: Deprecated functionality
 The 'calogica/dbt_expectations' package is deprecated in favor of
 'metaplane/dbt_expectations'. Please update your 'packages.yml' configuration to
 use 'metaplane/dbt_expectations' instead.
 21:37:07 Updating lock file in file path: /mnt/c/Users/hrishi/Desktop/temp/DBT - Udemy/dbtlearn/package-lock.yml
 21:37:07 Installing dbt-labs/dbt_utils
 21:37:12 Installed from version 1.3.0
 21:37:12 Up to date!
 21:37:12 Installing calogica/dbt_expectations
 21:37:16 Installed from version 0.10.4
 21:37:16 Up to date!
 21:37:16 Installing calogica/dbt_date
 21:37:17 Installed from version 0.10.1
 21:37:17 Up to date!
 (dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrishi/Desktop/temp/DBT - Udemy/dbtlearn$ |
```

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

```
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)
```

The screenshot shows a code editor with multiple tabs: dashboards.yml, packages.yml, schema.yml (highlighted), sources.yml, and mart_fullmoon_r. The schema.yml file contains a models section with a dim_hosts_cleansed model. This model has columns host_id and host_name, and a tests section with a not_null test for host_id and an accepted_values test for host_name with values 't' and 'f'. Below this, there is a comment about adding documentation or tests. A new model dim_listings_w_hosts is being defined, and its tests section is being expanded. Inside the tests section, a dbt_expectations.expect_table_row_count_to_equal_other_table test is being added, with its compare_model option set to source('airbnb', 'listings').

```
models >  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_hosts
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 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_airbnb_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_listings_
[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

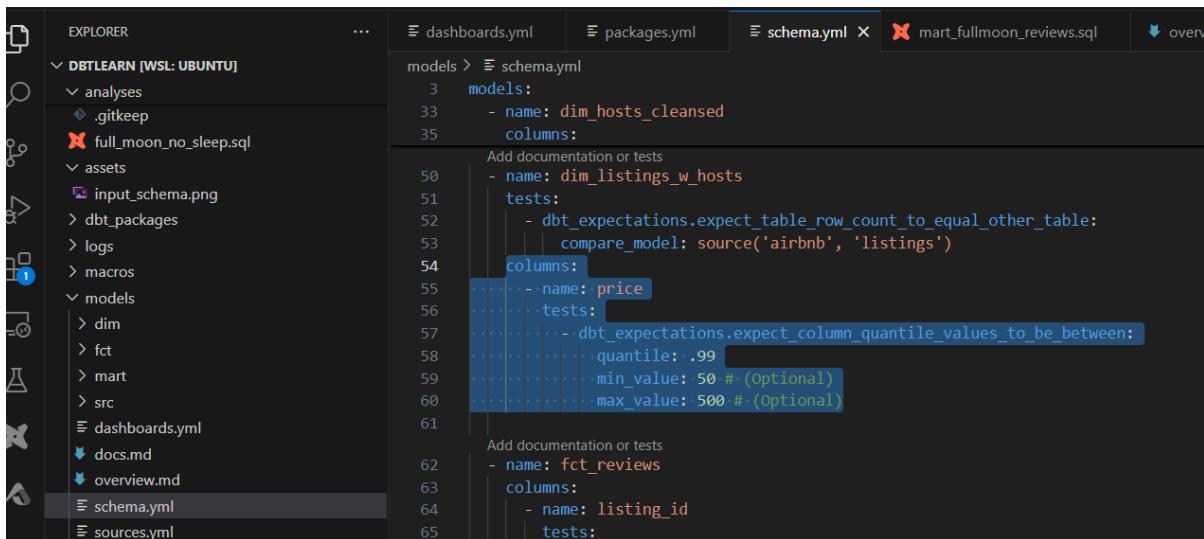
'expect_column_quantile_values_to_be_between' test. This will allow us to check the value in some range. Lets say 50-500. And since this will be applied on column – 'price' in dim_listings_w_hostings table we need to make sure we mention it as shown below with column parameter-

⌚ 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)
```



The screenshot shows a code editor interface with several tabs at the top: dashboards.yml, packages.yml, schema.yml (highlighted in blue), and mart_fullmoon_reviews.sql. The main area displays a portion of a schema.yml file under the models section. A specific section of code is highlighted with a blue selection:

```
models > models:  
  3   - name: dim_hosts_cleansed  
  35  columns:  
      Add documentation or tests  
      - name: dim_listings_w_hosts  
        tests:  
        52          - dbt_expectations.expect_table_row_count_to_equal_other_table:  
                    compare_model: source('airbnb', 'listings')  
        54          columns:  
        55            - name: price  
        56            tests:  
        57              - dbt_expectations.expect_column_quantile_values_to_be_between:  
                quantile: .99  
                min_value: 50 # (Optional)  
                max_value: 500 # (Optional)
```

The code highlights the 'expect_column_quantile_values_to_be_between' test configuration for the 'price' column, specifying a quantile of 0.99 and a range from 50 to 500.

```
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrishi/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  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/hrishi/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)
```

The screenshot shows the DBT IDE interface. On the left, the Explorer panel displays the project structure under 'DBLEARN [WSL: UBUNTU]'. It includes sections for analyses, assets, dbt_packages, logs, macros, models (with subfolders dim, fct, mart, src), dashboards.yml, and docs.md. The main area shows the contents of 'schema.yml'. The code editor highlights a section of the 'models' block for the 'dim_listings_w_hosts' model:

```
models:
  - name: dim_listings_w_hosts
    tests:
      - dbt_expectations.expect_table_row_count_to_equal_other_table:
          compare_model: source('airbnb', 'listings')
    columns:
      - name: price
        tests:
          - dbt_expectations.expect_column_quantile_values_to_be_between:
              quantile: .99
              min_value: 50 # (Optional)
              max_value: 500 # (Optional)
      - dbt_expectations.expect_column_max_to_be_between:
          #: min_value: 1 # (Optional)
          max_value: 5000 # (Optional)
```

```
(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  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 [PASS
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_listings_
[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    compiled code at target/compiled/dbtlearn/models/schema.yml/dbt_expectations_expect_column_c59e300e0dddb335c4211147100
aclc6.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.

The screenshot shows the Snowflake interface. On the left, the database structure is visible under the AIRBNB schema, specifically the DEV database. The DIM_LISTINGS_W_HOSTS table is selected. On the right, a query is run:

```
1 | select max(price) from airbnb.dev.dim_listings_w_hosts
```

The results show a single row with the value 8000.00.

#	MAX(PRICE)
1	8000.00

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.

```
00:49:24  Done. PASS=2 WARN=0 ERROR=1 SKIP=0 TOTAL=3
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$ dbt test --select dim_listings_w_hosts
00:49:59  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 [PASS
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_listings_
[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    compiled code at target/compiled/dbtlearn/models/schema.yml/dbt_expectations_expect_column_c59e300e0dddb335c4211147100
aclc6.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.

```
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')  
  columns:  
    - name: price  
      tests:  
        - dbt_expectations.expect_column_values_to_be_of_type:  
          column_type: number  
        - dbt_expectations.expect_column_quantile_values_to_be_between:  
          quantile: .99  
          min_value: 50 # (optional)  
          max_value: 500 # (optional)  
        - dbt_expectations.expect_column_max_to_be_between:  
          # min_value: 1 # (optional)  
          max_value: 5000 # (optional)  
      config:  
        severity: warn
```



```
00:53:53 2 of 4 PASS dbt_expectations_expect_column_quantile_values_to_be_between_dim_listings_w_hosts_price_500_5000 [PASS] [RUN]  
S in 0.12s]  
00:53:53 3 of 4 START test dbt_expectations_expect_column_values_to_be_of_type_dim_listings_w_hosts_price_number [RUN]  
00:53:53 3 of 4 PASS dbt_expectations_expect_column_values_to_be_of_type_dim_listings_w_hosts_price_number [PASS in 0.44s] [RUN]  
00:53:53 4 of 4 START test dbt_expectations_expect_table_row_count_to_equal_other_table_dim_listings_w_hosts_source_airbnb_listin  
gs_ [RUN]
```

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.

```

version: 2
sources:
  - name: airbnb
    schema: raw
    tables:
      - name: listings
        identifier: raw_listings
        columns:
          - name: room_type
        tests:
          - dbt_expectations.expect_column_distinct_count_to_equal:
              value: 4
      - name: hosts
        identifier: raw_hosts

```

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]+"

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)

```

```

EXPLORER
...
DBT - UDEMY [WSL: UBUNTU]
  dbtlearn
    models
      dim
        dim_hosts_cleansed.sql
        dim_listings_cleansed.sql
        dim_listings_w_hosts.sql
      fct
      mart
      src
        src_hosts.sql
        src_listings.sql
        src_reviews.sql
      dashboards.yml
      docs.md
      overview.md
      schema.yml
      sources.yml

sources.yml X
dbtlearn > models > sources.yml
  version: 2
  ...
  sources:
    - name: airbnb
      schema: raw
      tables:
        Generate model
        - name: listings
          identifier: raw_listings
          columns:
            - name: room_type
          tests:
            - dbt_expectations.expect_column_distinct_count_to_equal:
                value: 4
        - name: price
          tests:
            - dbt_expectations.expect_column_values_to_match_regex:
                regex: "^\$\\"[0-9][0-9\\.]+\$"
  ...
  Generate model

```

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/hrishi/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 [PAS
S 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_ [FATAL 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/hrishi/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`

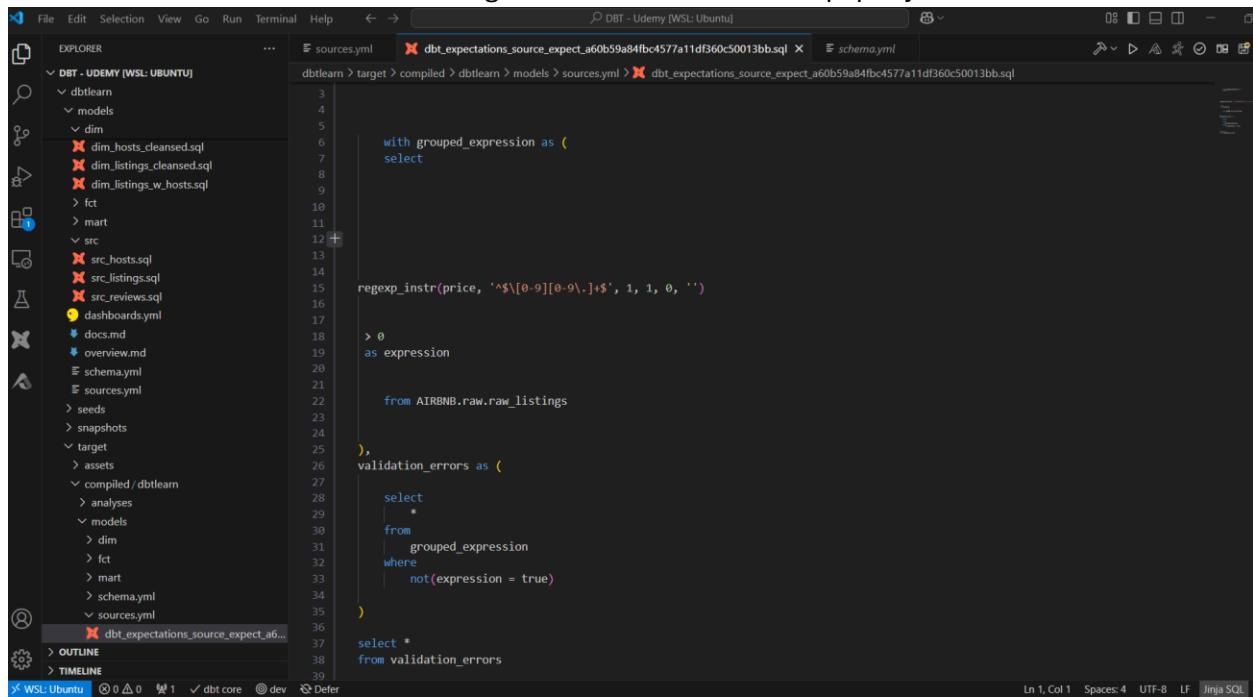
So this will return a big debug screen but what you need to look is for the specific test being performed for which the error occurred and its sql file -

```
hrishi@Hrishikesh:/mnt/c/Us X + ▾
)
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/hrishi/Desktop/temp/DBT - Udemy/dbtlearn/target/manifest.json
21:43:43 Wrote artifact SemanticManifest to /mnt/c/Users/hrishi/Desktop/temp/DBT - Udemy/dbtlearn/target/semantic_manifest.json
21:43:43 Wrote artifact RunExecutionResult to /mnt/c/Users/hrishi/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': [

