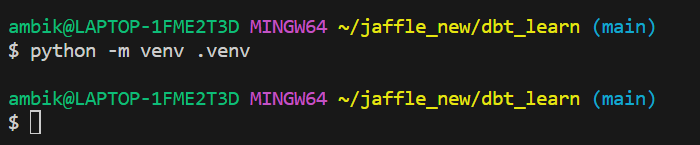
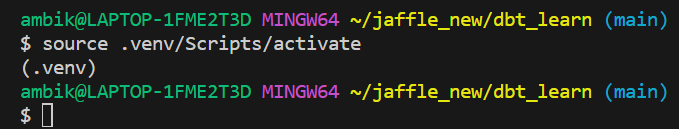
DBT BOOT CAMP TRAINING SCREEN SHOTS:

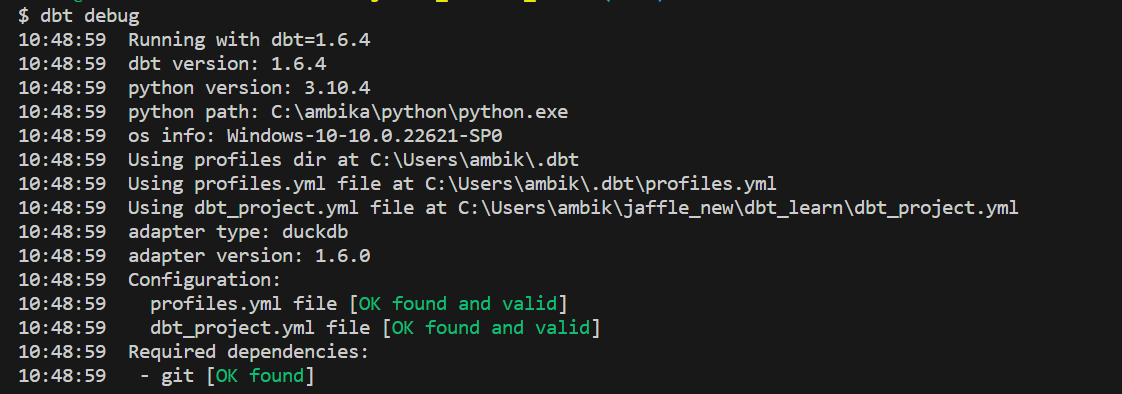
Creating Python Environment:

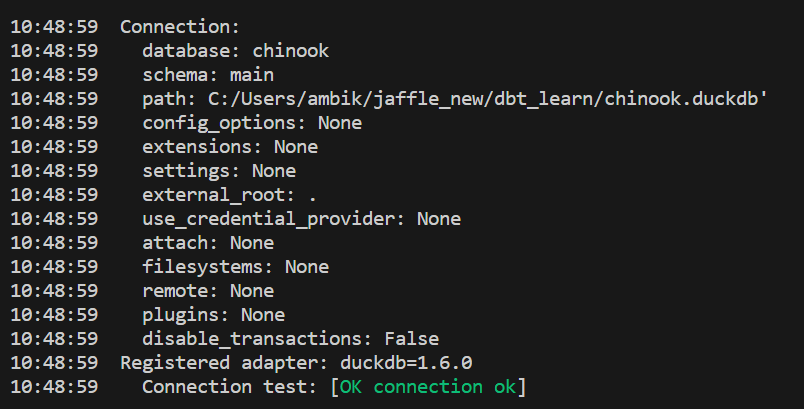


Activating Python Environment:



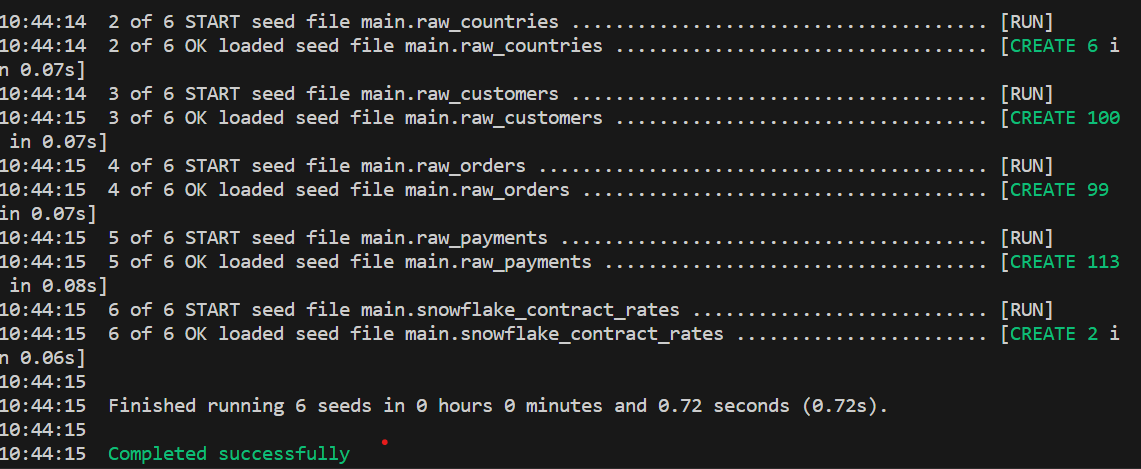
Dbt debug:



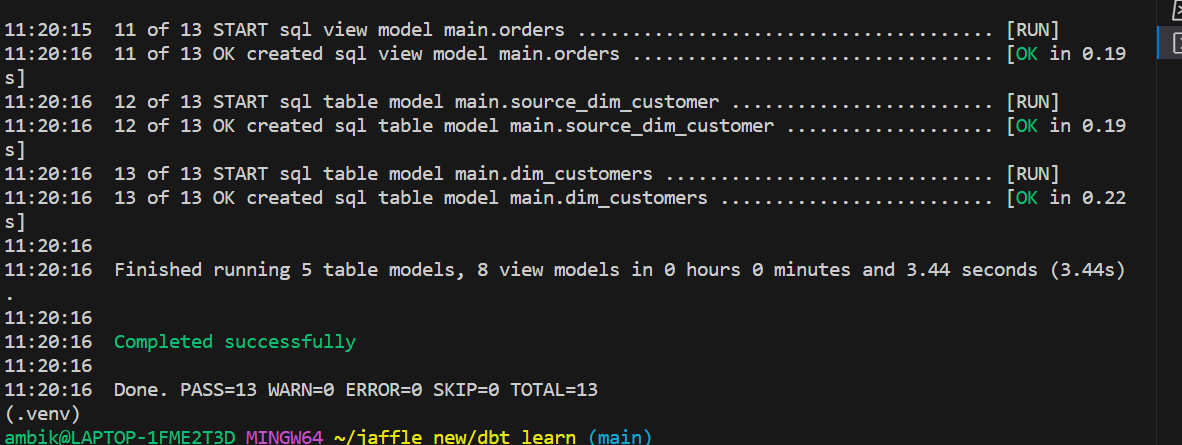


Dbt-seed full-refresh accepts the new seed file along with old files in the seeds folder





DBT Run:

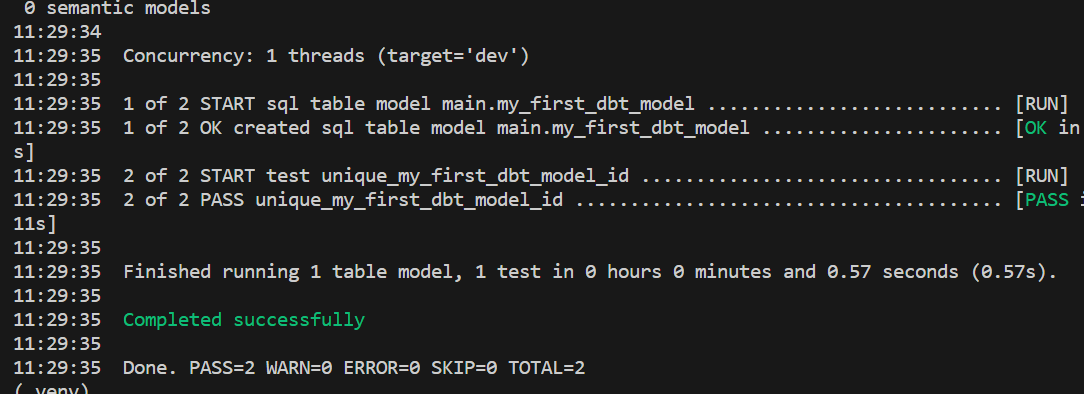


Dbt run -s / dbt run –select “modle\_name “ or dbt run –select model\_name

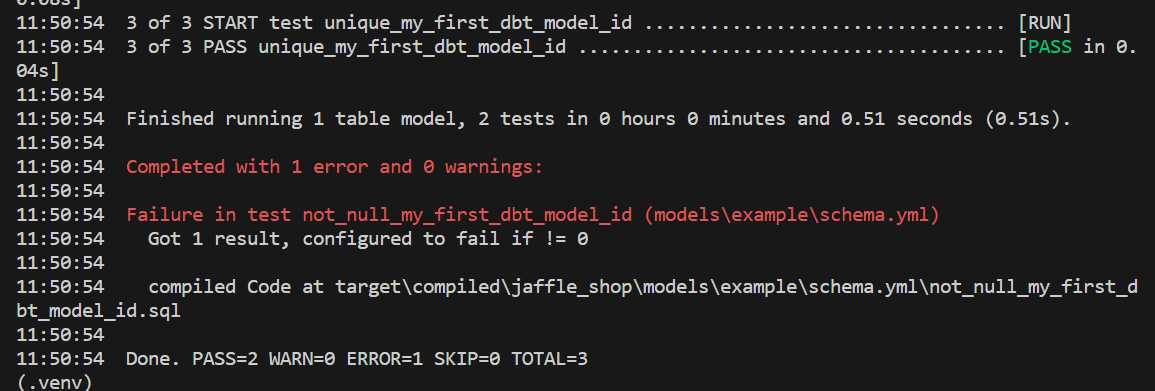
use the below command after dbt run or dbt build to rerun failed models  
dbt run --select result:error --state path/to/target-folder

DBT BUILD

$ dbt build -s my\_first\_dbt\_model



The model is built as well the model is tested as DBT build command builds the model as well it test the model as well



As this table has got null values in the table..it is failing in the not\_null test

Schema.yml

version: 2

models:

    - name: my\_first\_dbt\_model

      description: "A starter dbt model"

      columns:

          - name: id

            description: "The primary key for this table"

            tests:

                - unique

                - not\_null

    - name: my\_second\_dbt\_model

      description: "A starter dbt model"

      columns:

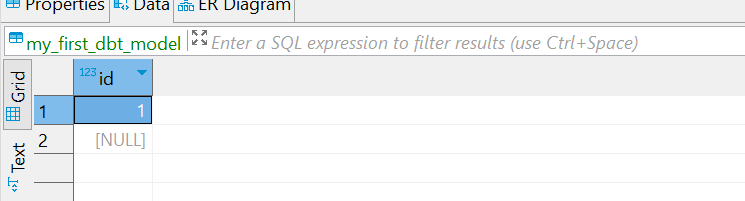
          - name: id

            description: "The primary key for this table"

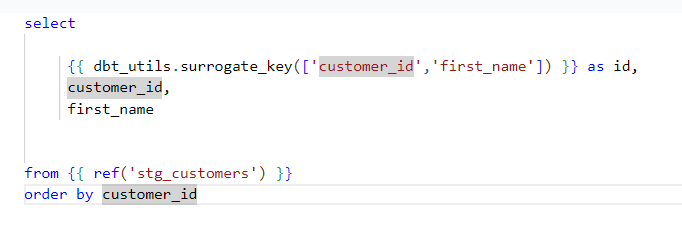
            tests:

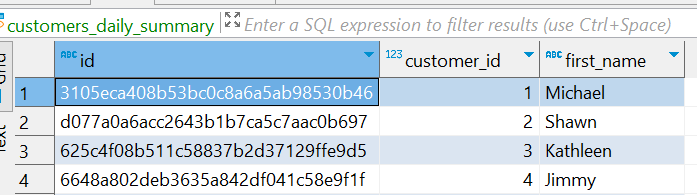
                - unique

                - not\_null

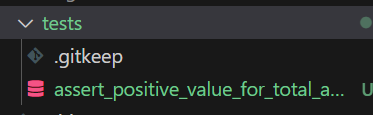


Surrogate Key:





DBT Test:



select

  order\_id,

    sum(amount) as total\_amount

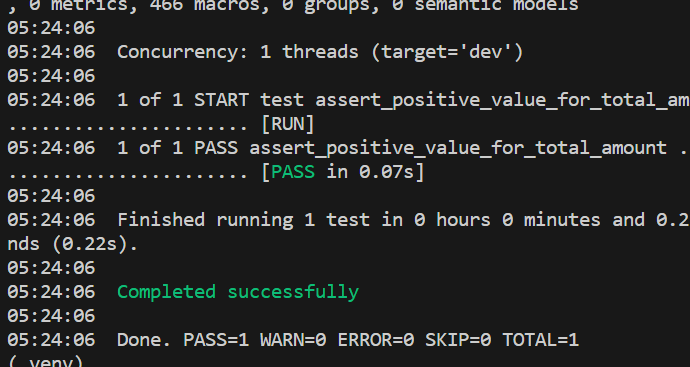
from {{ ref('stg\_payments') }}

group by 1

having not(total\_amount >= 0)

This comes under singular test …when I run this command

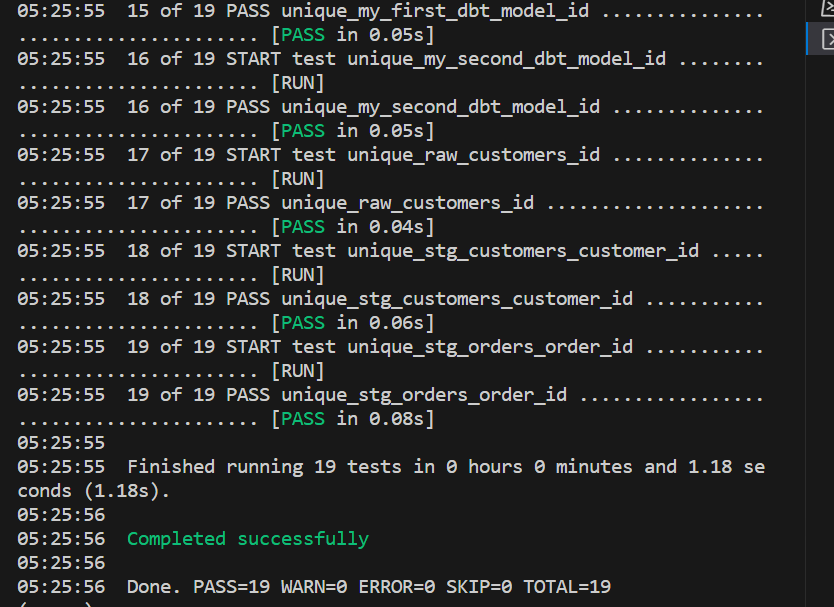
dbt test --select test\_type:singular



GENERIC TEST:

This Test returns the test results for all the tests I have defined for Sources, Models.





