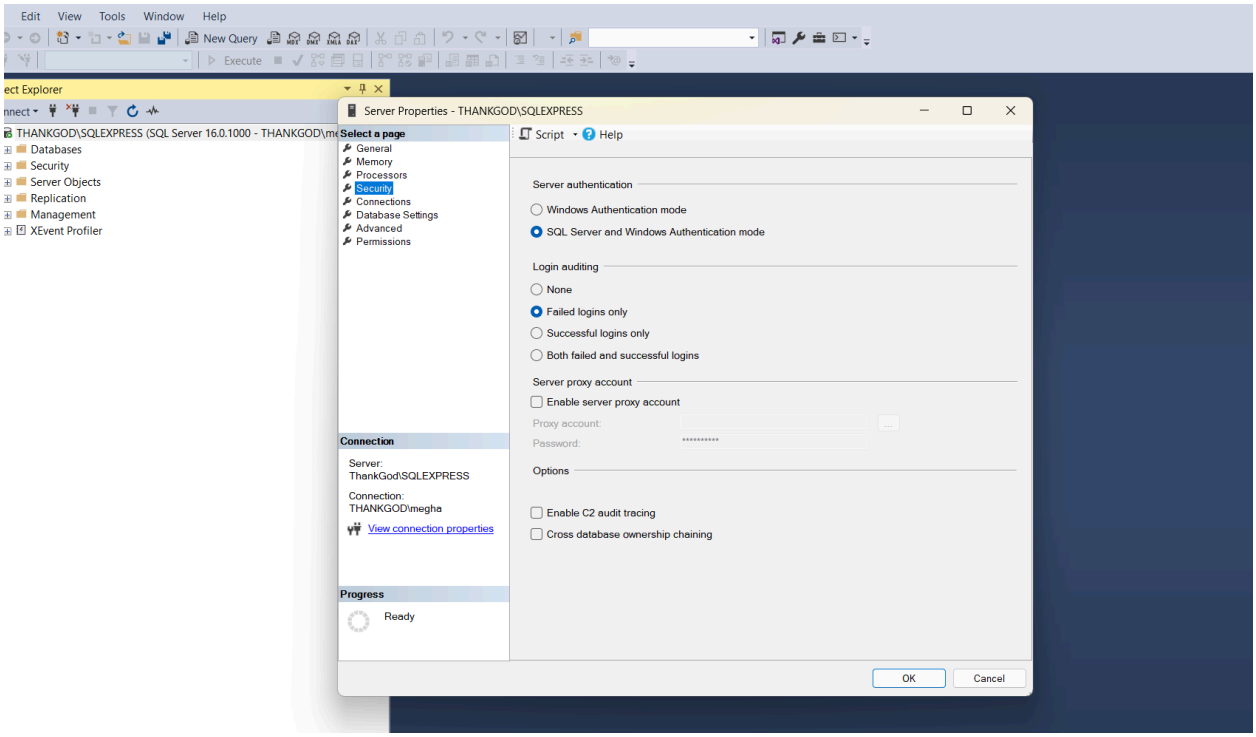
Requirements

- Works for python 3.12 and lower. It doesnt support for version 3.13 and above
- Make sure to configure the authentication in ssms as I am using Microsoft SQL