```

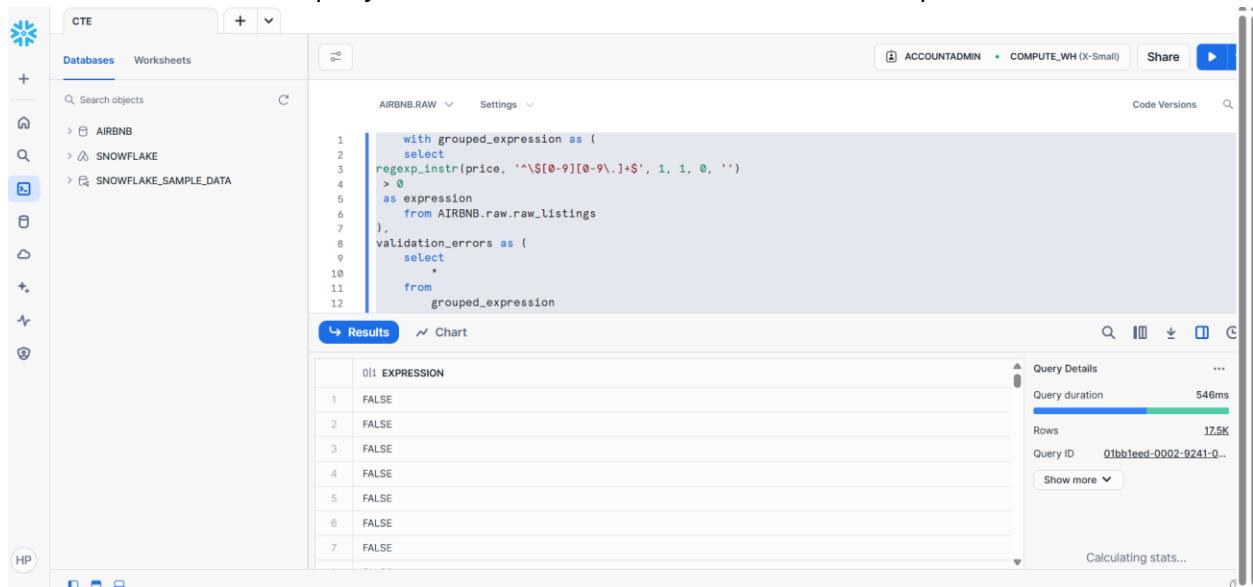
Used code command and then the target folder link to view the sql query in code editor -



The screenshot shows a code editor interface with the title "DBT - Udemy [WSL: Ubuntu]". The left sidebar displays the project structure:

- DBT - UDEMY [WSL: UBUNTU]
 - dbteam
 - models
 - dim
 - dim_hosts_cleaned.sql
 - dim_listings_cleaned.sql
 - dim_listings_w_hosts.sql
 - fct
 - mart
 - src
 - src_hosts.sql
 - src_listings.sql
 - src_reviews.sql
 - dashboards.yml
 - docs.md
 - overview.md
 - schema.yml
 - sources.yml
 - dbt_expectations_source_expect_a6...
 - target
 - assets
 - compiled/dbteam
 - analyses
 - models
 - dim
 - fct
 - mart
 - schema.yml
 - sources.yml
 - dbt_expectations_source_expect_a6...
- OUTLINE
- TIMELINE

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



The screenshot shows a data warehouse interface with the following details:

- Left sidebar: Databases (AIRBNB, SNOWFLAKE, SNOWFLAKE_SAMPLE_DATA) and Worksheets.
- Top right: ACCOUNTADMIN, COMPUTE_WH (X-Small), Share, and a blue play button.
- Middle section: A code editor window titled "AIRBNB.RAW" with the following SQL query:

```
1   with grouped_expression as (
2     select
3       regexp_instr(price, '^$|[0-9][0-9\\.]+$', 1, 1, 0, '')
4     > 0
5     as expression
6     from AIRBNB.raw.raw_listings
7   ),
8   validation_errors as (
9     select
10    *
11    from grouped_expression
12  ),
```

- Bottom section: Results table titled "0|1 EXPRESSION" showing the output:

	0 1 EXPRESSION
1	FALSE
2	FALSE
3	FALSE
4	FALSE
5	FALSE
6	FALSE
7	FALSE

- Right side panel: Query Details showing Query duration (546ms), Rows (17.5K), and Query ID (01bb1eed-0002-9241-0...).

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.

```

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

```

	PRICE	0 1 EXPRESSION
1	\$90.00	FALSE
2	\$33.00	FALSE
3	\$180.00	FALSE
4	\$70.00	FALSE
5	\$90.00	FALSE
6	\$47.00	FALSE
7	\$169.00	FALSE
8	\$70.00	FALSE
9	\$65.00	FALSE

Query Details
Query duration 456ms
Rows 17.5K
Query ID 01bb1eee-0002-91e7-0...
Show more
Calculating stats...

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.

```

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

```

	PRICE	0 1 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 backslashes –

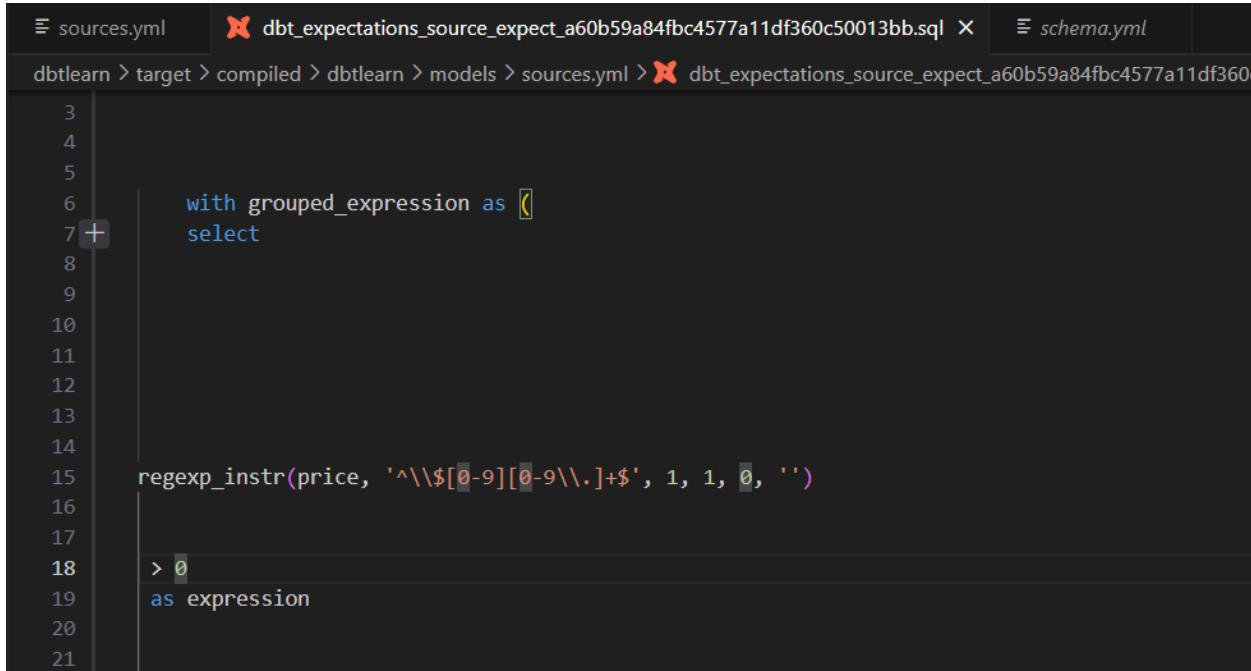
```
    value: 4
  - name: price
    tests:
      - dbt_expectations.expect_column_values_to_match_regex:
          regex: "^\$\d{0-9}\d{0-9}.\d+$"

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/hrishi/Desktop/temp/DBT - Udemy/dbtlearn$ dbt test --select source:airbnb.listings
22:12:51 Running with dbt=1.9.0
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 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/hrishi/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_a60b59a84fbc4577a11df360c50013bb.sql

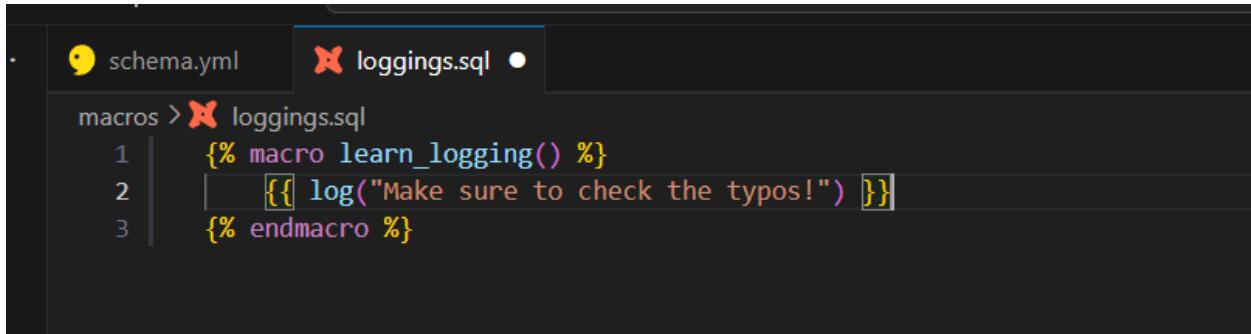
3
4
5
6     with grouped_expression as [ ]
7     select
8
9
10
11
12
13
14
15     regexp_instr(price, '^$\d{0-9}\d{0-9}.\d+$', 1, 1, 0, '')
16
17
18     > 0
19     as expression
20
21
```

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 -



```
schema.yml loggings.sql
macros > loggings.sql
1   {% macro learn_logging() %}
2     {{ log("Make sure to check the typos!") }}
3   {% endmacro %}
```

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.

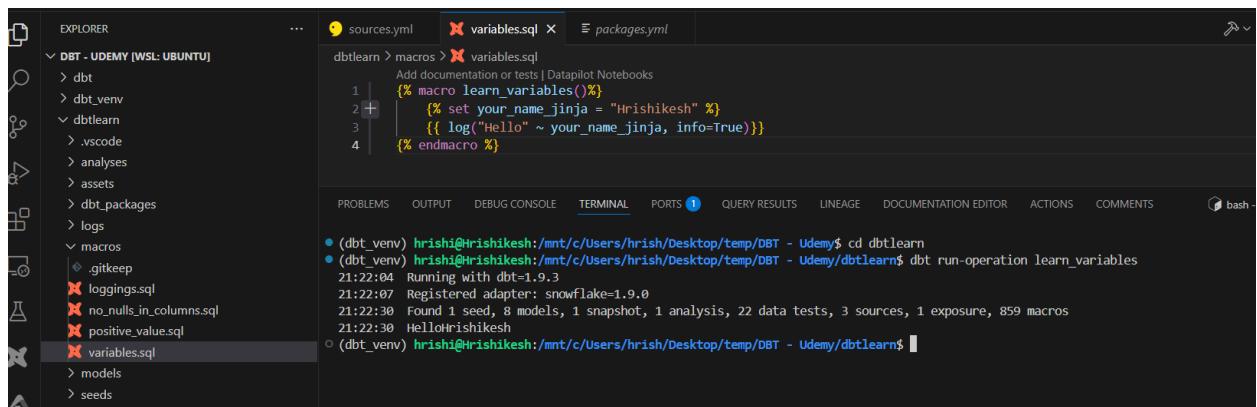
This will execute the message in the logs folder.