DBT Docs BLOCK:

{% docs status %}

One of the following values:

| status         | definition                                       |

|----------------|--------------------------------------------------|

| placed         | Order placed, not yet shipped                    |

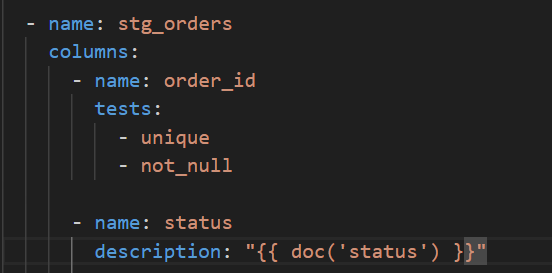
| shipped        | Order has been shipped, not yet been delivered   |

| completed      | Order has been received by customers             |

| return pending | Customer indicated they want to return this item |

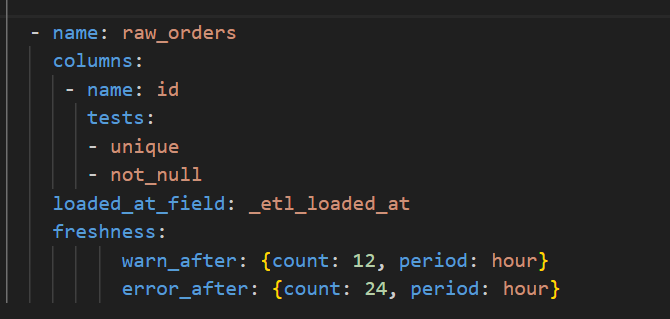
| returned       | Item has been returned                           |

{% enddocs %}



DBT Source Freshness:

dbt source freshness is a dbt command to test whether your source tables have been updated within a desired frequency.



This Source freshness will run success …after 24 hours …if the data is updated

This Source freshness will warn…after 12 hours …if the data is not updated

This Source freshness will throw error…after 24 hours …if the data is not updated And it will say ur historical data is stale.

I haven’t updated the table raw\_orders (I am checking source\_freshness for this table)

I have created \_etl\_loaded\_at (timestamp) in the yml file

I have given the column type for this column in dbt\_project.yml (as timestampntz)

seeds:

    +column\_types:

      id: int

      user\_id: int

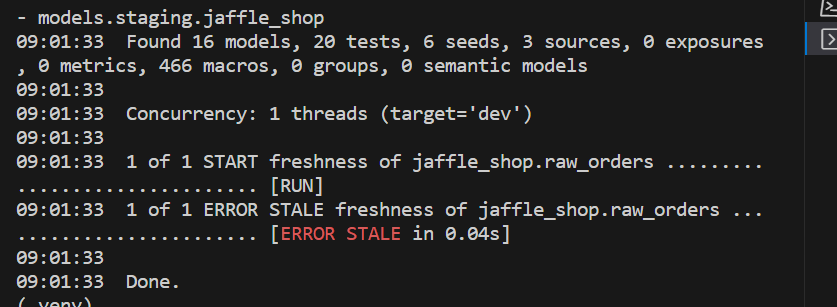
      order\_date: date

      status: varchar

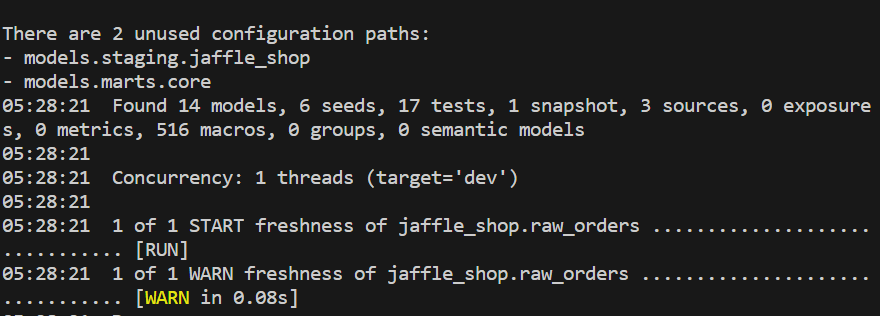
      \_etl\_loaded\_at: timestamptz

Now…I haven’t updated my data for the past 24 hours …so..I am getting this error

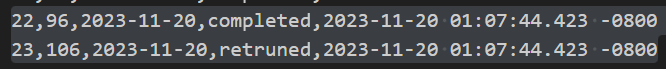
$ dbt source freshness



Now…I haven’t updated my data for the past 12 hours …so..I am getting this error



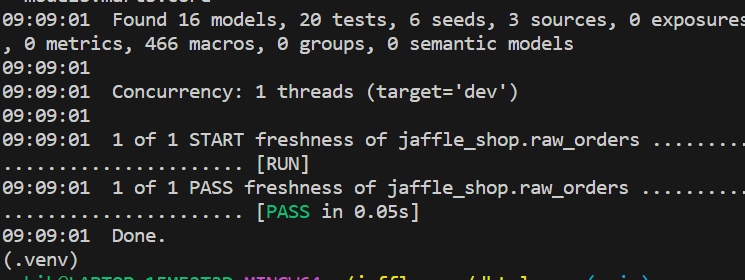
Now, I have added these two records at the end



Now I run dbt seed –full-refresh and build the model

After that I run dbt sourc freshness

Now I am able to the test of source freshness is successful



**Incremental models**

Incremental models are built as tables in your [data warehouse](https://docs.getdbt.com/terms/data-warehouse). The first time a model is run, the [table](https://docs.getdbt.com/terms/table) is built by transforming *all* rows of source data. On subsequent runs, dbt transforms *only* the rows in your source data that you tell dbt to filter for, inserting them into the target table which is the table that has already been built.