- Bypass the certificates if needed from the profiles.yml

Installation

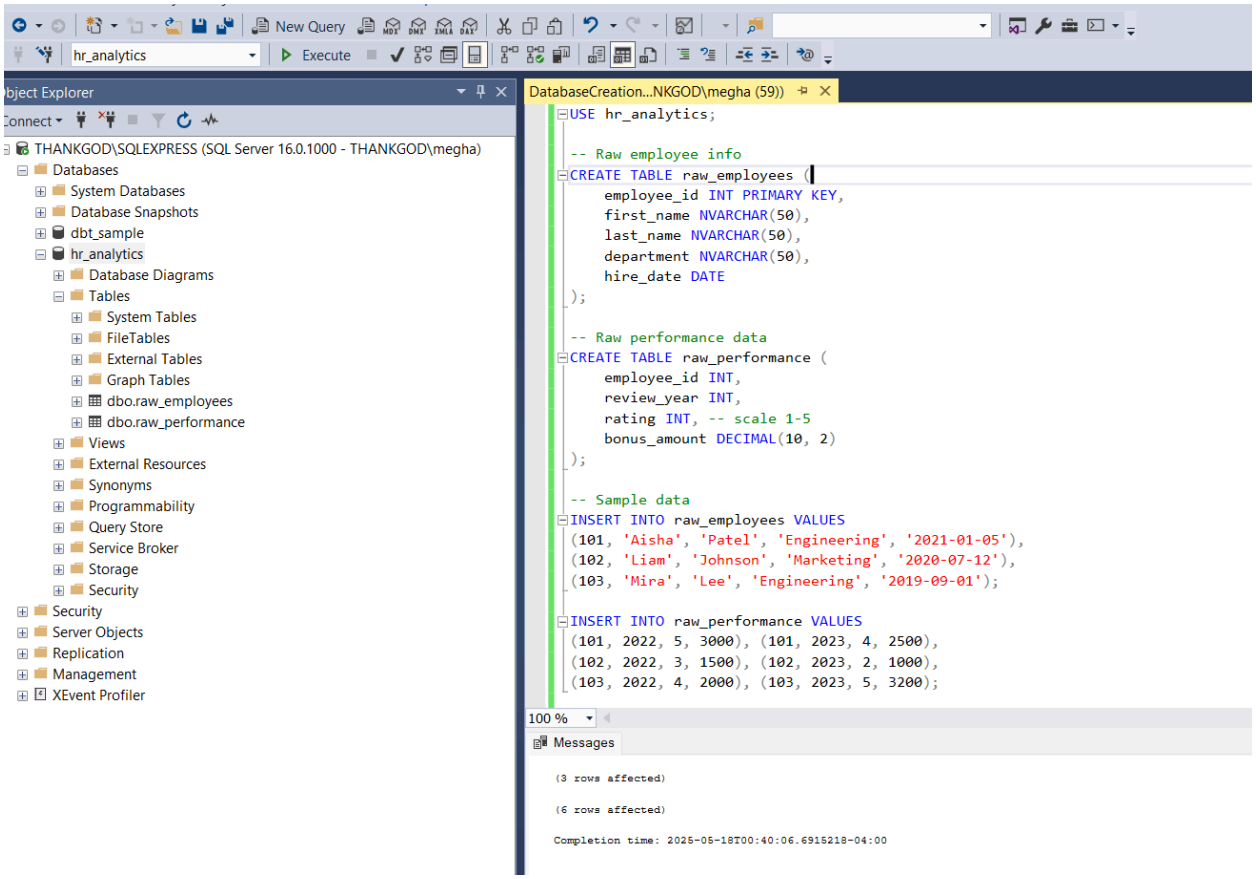
Make Sure Microsoft sql is installed and running

Please run this command in the terminal to install:

pip install dbt-core
pip install dbt-sqlserver

Step 1: Creating a database in SQL Studio

Created a database hr_analytics with two tables raw_employees and raw_performance