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.-



```
sources.yml variables.sql packages.yml
dbtlearn > macros > variables.sql
Add documentation or tests | Datapilot Notebooks
1   {% macro learn_variables() %}
2     {% set your_name_jinja = "Hrishikesh" %}
3     {{ log("Hello" ~ your_name_jinja, info=True) }}
4   {% endmacro %}
```

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

```
(dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy$ cd dbtlearn
● (dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy$ dbt run-operation learn_variables
21:22:04 Running with dbt=1.9.3
21:22:07 Registered adapter: snowflake=1.9.0
21:22:30 Found 1 seed, 8 models, 1 snapshot, 1 analysis, 22 data tests, 3 sources, 1 exposure, 859 macros
21:22:30 HelloHrishikesh
○ (dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy$
```

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 variables, and are managed by dbt. Can be passed through command lines as and when needed.

The screenshot shows a terminal window with several tabs open. The current tab is 'variables.sql'. The code in the editor is:

```
{% macro learn_variables() %}  
    {% set your_name_jinja = "Hrishikesh" %}  
    {{ log("Hello " ~ your_name_jinja, info=True) }}  
  
    {{ log("Hello dbt user " ~ var("user_name") ~ "!", info=True) }}  
{% endmacro %}
```

The terminal output shows the execution of the dbt command:

```
(dbt_venv) hrishi@hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy$ cd dbtlearn  
(dbt_venv) hrishi@hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$ dbt run-operation learn_variables  
21:22:04 Running with dbt=1.9.3  
21:22:07 Registered adapter: snowflake=1.9.0  
21:22:30 Found 1 seed, 8 models, 1 snapshot, 1 analysis, 22 data tests, 3 sources, 1 exposure, 859 macros  
21:22:30 Hello@hrishikesh  
(dbt_venv) hrishi@hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$ dbt run-operation learn_variables --vars '{user_name: Hrishikesh}'  
21:39:12 Running with dbt=1.9.3  
21:39:16 Registered adapter: snowflake=1.9.0  
21:39:28 Unable to do partial parsing because config vars, config profile, or config target have changed  
21:39:37 Found 8 models, 1 snapshot, 1 analysis, 22 data tests, 1 seed, 3 sources, 1 exposure, 859 macros  
21:39:37 Hello@hrishikesh  
(dbt_venv) hrishi@hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$ dbt run-operation learn_variables --vars '{user_name: Hrishikesh}'  
21:40:24 Running with dbt=1.9.3  
21:40:27 Registered adapter: snowflake=1.9.0  
21:40:46 Found 8 models, 1 snapshot, 1 analysis, 22 data tests, 1 seed, 3 sources, 1 exposure, 859 macros  
21:40:46 Hello@hrishikesh  
21:40:46 Hello dbt user Hrishikesh!!!  
(dbt_venv) hrishi@hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$
```

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 –

```
{% macro learn_variables()%}
    {% set your_name_jinja = "Hrishikesh" %}
    {{ log("Hello " ~ your_name_jinja, info=True) }}

    {{ log("Hello dbt user " ~var("user_name", "NO USERNAME IS SET!!!!") ~ "!", info=True) }}
{% endmacro %}
```

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 –

The screenshot shows a code editor with three tabs: sources.yml, variables.sql, and dbt_project.yml. The dbt_project.yml tab is active, displaying the following configuration:

```
dbtlearn > dbt_project.yml
31 # directory as views. These settings can be overridden in the individual model
32 # files using the `{{ config(...) }}` macro.
33 models:
34   dbtlearn:
35     # Config indicated by + and applies to all files under models/example/
36     +materialized: view # here hastag means i do not want a subfolder named materialize
37     +post-hook:
38       - "GRANT SELECT ON {{this}} TO ROLE REPORTER"
39     dim:
40       +materialized: table
41     src:
42       +materialized: ephemeral
43
44 vars:
45   user_name: default_value_to_be_decided
```

Below the code editor is a terminal window showing the execution of dbt commands:

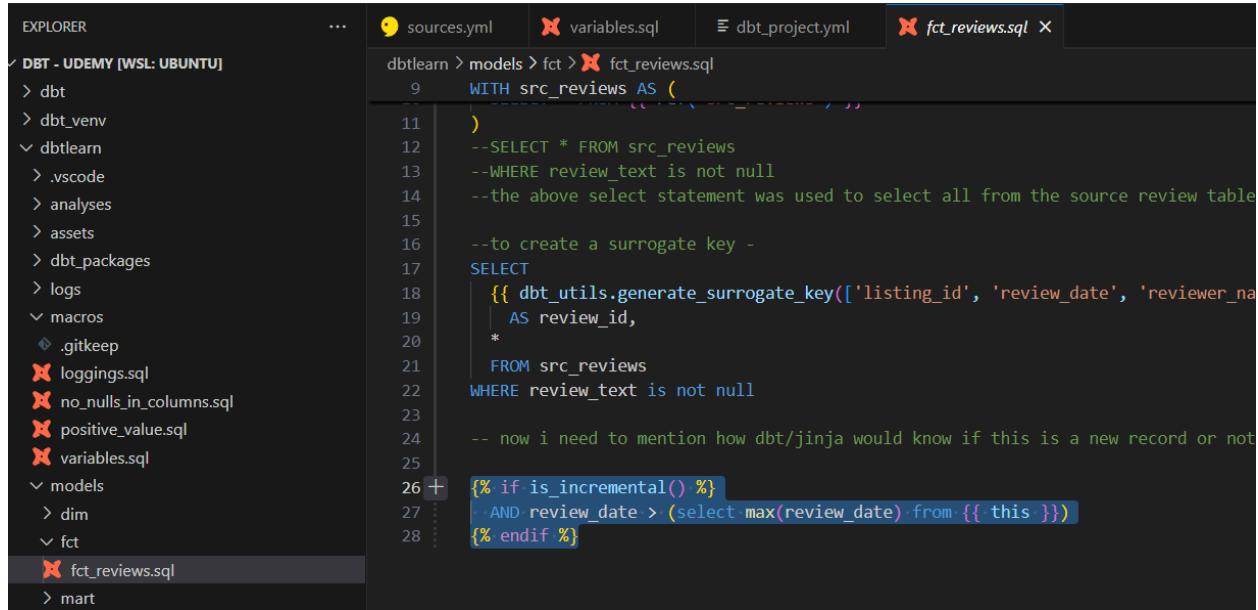
- (dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemydbtlearn\$ dbt run-operation
- 21:46:02 Running with dbt=1.9.3
- 21:46:07 Registered adapter: snowflake=1.9.0
- 21:46:19 Unable to do partial parsing because config vars, config profile, or config target have ch
- 21:46:28 Found 8 models, 1 snapshot, 1 analysis, 22 data tests, 1 seed, 3 sources, 1 exposure, 859
- 21:46:28 Hello Hrishikesh
- 21:46:28 Hello dbt user NO USERNAME IS SET!!!!
- (dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemydbtlearn\$ dbt run-operation
- 21:51:31 Running with dbt=1.9.3
- 21:51:35 Registered adapter: snowflake=1.9.0
- 21:51:47 Unable to do partial parsing because a project config has changed
- 21:51:57 Found 8 models, 1 snapshot, 1 analysis, 22 data tests, 1 seed, 3 sources, 1 exposure, 859
- 21:51:57 Hello Hrishikesh
- 21:51:57 Hello dbt user default_value_to_be_decided!
- (dbt_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemydbtlearn\$

So, now what happened here?

Whenever you execute dbt, it will first pick all the variable name mentioned in the dbt_project.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.



The screenshot shows a code editor interface with the following details:

- EXPLORER** sidebar:
 - DBT - UDEMY [WSL: UBUNTU]
 - > dbt
 - > dbt_venv
 - dbtlearn
 - > .vscode
 - > analyses
 - > assets
 - > dbt_packages
 - > logs
 - macros
 - .gitkeep
 - loggings.sql
 - no_nulls_in_columns.sql
 - positive_value.sql
 - variables.sql
 - models
 - > dim
 - > fct
 - fct_reviews.sql
 - > mart

- sources.yml**, **variables.sql**, **dbt_project.yml** tabs are visible above the code editor.
- fct_reviews.sql** tab is active.
- Code Content:**

```
dbtlearn > models > fct > fct_reviews.sql
  9   WITH src_reviews AS (
 10     )
 11   --SELECT * FROM src_reviews
 12   --WHERE review_text is not null
 13   --the above select statement was used to select all from the source review table
 14
 15   --to create a surrogate key -
 16   SELECT
 17     {{ dbt_utils.generate_surrogate_key(['listing_id', 'review_date', 'reviewer_name']) }} AS review_id,
 18     *
 19   FROM src_reviews
 20   WHERE review_text is not null
 21
 22   -- now i need to mention how dbt/jinja would know if this is a new record or not
 23
 24   {% if is_incremental() %}
 25     AND review_date > (select max(review_date) from {{ this }})
 26   {% endif %}
```

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.

The screenshot shows a code editor interface with a sidebar labeled 'EXPLORER' containing a tree view of a 'DBT - UDEMY [WSL: UBUNTU]' project. The tree includes nodes for dbt, dbt_venv, dbtlearn, .vscode, analyses, assets, dbt_packages, logs, macros, .gitkeep, loggings.sql, no_nulls_in_columns.sql, no_value.sql, variables.sql, models, dim, fct, and fct_reviews.sql. Below these are dashboards.yml, docs.md, and overview.md. The main panel displays a SQL script named 'fct_reviews.sql' with syntax highlighting for Jinja templating. The script starts with a comment about creating a surrogate key and then defines a SELECT statement that handles incremental loading based on start_date and end_date variables. It includes logic to filter rows where review_date is greater than or equal to start_date and less than end_date, or falls back to a MAX(review_date) query if no vars are provided.

```

15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35

```

```

--to create a surrogate key -
SELECT
  {{ dbt_utils.generate_surrogate_key(['listing_id', 'review_date', 'reviewer_name', 'review_text']) }}
  *
  FROM src_reviews
  WHERE review_text is not null
-- now i need to mention how dbt/jinja would know if this is a new record or not.
{% if is_incremental() %}
  {% if var("start_date", False) and var("end_date", False) %}
    {{ log('Loading ' ~ this ~ ' incrementally (start_date: ' ~ var("start_date") ~ ', end_date: ' ~ var("end_date") ~ ')', info=True) }}
    AND review_date >= '{{ var("start_date") }}'
    AND review_date < '{{ var("end_date") }}'
  {% else %}
    AND review_date > (select max(review_date) from {{ this }})
    {{ log('Loading ' ~ this ~ ' incrementally (all missing dates)', info=True) }}
  {% endif %}
  {% endif %}
  {% endif %}