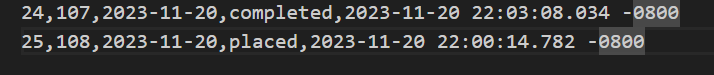
Incremental implemented as snapshot (SCD type-2)

Using DBT Snapshot:

Before updating:

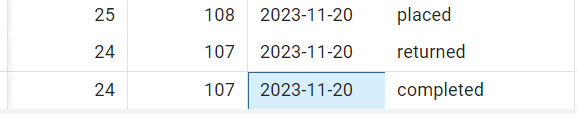
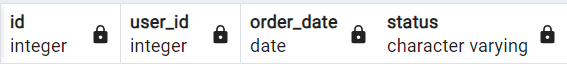


After updating

-

By running DBT command dbt seed I am loading the above data to raw orders table

By running Snapshot command …

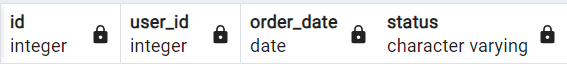


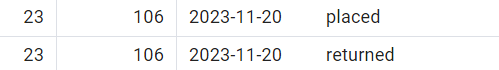




Again I am inserting new value as id= 26 and updating the old record 23

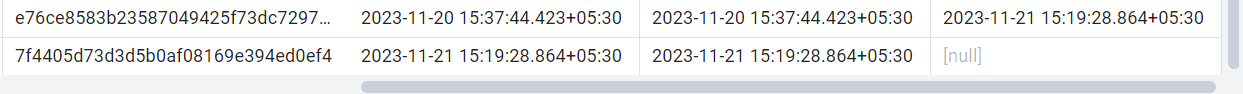
After running dbt seed and dbt snapshot













Step 6: I am changing the logic of the incremental model by adding unique key

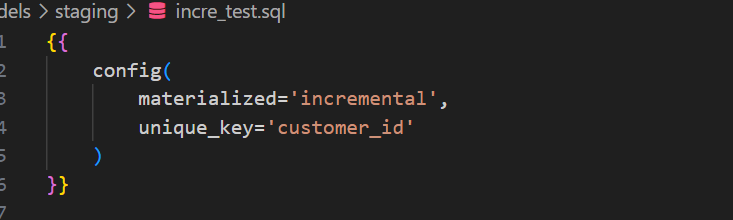
Step 7: run dbt seed –full-refresh

Step 8: db build -s stg\_orders

Old records of the models should understand the logic of the updated incremental logic as well

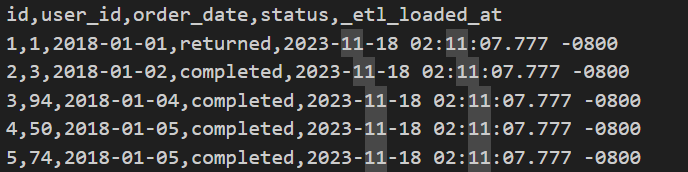
Upcoming streams should use the updated incremental logic

After running this command … we can see the old record getting updated



This is the unique key I am using…

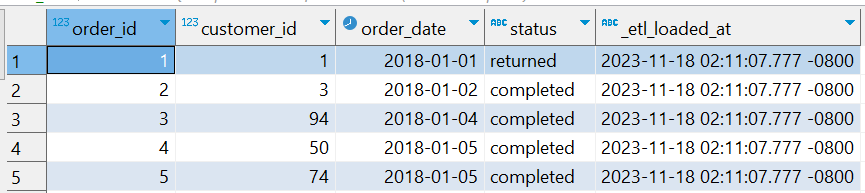
I am updating the customer id in the seed



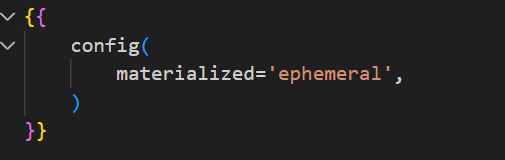
I am updating the 5th record

Step 9: 

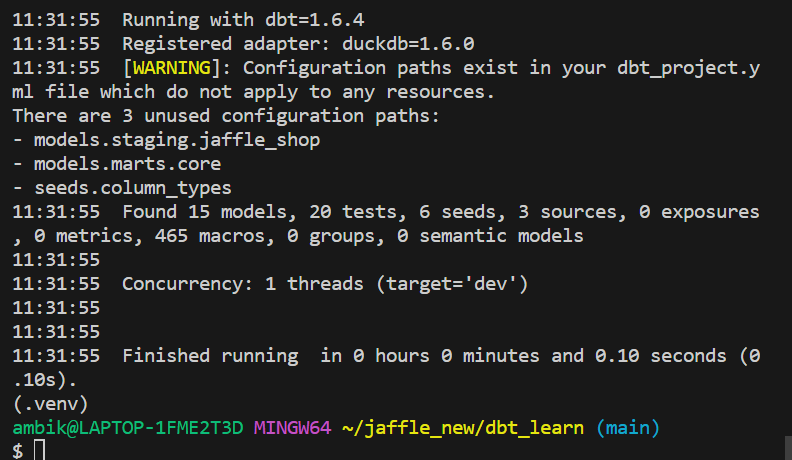
I run step 9..i could see the old record getting updated with the new value when I use this merge key



Ephemeral Models:



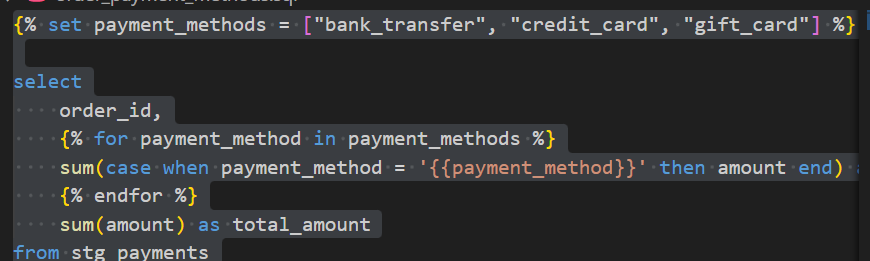
When I run this model..