```

- 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_venv) hrishi@HrishiKesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$ dbt run --select fct_reviews --vars '{start_date: "2024-02-15 00:00:00", end_date: "2024-03-15 23:59:59"}'
22:24:25  Running with dbt=1.9.3
22:24:29  Registered adapter: snowflake=1.9.0
22:24:42  Unable to do partial parsing because config vars, config profile, or config target have changed
22:24:54  Found 8 models, 1 snapshot, 1 analysis, 22 data tests, 1 seed, 3 sources, 1 exposure, 859 macros
22:24:54
22:24:54  Concurrency: 1 threads (target='dev')
22:24:54
22:24:59  1 of 1 START sql incremental model DEV.fct_reviews ..... [RUN]
22:24:59  Loading AIRBNB.DEV.fct_reviews incrementally (start_date: 2024-02-15 00:00:00, end_date: 2024-03-15 23:59:59)
22:25:02  1 of 1 OK created sql incremental model DEV.fct_reviews ..... [SUCCESS 0 in 2.79s]
22:25:02
22:25:02  Finished running 1 incremental model in 0 hours 0 minutes and 7.70 seconds (7.70s).
22:25:02
22:25:02  Completed successfully
22:25:02
22:25:02  Done. PASS=1 WARN=0 ERROR=0 SKIP=0 TOTAL=1
(dbt_venv) hrishi@HrishiKesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$ |
```

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.

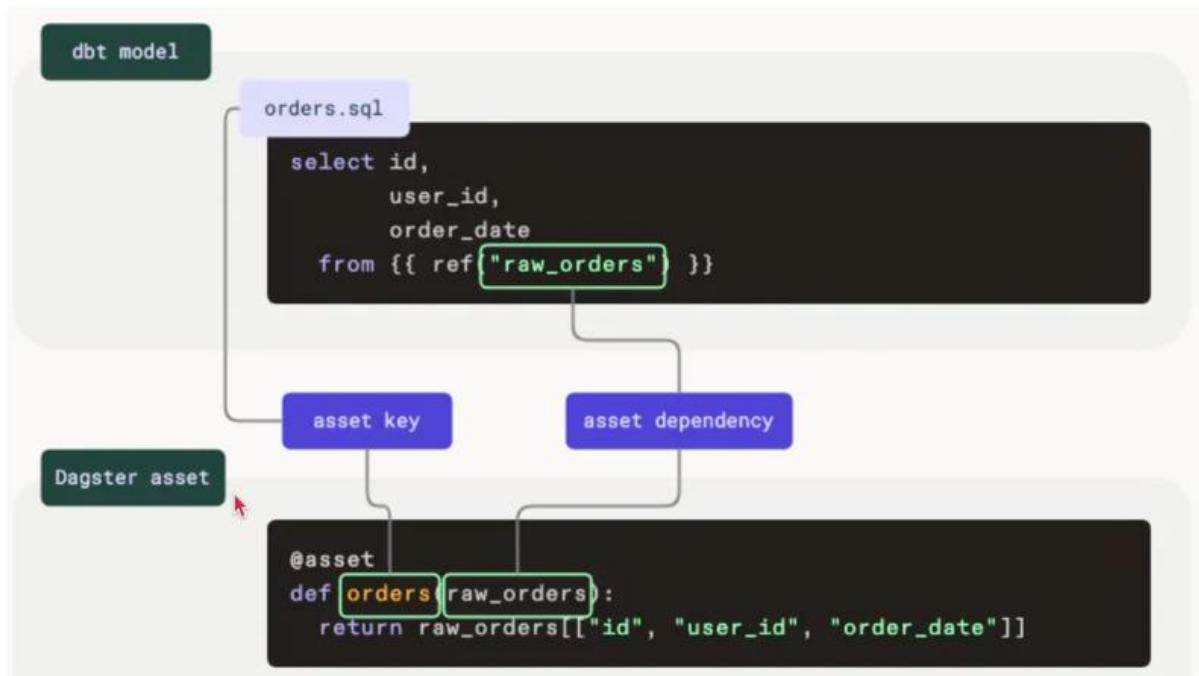
ORCHESTRATION

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.

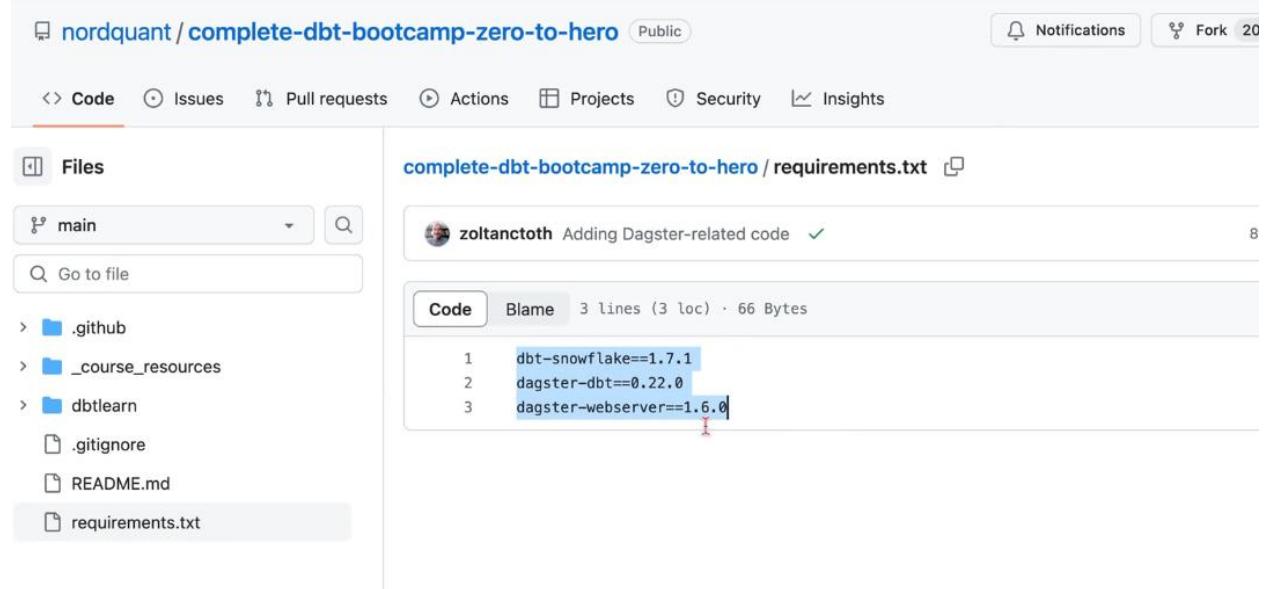
In short, Dagster's design and features complement dbt's strengths, making it easier to orchestrate, monitor, and manage your dbt models within a broader data pipeline.



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 –



The screenshot shows a GitHub repository page for "nordquant / complete-dbt-bootcamp-zero-to-hero". The "Files" tab is selected, showing the contents of the "requirements.txt" file. The file contains the following code:

```
dbt-snowflake==1.7.1
dagster-dbt==0.22.0
dagster-webserver==1.6.0
```

Below the file listing, a terminal window displays the command-line output of running pip install -r requirements.txt. The output shows the download progress for each dependency:

```
(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.

```
hrishi@Hrishikesh: /mnt/c/Us + - x
drwxrwxrwx 1 hrishi hrishi 4096 Mar 21 16:08 macros
drwxrwxrwx 1 hrishi hrishi 4096 Mar 18 16:08 models
-rw-rw-rwx 1 hrishi hrishi 223 Mar 18 16:37 package-lock.yml
-rw-rw-rwx 1 hrishi hrishi 236 Mar 18 16:35 packages.yml
drwxrwxrwx 1 hrishi hrishi 4096 Mar 11 15:50 seeds
drwxrwxrwx 1 hrishi hrishi 4096 Mar 11 17:31 snapshots
drwxrwxrwx 1 hrishi hrishi 4096 Mar 18 16:11 targets
drwxrwxrwx 1 hrishi hrishi 4096 Mar 14 19:20 tests
(dagster_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$ dagster-dbt project scaffold --project-name dbt_airbnb_dagster_project
Usage: dagster-dbt project scaffold [OPTIONS]
Try 'dagster-dbt project scaffold -h' for help.
Error
Invalid value: The project name 'dbt_airbnb_dagster_project' conflicts with an existing PyPI package. Conflicting package names will cause import errors in your project if the existing PyPI package is included as a dependency in your scaffolded project. Please choose another name, or add the '--ignore-package-conflict' flag to bypass this check.

(dagster_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$ dagster-dbt project scaffold --project-name dbt_airbnb_dagster_pipeline --ignore-package-conflict
Running with dagster-dbt version: 0.22.0.
Initializing Dagster project dbt_airbnb_dagster_pipeline in the current working directory for dbt project directory /mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn
Using profiles.yml found in /home/hrishi/.dbt/profiles.yml.
Your Dagster project has been initialized. To view your dbt project in Dagster, run the following commands:

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

(dagster_venv) hrishi@Hrishikesh:/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dbtlearn$ |
```

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.

The screenshot shows a code editor interface with several tabs at the top: 'sources.yml', 'variables.sql', 'dbt_project.yml 1', 'fct_reviews.sql', and 'schedules.py 1'. The 'schedules.py' tab is active, displaying Python code for setting up a daily dbt schedule. Below the tabs, the file content is shown:

```
1 """
2 To add a daily schedule that materializes your dbt assets, uncomment the following lines.
3 """
4 from dagster_dbt import build_schedule_from_dbt_selection
5
6 from .assets import dbtlearn_dbt_assets
7
8 schedules = [
9     build_schedule_from_dbt_selection(
10         [dbtlearn_dbt_assets],
11         job_name="materialize_dbt_models",
12         cron_schedule="0 * * * *",
13         dbt_select="fqn:*",
14     ),
15 ]
```

By uncommenting the schedules we have enabled schedules in dagster.

Now, its time to launch dagster.

```
(dagster_venv) hrishi@Hrishikesh:/mnt/c/Users/hrishi/Desktop/temp/DBT - Udemy$ ls
'dBT_course.docx'  dbt  dbtlearn  requirements.txt  '~WRL2269.tmp'
dagster_venv  dbt  dbtlearn  requirements.txt  '~WRL2269.tmp'
(dagster_venv) hrishi@Hrishikesh:/mnt/c/Users/hrishi/Desktop/temp/DBT - Udemy$ cd dbtlearn
(dagster_venv) hrishi@Hrishikesh:/mnt/c/Users/hrishi/Desktop/temp/DBT - Udemy$ cd dbtlearn
ls
README.md  assets  dbt_docker  logs  models  packages.yml  snapshot  tests
dbt  dbt_airbnb_dagster_pipeline  dbt_project.yml  parcs  package-lock.json  tests  yaml
(dagster_venv) hrishi@Hrishikesh:/mnt/c/Users/hrishi/Desktop/temp/DBT - Udemy$ cd dbt_airbnb_dagster_pipeline
(dagster_venv) hrishi@Hrishikesh:/mnt/c/Users/hrishi/Desktop/temp/DBT - Udemy$ cd dbt_airbnb_dagster_pipeline$ ls
pyproject.toml  setup.py
(dagster_venv) hrishi@Hrishikesh:/mnt/c/Users/hrishi/Desktop/temp/DBT - Udemy$ DAGSTER_DBT_PARSE_PROJECT_ON_LOAD=1 dagster dev
2025-03-23 14:35:36 -0500 - dagster - INFO - Using temporary directory /mnt/c/Users/hrishi/Desktop/temp/DBT - Udemy/dbtlearn/dbt_airbnb_dagster_pipeline/tmpf9op8g51 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:

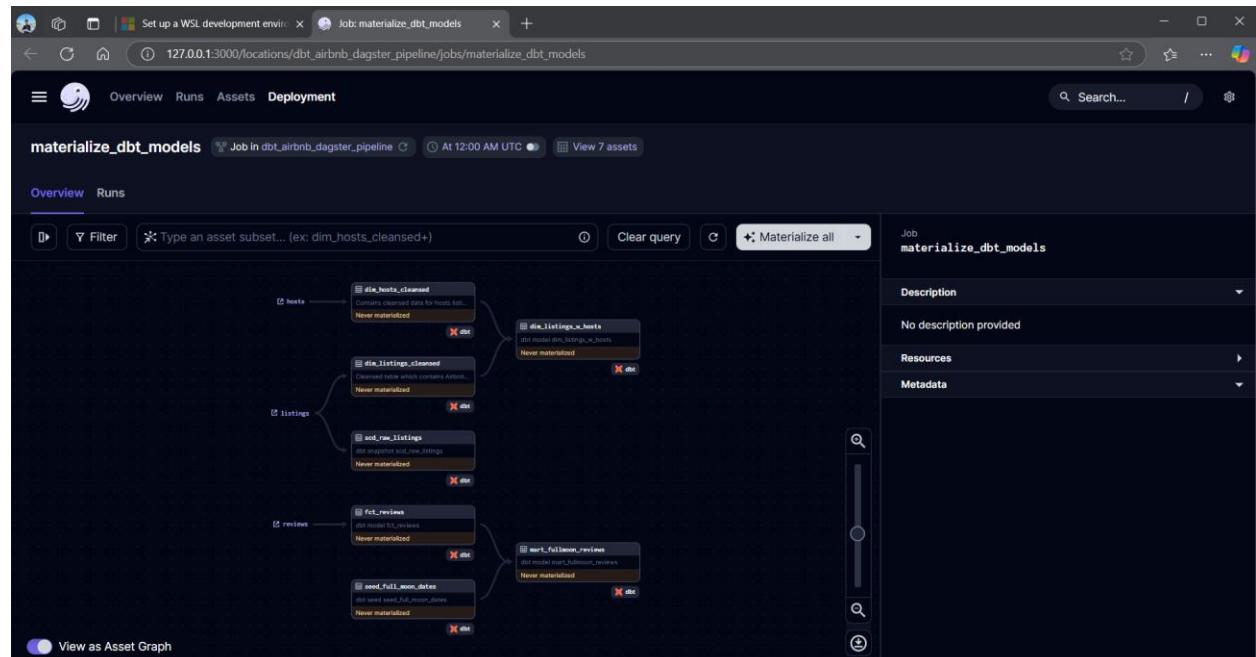
telemetry:
  enabled: false

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/39dv5sF).

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
```