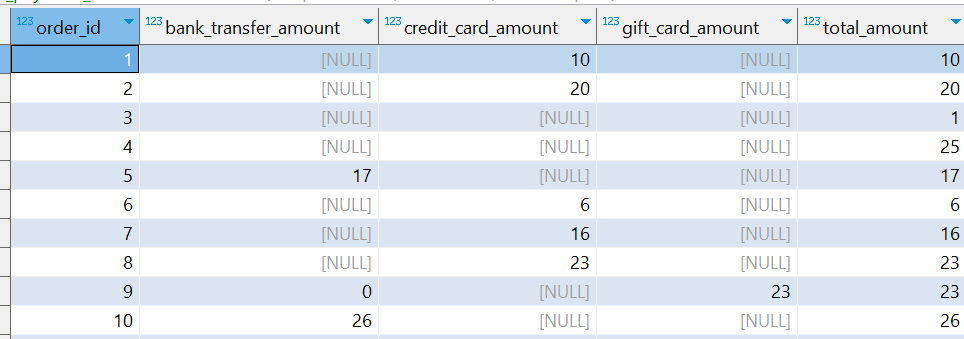
It won’t show the model created successfully message as well schema and the table name

Jinja:

Below is the jinja code written for includes list of payment methods and to calculate amount



When I run this code



Accroding to the payment method selected …amount mentioned for each payment category.

This is the complied code..

select

    order\_id,

    sum(case when payment\_method = 'bank\_transfer' then amount end) as bank\_transfer\_amount,

    sum(case when payment\_method = 'credit\_card' then amount end) as credit\_card\_amount,

    sum(case when payment\_method = 'gift\_card' then amount end) as gift\_card\_amount,

    sum(amount) as total\_amount

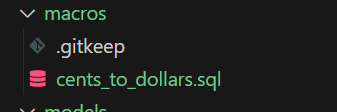
from stg\_payments

group by 1

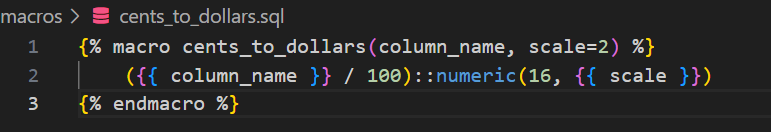
  );

Here you can see the jinja sql

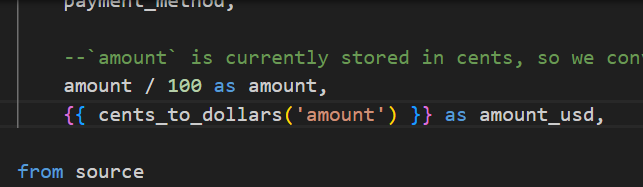
Macros:



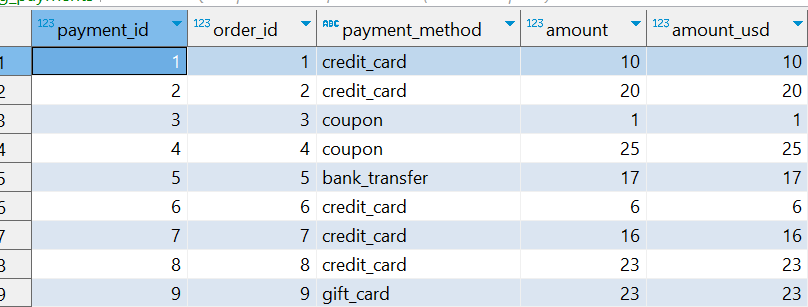
Under macro folder…I am creating the sql file



In Stg\_payments..model I am going to use this macro

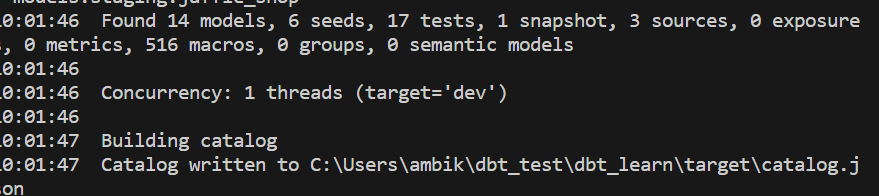


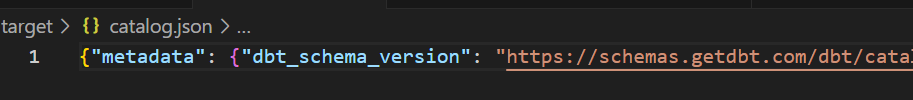
Now I rebuild this model



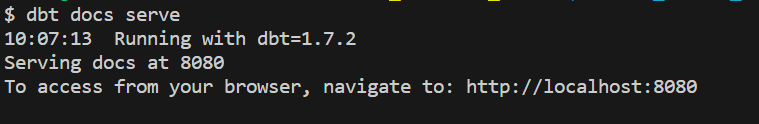
Amount…is the column calculated using Macro