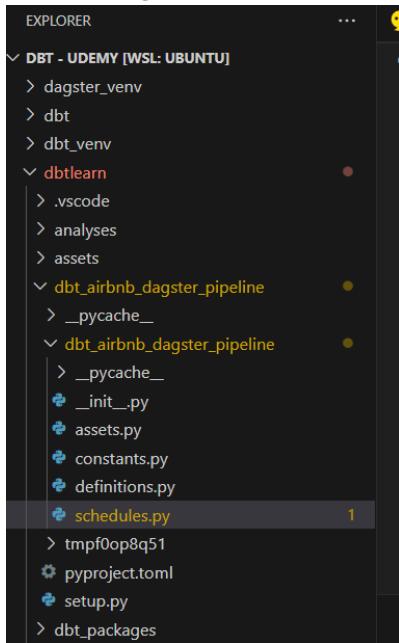
I used “dagster_dbt_parse_project_on_load=1 dagster dev” command.

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

```
import os
from pathlib import Path
from dagster_dbt import DbtCliResource

dbt_project_dir = Path(__file__).joinpath("../", "..", "..").resolve() #specifies that the dbt project is from dbtlearn folder
dbt = DbtCliResource(project_dir=os.fspath(dbt_project_dir))

# If DAGSTER_DBT_PARSE_PROJECT_ON_LOAD is set, a manifest will be created at run time.
# Otherwise, we expect a manifest to be present in the project's target directory.
if os.getenv("DAGSTER_DBT_PARSE_PROJECT_ON_LOAD"):
    dbt_manifest_path = (
        dbt.cli(
            ["--quiet", "parse"],
            target_path="target",
        )
        .wait()
        .target_path.joinpath("manifest.json")
    )
else:
    dbt_manifest_path = dbt_project_dir.joinpath("target", "manifest.json") #this manifest.json file is very important as it states everything that a software needs to know about the dbt project.
```

File Purpose

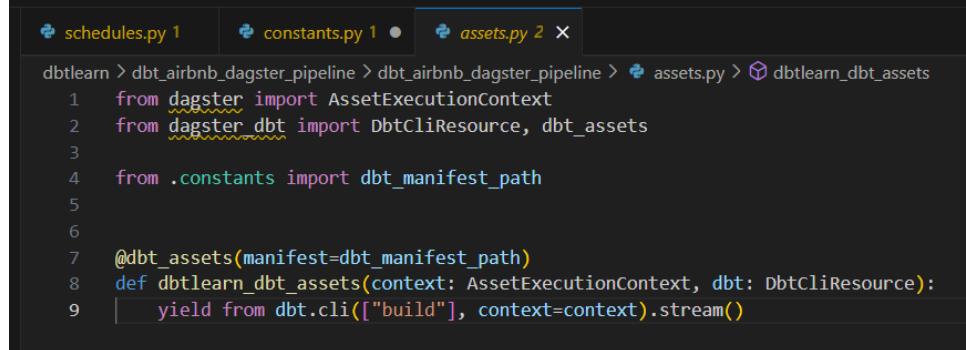
constants.py holds **shared configuration** and **paths** that tell Dagster how to locate and interact with your dbt project.

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



```
dbtlearn > dbt_airbnb_dagster_pipeline > dbt_airbnb_dagster_pipeline > assets.py > dbtlearn_dbt_assets
1 from dagster import AssetExecutionContext
2 from dagster_dbt import DbtCliResource, dbt_assets
3
4 from .constants import dbt_manifest_path
5
6
7 @dbt_assets(manifest=dbt_manifest_path)
8 def dbtlearn_dbt_assets(context: AssetExecutionContext, dbt: DbtCliResource):
9     yield from dbt.cli(["build"], context=context).stream()
```

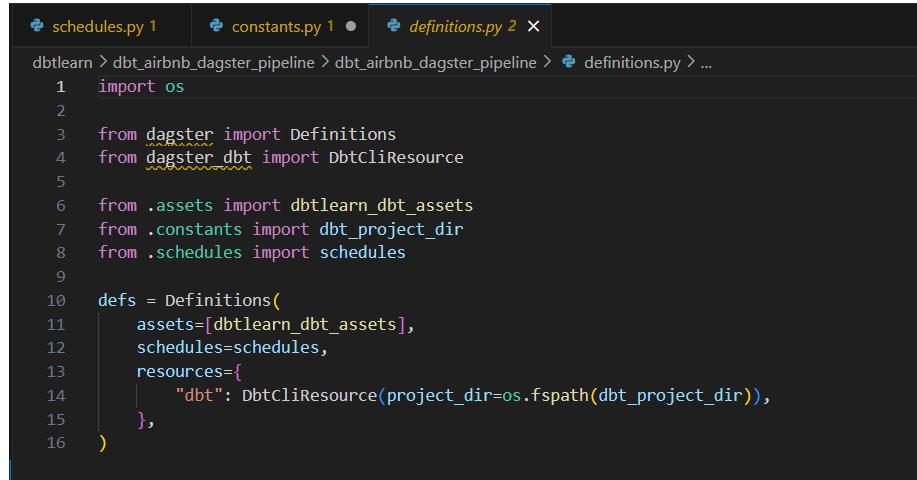
File Purpose

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



```
dbtlearn > dbt_airbnb_dagster_pipeline > dbt_airbnb_dagster_pipeline > definitions.py > ...
1 import os
2
3 from dagster import Definitions
4 from dagster_dbt import DbtCliResource
5
6 from .assets import dbtlearn_dbt_assets
7 from .constants import dbt_project_dir
8 from .schedules import schedules
9
10 defs = Definitions(
11     assets=[dbtlearn_dbt_assets],
12     schedules=schedules,
13     resources={
14         "dbt": DbtCliResource(project_dir=os.fspath(dbt_project_dir)),
15     },
16 )
```

File Purpose

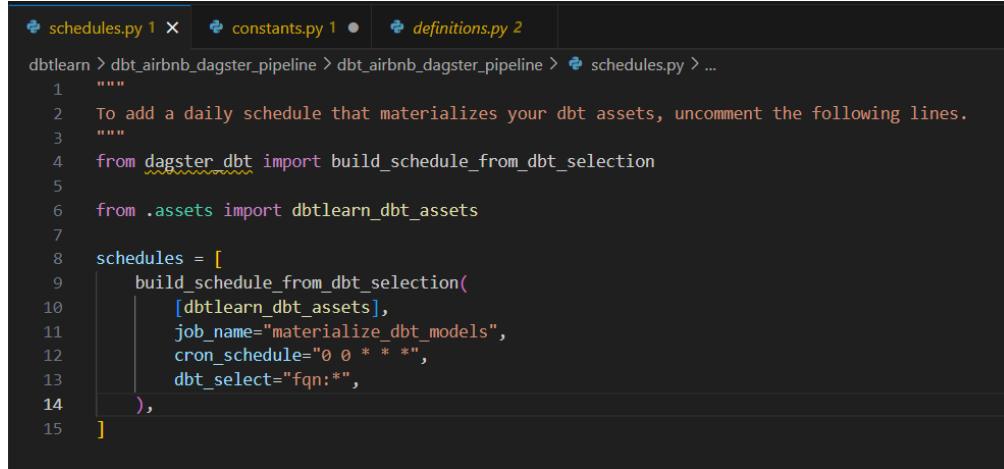
definitions.py serves as the **central registry** for your Dagster project. It ties together your assets, schedules, and resources so Dagster knows what to load and run.

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



```
schedules.py 1 | constants.py 1 | definitions.py 2
dbtlearn > dbt_airbnb_dagster_pipeline > dbt_airbnb_dagster_pipeline > schedules.py > ...
1 """
2 To add a daily schedule that materializes your dbt assets, uncomment the following lines.
3 """
4 from dagster_dbt import build_schedule_from_dbt_selection
5
6 from .assets import dbtlearn_dbt_assets
7
8 schedules = [
9     build_schedule_from_dbt_selection(
10         [dbtlearn_dbt_assets],
11         job_name="materialize_dbt_models",
12         cron_schedule="0 0 * * *",
13         dbt_select="fqn:*",
14     ),
15 ]
```

File Purpose

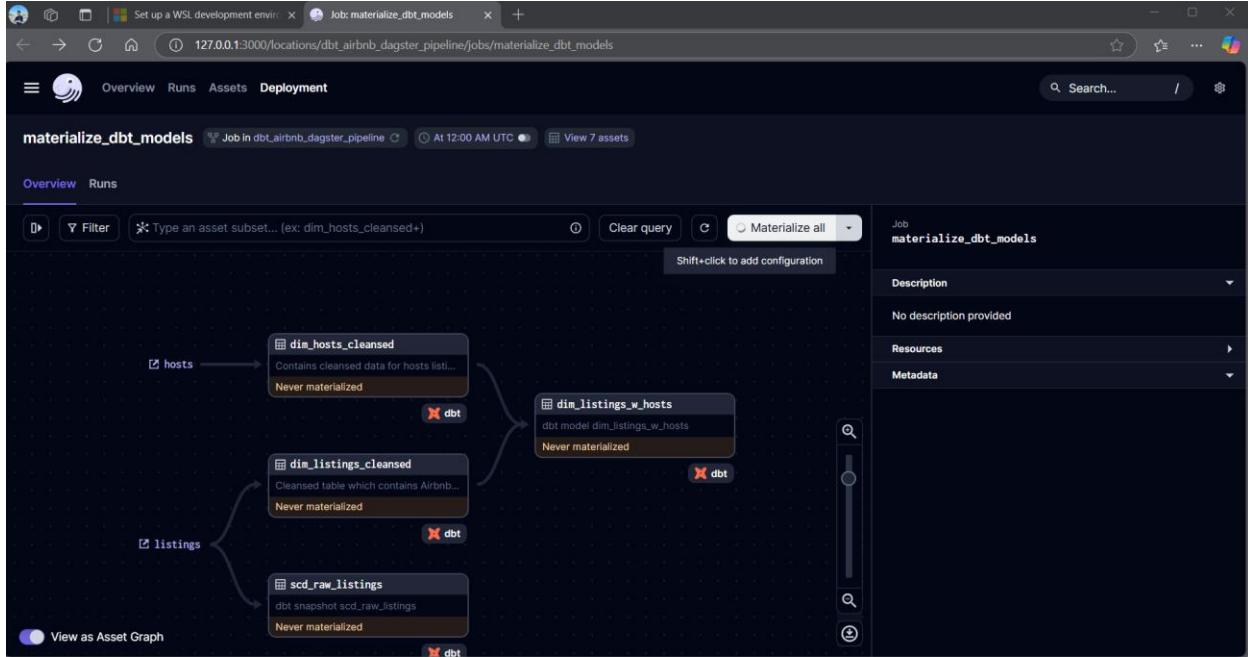
schedules.py defines how often and which dbt assets should be run automatically.