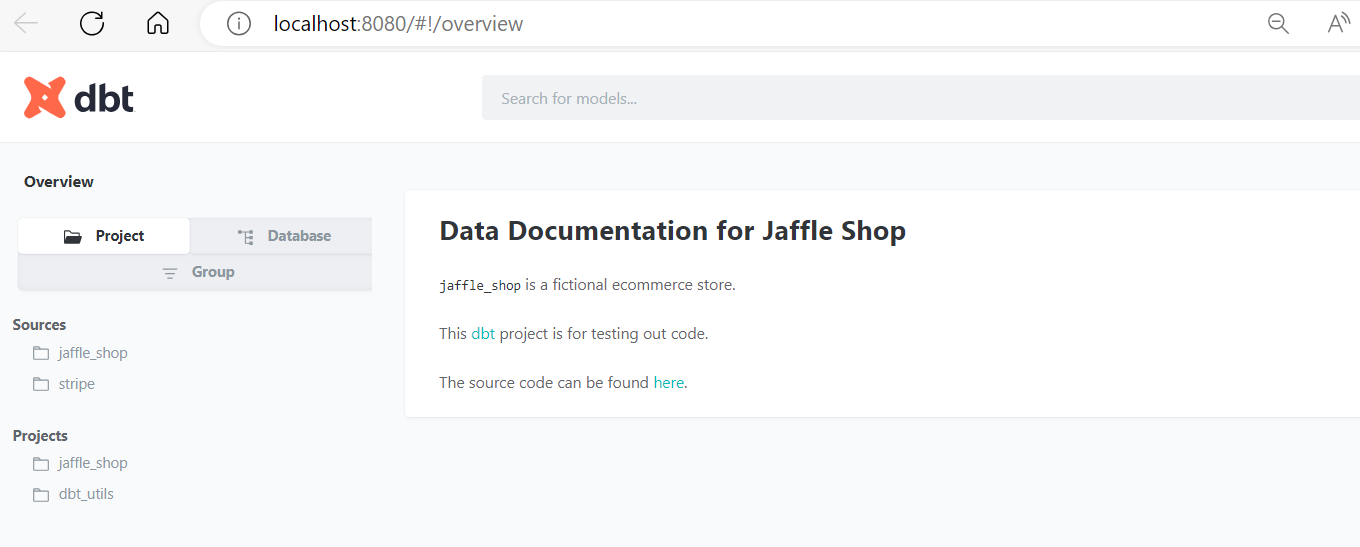
DBT DOCS GENERATE  

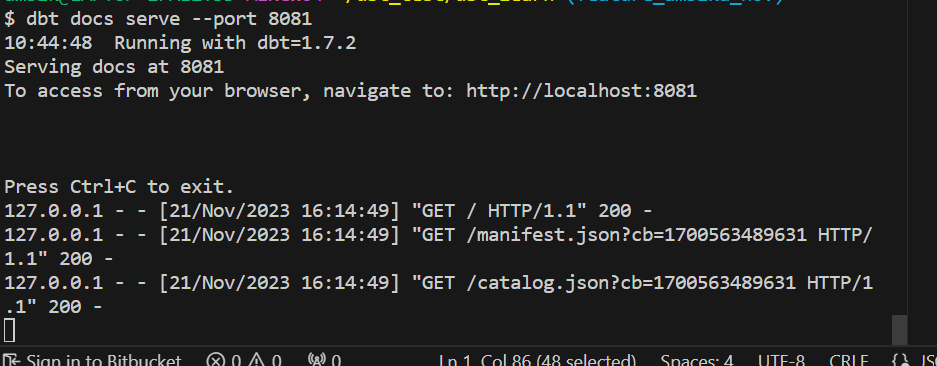


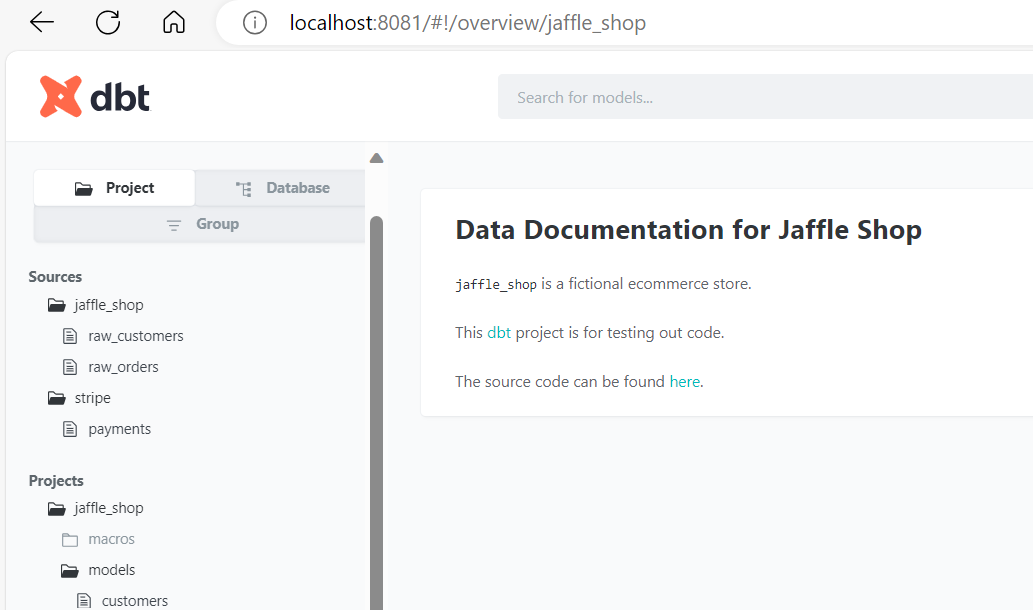
DBT DOCS SERVE:





Using –port flag

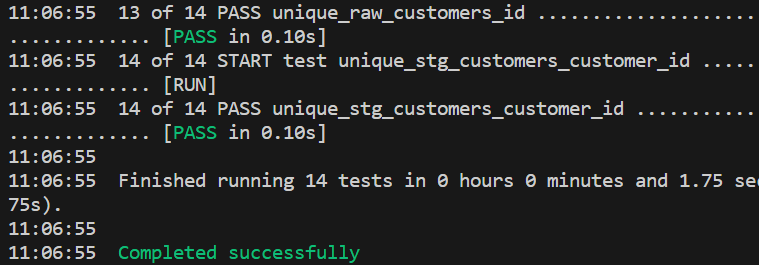




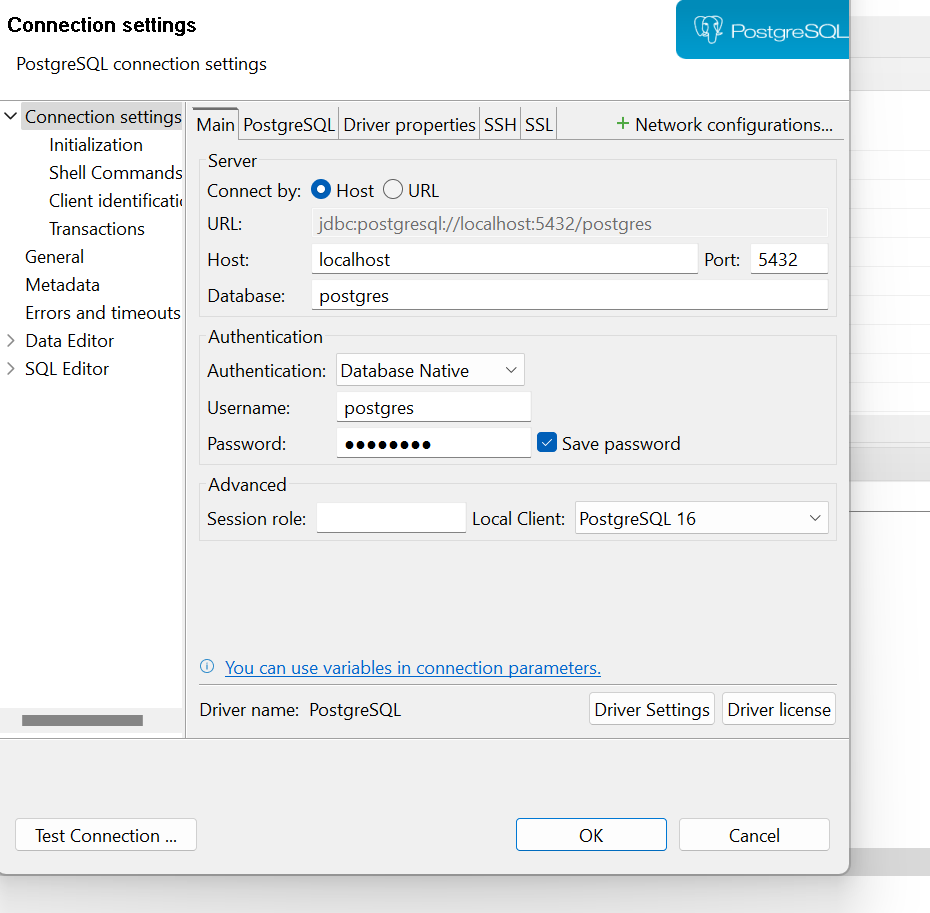
DBT Exclude

This command is going to remove stg\_orders from the test





url : activity-log-devdb.postgres.database.azure.com  
port: 5432  
Database: dbt\_training  
Username: dbt\_user  
Password: re23pUpHF7j9x!&



**Question 3.3:Can you describe the process of integrating dbt with a version control system?**

Integrating dbt with a version control system (VCS) allows for effective collaboration, code management, and tracking of changes in your dbt project.

Here's a step-by-step process to integrate dbt with a VCS:

* Set up a version control repository: Choose a VCS platform (e.g., Git, GitHub, GitLab) and create a new repository to store your dbt project's code.
* Initialize dbt as a Git repository: Navigate to your dbt project's root directory in your command-line interface or terminal.

Run the following commands:

git init

git add .

git commit -m "Initial commit"

* Connect your local repository to the remote repository: Link your local Git repository to the remote repository you created on the VCS platform.

Run the following command, replacing with the URL of your remote repository:

git remote add origin <remote-repo-url>

* Push your local repository to the remote repository: Upload your local dbt project code to the remote repository using the following command:

git push -u origin master

* Collaborate and manage changes: With the integration complete, you can now collaborate with your team on the dbt project. Each team member can clone the repository, make changes in their local environment, and use Git commands (git add, git commit, git push) to push their changes to the remote repository.
* Branching and pull requests: Utilize Git branching strategies to work on separate features or experiments. When ready to merge changes, team members can create pull requests on the VCS platform, allowing for code review and seamless integration of changes into the main branch.

By integrating dbt with a version control system, you establish a structured and collaborative development environment, enabling effective teamwork, change tracking, and the ability to roll back changes if necessary.