Key Points

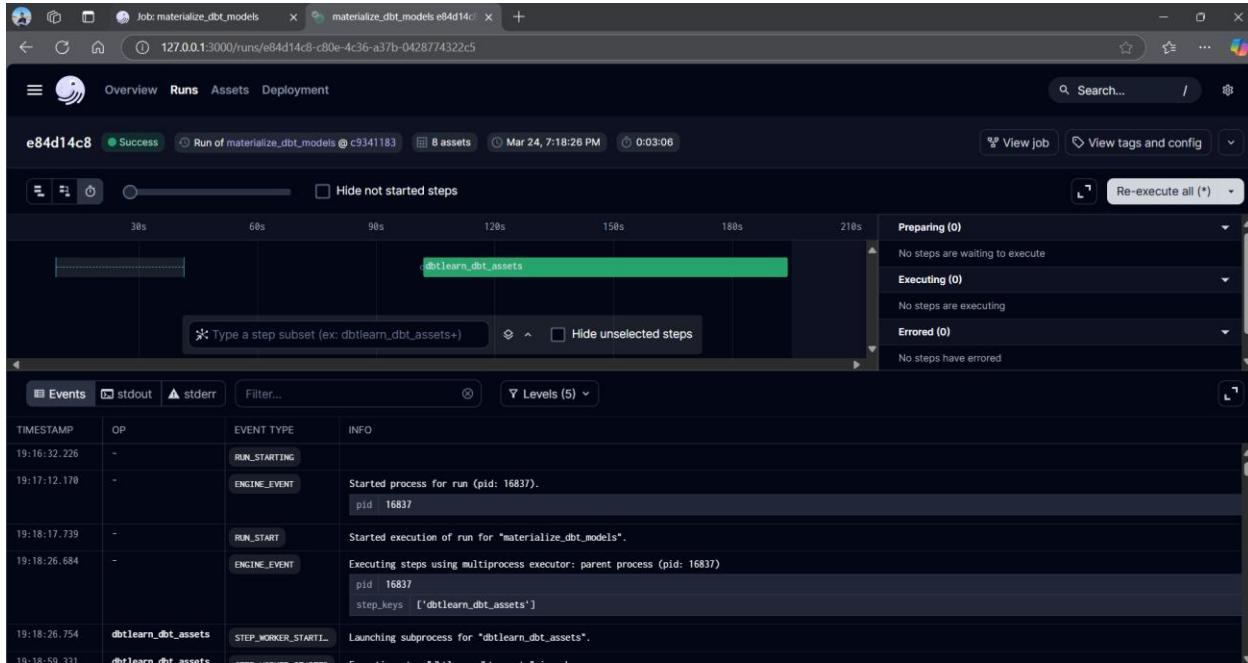
- **build_schedule_from_dbt_selection(...):** Creates a Dagster schedule that runs your dbt models on a set cron schedule.
- **cron_schedule="0 0 * * *":** Runs every day at midnight (example timing).
- **dbt_select="fqn:*":** Tells the schedule to select all dbt assets (the “fully qualified name” selector).

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):

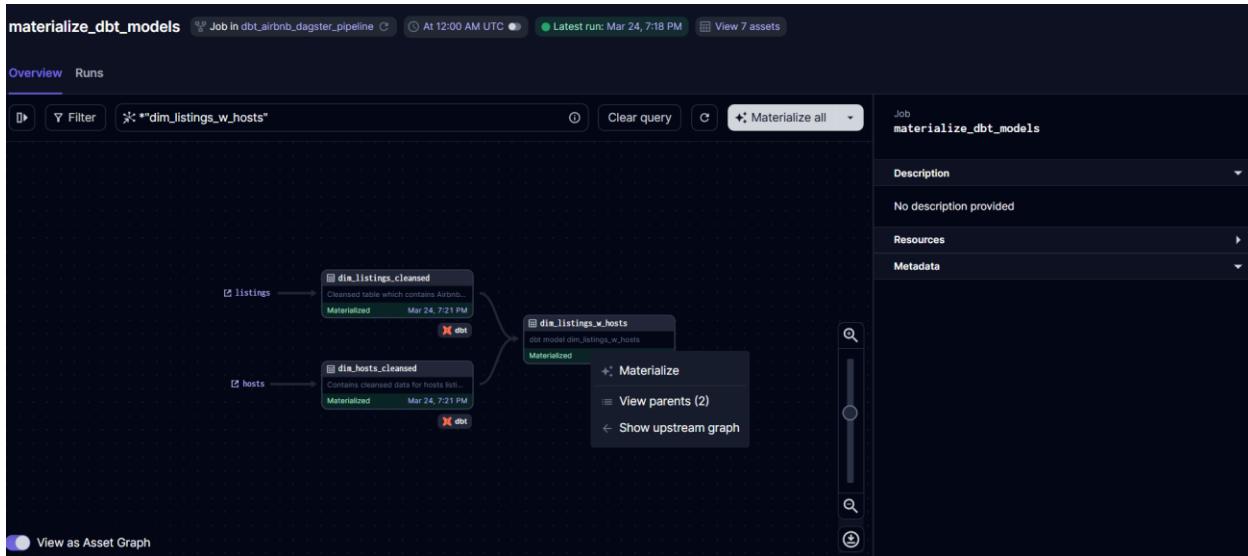
- **Timestamps**: Precise logs of execution events.
- **Event Types**:
 - RUN_START, RUN_STARTING: Marks the beginning of the run.
 - ENGINE_EVENT: Internal Dagster events (e.g., launching process).
 - STEP_WORKER_STARTING: Execution start for a step (e.g., dbt assets).
- **INFO column**: Human-readable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

Schedule name	Schedule	Running	Last tick	Last run	Actions
dbt_airbnb_dagster_pipeline					1
materialize_dbt_models_schedule	materialize_dbt_models	At 12:00 AM UTC Next tick: Mar 26, 12:00 AM UTC	<input checked="" type="checkbox"/>	None	None

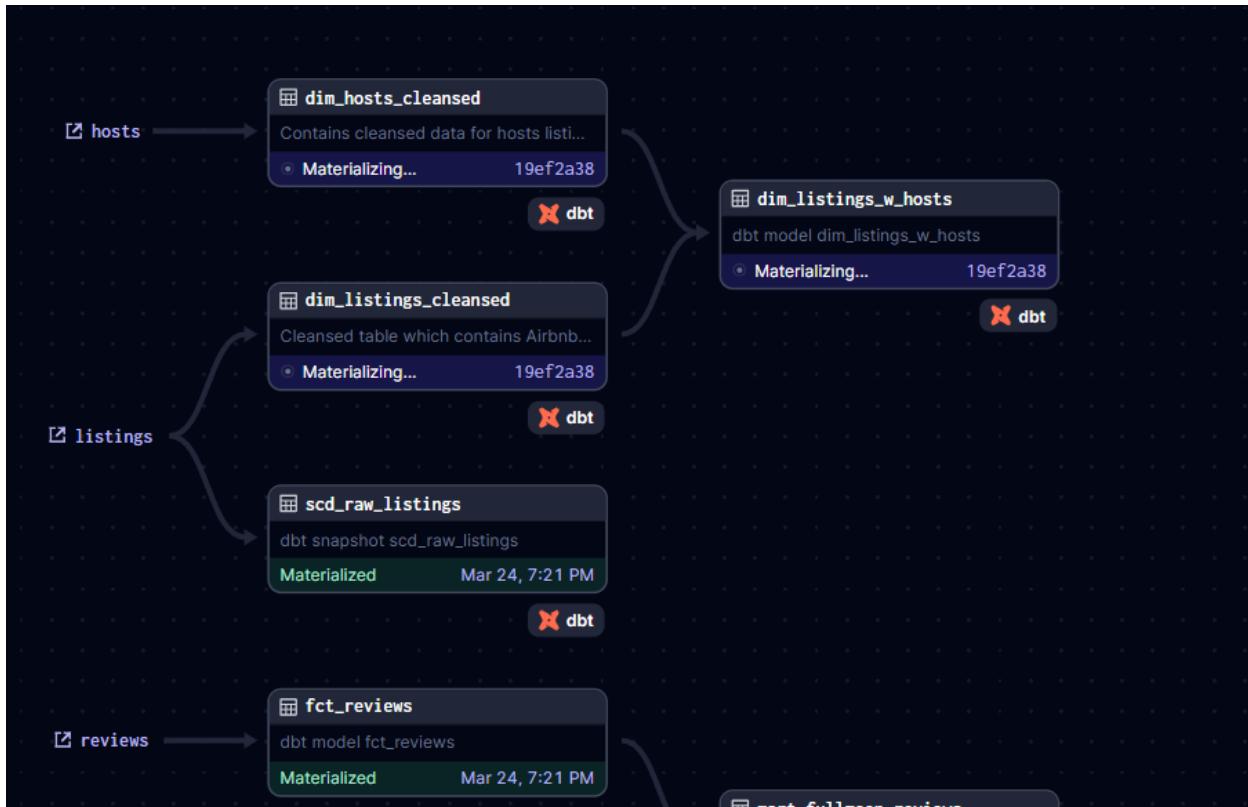
We also have a scheduled job set to run at midnight which will materialize the models as per the set schedule in schedules.py

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 –



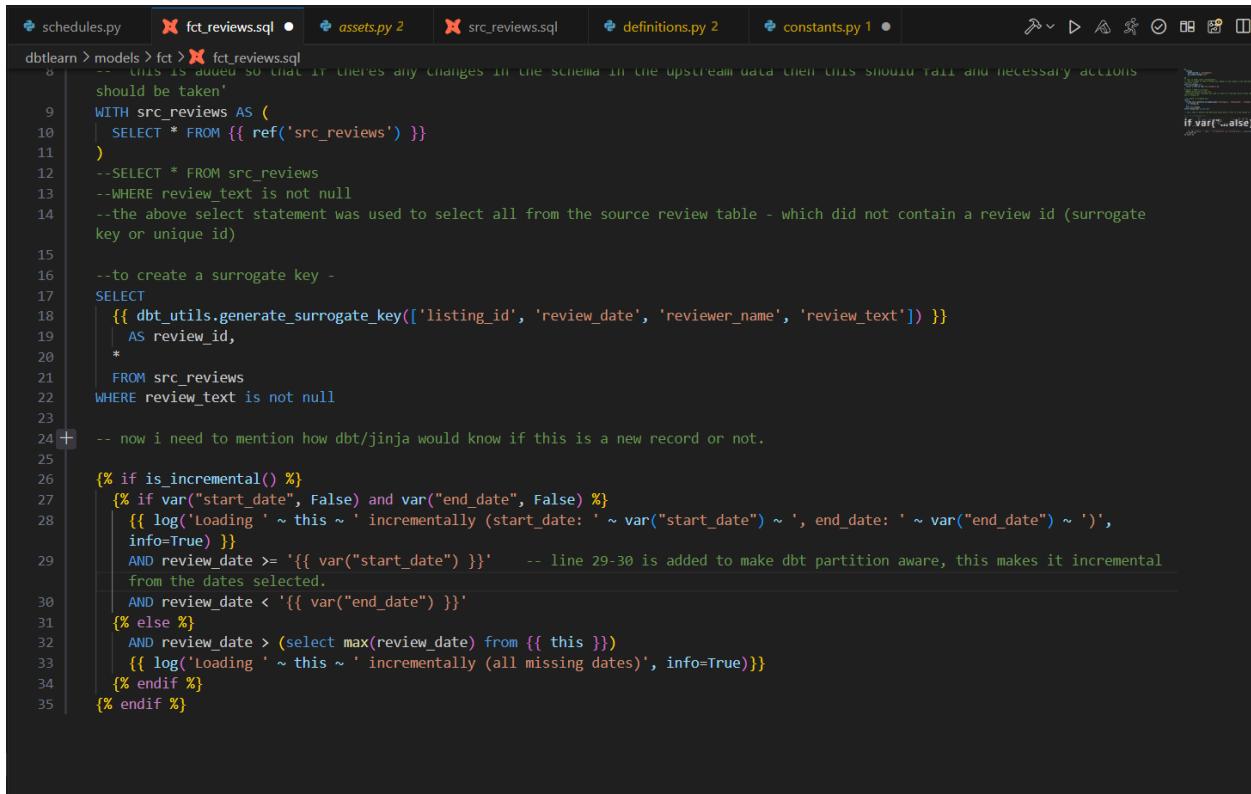
As you can see only the models that were in scope were being materialized.

- 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, we need incremental models so that 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 -



```
8 -- this is added so that if there's any changes in the schema in the upstream data then this should fail and necessary actions
9 -- should be taken'
10 WITH src_reviews AS (
11   |   SELECT * FROM {{ ref('src_reviews') }}
12 )
13 --SELECT * FROM src_reviews
14 --WHERE review_text is not null
15 --the above select statement was used to select all from the source review table - which did not contain a review id (surrogate
16 --key or unique id)
17
18 --to create a surrogate key -
19 SELECT
20   | {{ dbt_utils.generate_surrogate_key(['listing_id', 'review_date', 'reviewer_name', 'review_text']) }}
21   | AS review_id,
22   *
23   | FROM src_reviews
24 WHERE review_text is not null
25
26 -- now i need to mention how dbt/jinja would know if this is a new record or not.
27
28 (% if is_incremental() %
29   | {% if var("start_date", False) and var("end_date", False) %}
30     | {{ log('Loading ' ~ this ~ ' incrementally (start_date: ' ~ var("start_date") ~ ', end_date: ' ~ var("end_date") ~ ')',
31           info=True) }}
32     | AND review_date >= '{{ var("start_date") }}'    -- line 29-30 is added to make dbt partition aware, this makes it incremental
33     |         from the dates selected.
34     | AND review_date < '{{ var("end_date") }}'
35   | {% else %}
36     | AND review_date > (select max(review_date) from {{ this }})
37     | {{ log('Loading ' ~ this ~ ' incrementally (all missing dates)', info=True)}}
38   | {% endif %}
39   | {% endif %}
```

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 –

```

schedules.py | ✘ fct_reviews.sql | ✘ assets.py 2 | ✘ src_reviews.sql | ✘ definitions.py 2 | ✘ constants.py 1 | ...
dbtlearn > dbt_airbnb_dagster_pipeline > dbt_airbnb_dagster_pipeline > assets.py > CustomDagsterDbtTranslator > get_metadata > dbt_resource_props
25     from .constants import dbt_manifest_path
26
27     # Non-partitioned assets
28     @dbt_assets(manifest=dbt_manifest_path, exclude="fct_reviews")
29     def dbtlearn_dbt_assets(context: AssetExecutionContext, dbt: DbtCliResource):
30         yield from dbt.cli(["build"], context=context).stream()
31
32
33     # Partitioning setup for fct_reviews
34     daily_partitions = DailyPartitionsDefinition(start_date="2022-01-24")
35
36
37     class CustomDagsterDbtTranslator(DagsterDbtTranslator):
38         def get_metadata(
39             self, dbt_resource_props: Mapping[str, Any]
40         ) -> Mapping[str, Any]:
41             metadata = {"partition_expr": "date"}
42             default = default_metadata_from_dbt_resource_props(dbt_resource_props)
43             return {**default, **metadata}
44
45
46     # Partitioned asset
47     @dbt_assets(
48         manifest=dbt_manifest_path,
49         select="fct_reviews",
50         partitions_def=daily_partitions,
51         dagster_dbt_translator=CustomDagsterDbtTranslator(),
52     )
53     def dbtlearn_partitioned_dbt_assets(context: AssetExecutionContext, dbt: DbtCliResource):
54         first_partition, last_partition = context.asset_partitions_time_window_for_output(
55             list(context.selected_output_names)[0]
56         )
57
58         dbt_vars = {
59             "start_date": str(first_partition),
60             "end_date": str(last_partition),
61         }
62
63         dbt_args = ["build", "--vars", json.dumps(dbt_vars)]
64         dbt_cli_task = dbt.cli(dbt_args, context=context, raise_on_error=False)
65         yield from dbt_cli_task.stream()
66

```

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

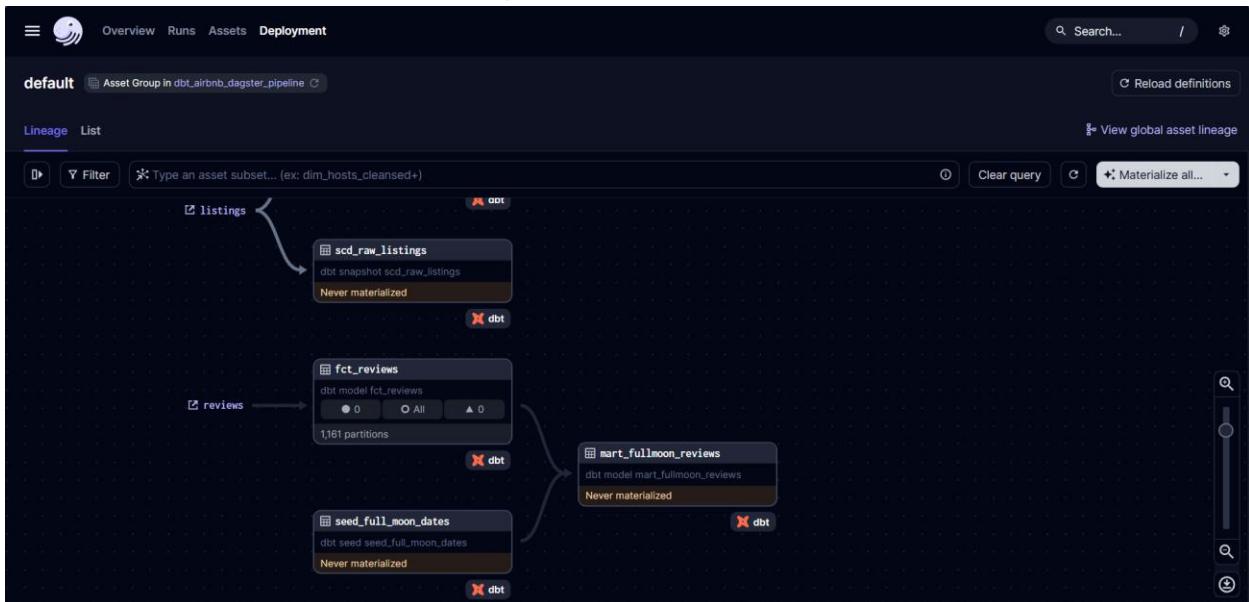
```

@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

ID default
Select partitions to materialize. Click and drag to select a range on the timeline.

[2022-01-24...2025-03-29]

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

This screenshot shows a backfill configuration dialog in the Dagster UI. At the top, it displays a note about launching runs to materialize 7 assets and a warning section with one warning. Below this is a 'Partition selection' section showing 1,161 partitions across a timeline from 2022-01-24 to 2025-03-29. There are buttons for 'Latest' and 'All' partitions. Under 'Options', there is a checkbox for 'Backfill only failed and missing partitions within selection'. A prominent purple info box at the bottom states: 'Dagster will materialize the selected partitions for the selected assets using varying backfill policies.' with a 'Preview' link. At the bottom right are 'Cancel' and 'Launch backfill' buttons.

❖ Launch runs to materialize 7 assets

Partition selection 1 partition

default
Select partitions to materialize. Click and drag to select a range on the timeline.

2025-03-29 Latest All

2022-01-24 2025-03-29

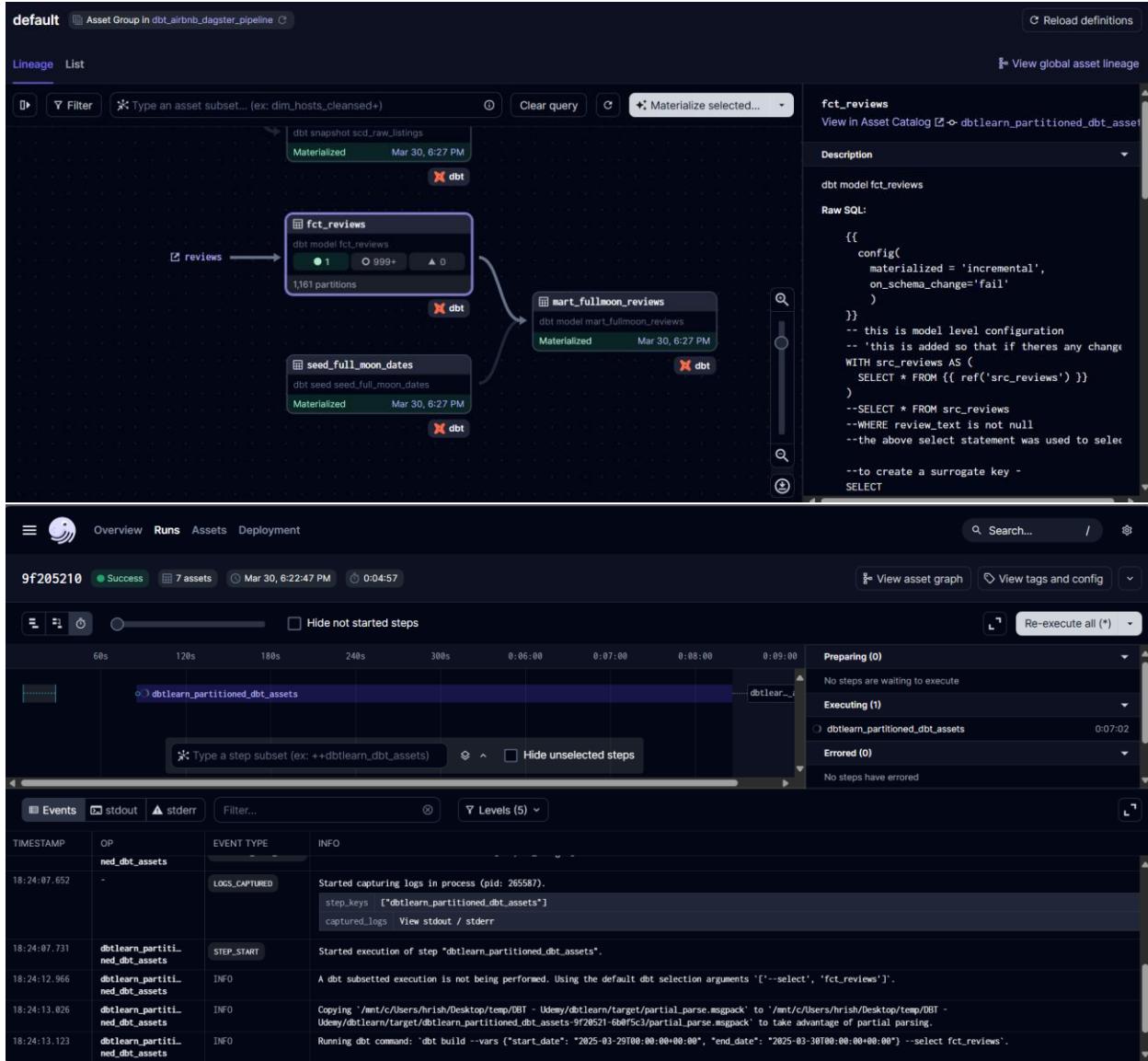
Tags 0 tags
Tags will be applied to all backfill runs

Options
 Backfill only failed and missing partitions within selection

Dagster will materialize the selected partitions for the selected assets using varying backfill policies. Only 1 partition failed and missing partitions will be materialized.

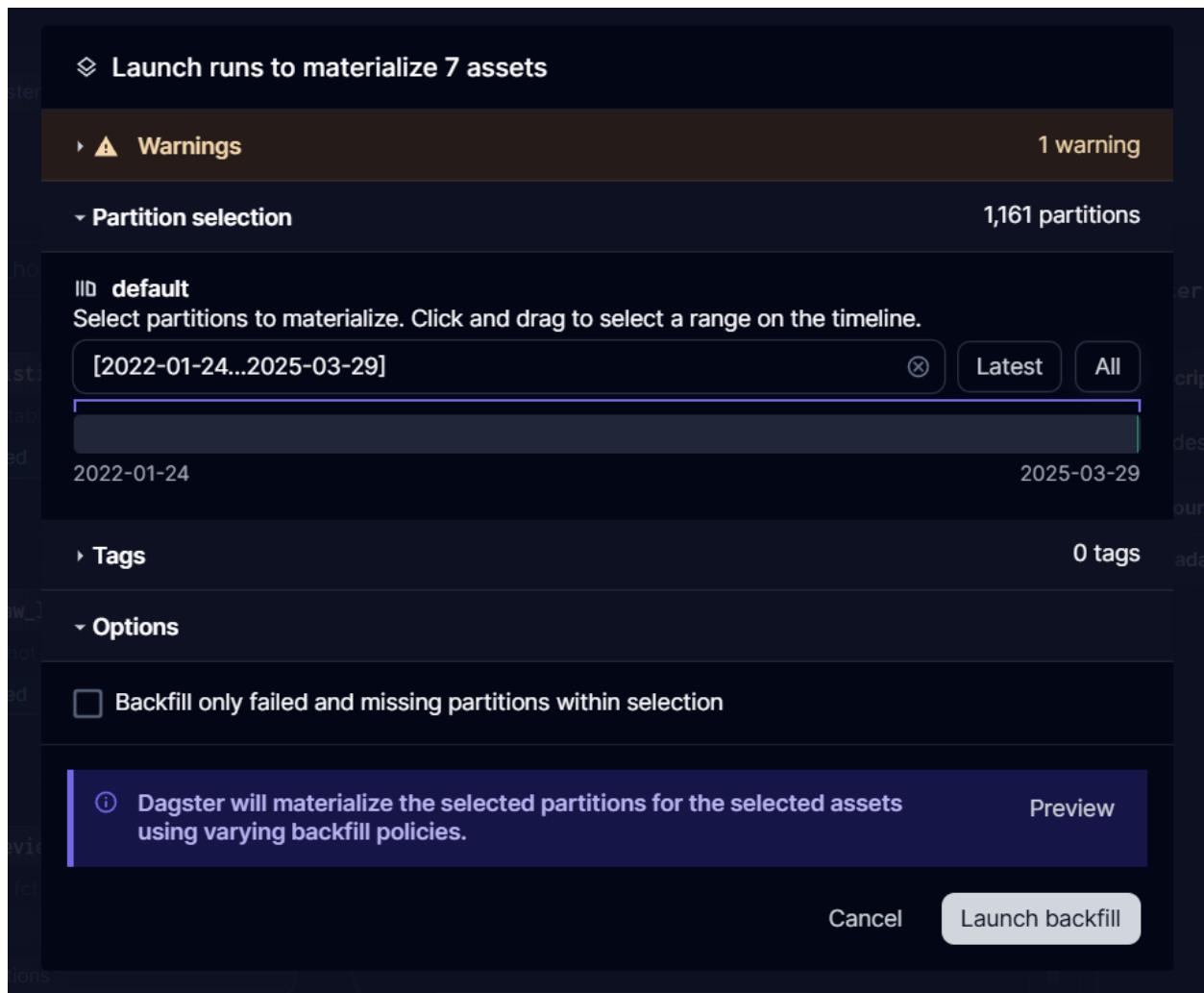
Cancel

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 –

① Error

```
sqlalchemy.exc.OperationalError: (sqlite3.OperationalError) unable to open database file
(Background on this error at: https://sqlalche.me/e/20/e3q8)
File "/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dagster_venv/lib/python3.10/site-packages/dagster/_core/instance/_init__.py", line 2505,
    submitted_run = self.run_coordinator.submit_run(
File "/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dagster_venv/lib/python3.10/site-packages/dagster/_core/run_coordinator/default_run_coordinator.py", line 160,
    self._instance.launch_run(dagster_run.run_id, context.workspace)
File "/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dagster_venv/lib/python3.10/site-packages/dagster/_core/instance/_init__.py", line 2560,
    self.report_engine_event(
File "/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dagster_venv/lib/python3.10/site-packages/dagster/_core/instance/_init__.py", line 2374,
    self.report_dagster_event(dagster_event, run_id=run_id, log_level=log_level)
File "/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dagster_venv/lib/python3.10/site-packages/dagster/_core/instance/_init__.py", line 2396,
    self.handle_new_event(event_record)
File "/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dagster_venv/lib/python3.10/site-packages/dagster/_core/instance/_init__.py", line 2315,
    self._event_storage.store_event(event)
File "/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dagster_venv/lib/python3.10/site-packages/dagster/_core/storage/event_log/sqlite/sqlite_event_log.py", line 135, in __enter__
    with self.run_connection(run_id) as conn:
File "/usr/lib/python3.10/contextlib.py", line 135, in __enter__
    return next(self.gen)
File "/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dagster_venv/lib/python3.10/site-packages/dagster/_core/storage/event_log/sqlite/sqlite_event_log.py", line 135, in __enter__
    with engine.connect() as conn:
File "/mnt/c/Users/hrish/Desktop/temp/DBT - Udemy/dagster_venv/lib/python3.10/site-packages/sqlalchemy/engine/base.py", line 3274, in connect
    return dialect.connect(

```

Copy **OK**

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

⚡ Launch runs to materialize fct_reviews

▶ ⚠ Warnings 1 warning

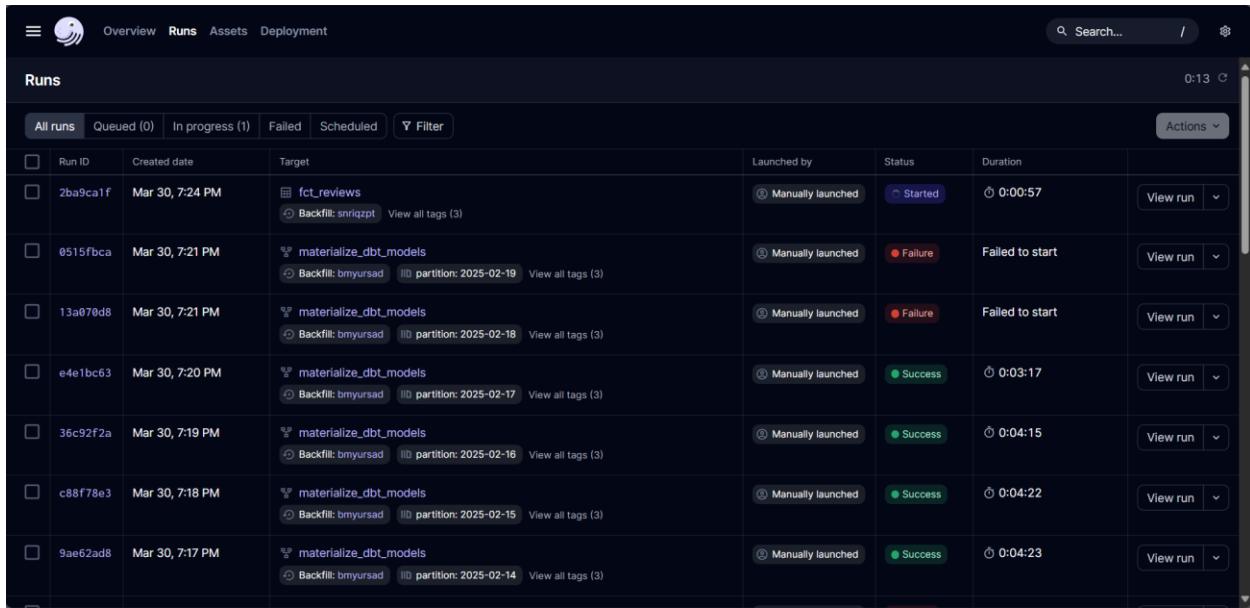
▼ Partition selection 49 partitions

IID default
Select partitions to materialize. Click and drag to select a range on the timeline.

[2025-02-07...2025-03-27] × Latest All

2022-01-24 2025-03-30

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



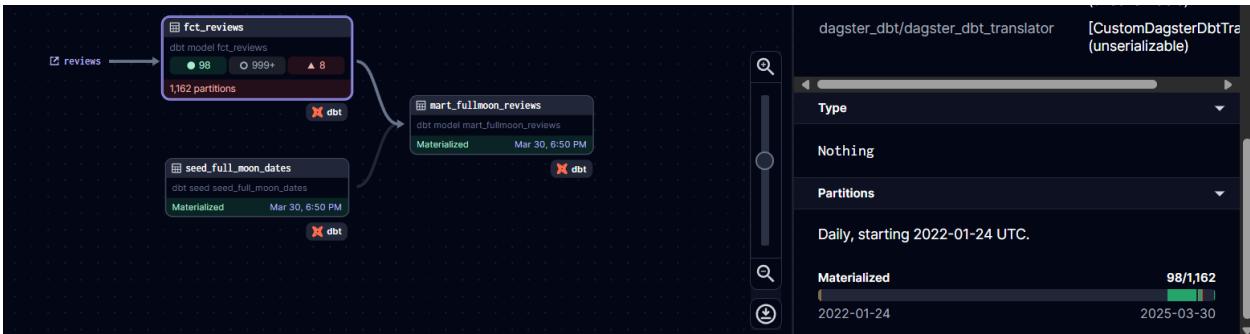
The screenshot shows a table of runs in the Dagster UI. The columns are: Run ID, Created date, Target, Launched by, Status, Duration, and Actions. There are 8 rows, each corresponding to a different partition. The 'Actions' column contains a 'View run' button for each row.

All runs	Queued (0)	In progress (1)	Failed	Scheduled	Filter	Actions
<input type="checkbox"/> 2ba9ca1f	Mar 30, 7:24 PM	fct_reviews Backfill: snriqzpt View all tags (3)	Manually launched	Started	0:00:57	<button>View run</button>
<input type="checkbox"/> 0515fbca	Mar 30, 7:21 PM	materialize_dbt_models Backfill: bmyursad partition: 2025-02-19 View all tags (3)	Manually launched	Failure	Failed to start	<button>View run</button>
<input type="checkbox"/> 13a070d8	Mar 30, 7:21 PM	materialize_dbt_models Backfill: bmyursad partition: 2025-02-18 View all tags (3)	Manually launched	Failure	Failed to start	<button>View run</button>
<input type="checkbox"/> e4e1bc63	Mar 30, 7:20 PM	materialize_dbt_models Backfill: bmyursad partition: 2025-02-17 View all tags (3)	Manually launched	Success	0:03:17	<button>View run</button>
<input type="checkbox"/> 36c92f2a	Mar 30, 7:19 PM	materialize_dbt_models Backfill: bmyursad partition: 2025-02-16 View all tags (3)	Manually launched	Success	0:04:15	<button>View run</button>
<input type="checkbox"/> c88f78e3	Mar 30, 7:18 PM	materialize_dbt_models Backfill: bmyursad partition: 2025-02-15 View all tags (3)	Manually launched	Success	0:04:22	<button>View run</button>
<input type="checkbox"/> 9ae62ad8	Mar 30, 7:17 PM	materialize_dbt_models Backfill: bmyursad partition: 2025-02-14 View all tags (3)	Manually launched	Success	0:04:23	<button>View run</button>

Required resources	
dbt	
Metadata	
table_schema	[Show Table Schema]
partition_expr	date
dagster_dbt/manifest	[DbtManifestWrapper] (unserializable)
dagster_dbt/dagster_dbt_translator	[CustomDagsterDbtTra (unserializable)]
Type	
Nothing	
Partitions	
Daily, starting 2022-01-24 UTC.	
Materialized	
2022-01-24	13/1,162 2025-03-30

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.