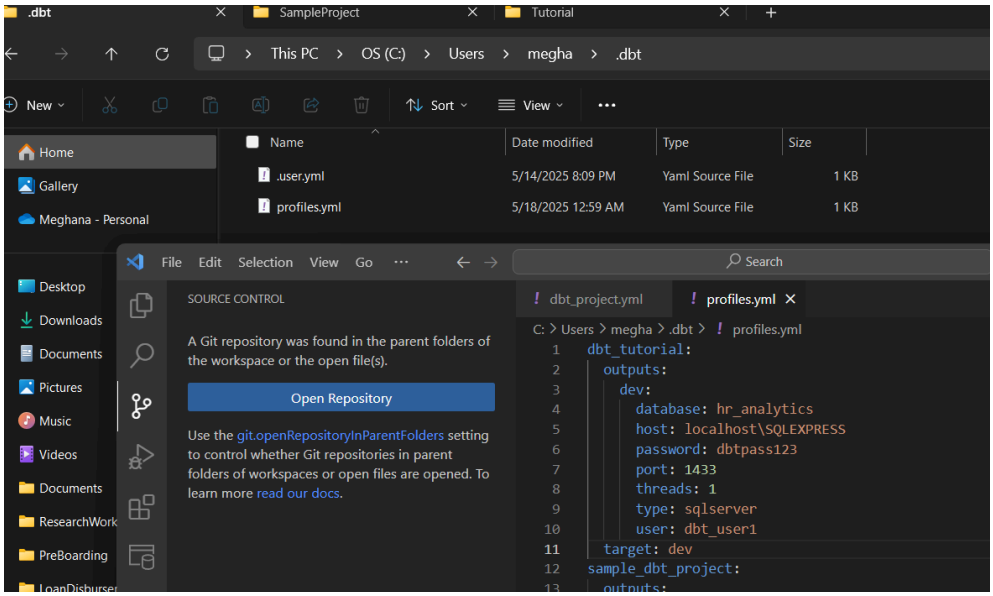
Step 2: Initialising a DBT project

Here I am using windows authentication. We can use SQL authentication as well, we have to create profiles for that

```
C:\Users\megha\Documents\ResearchWork\DBTProjects\Tutorial>dbt init dbt_tutorial
04:47:33  Running with dbt=1.8.9
04:47:33
Your new dbt project "dbt_tutorial" 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!

04:47:33  Setting up your profile.
Which database would you like to use?
[1] fabric
[2] sqlserver
```

dbt init project_name

As I am using windows authentication, just press enter when asked about user and password details.

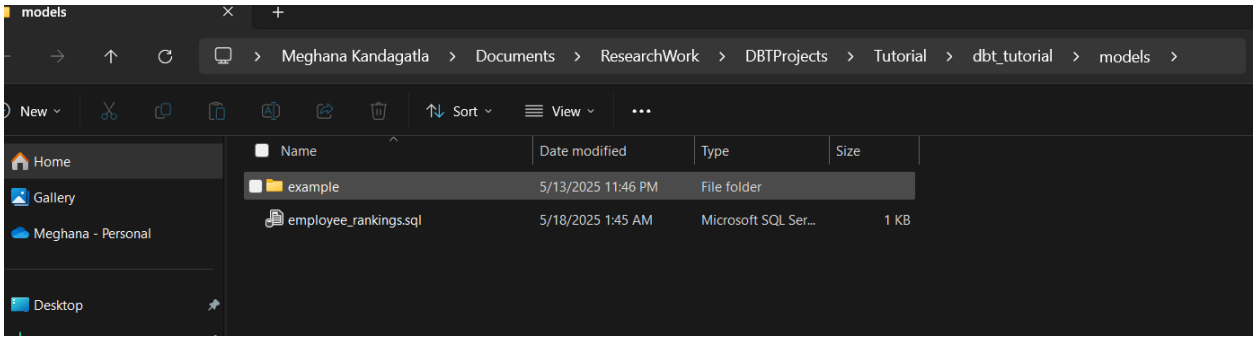


Editing the profiles.yml to windows authentication. Now navigate to the project folder and then run dbt debug to check if everything is working or not.

```
C:\Users\megha\Documents\ResearchWork\DBTProjects\Tutorial>cd dbt_tutorial

C:\Users\megha\Documents\ResearchWork\DBTProjects\Tutorial\dbt_tutorial>dbt debug
05:14:45  Running with dbt=1.8.9
05:14:45  dbt version: 1.8.9
05:14:45  python version: 3.12.10
05:14:45  python path: C:\Users\megha\AppData\Local\Programs\Python\Python312\python.exe
05:14:45  os info: Windows-11-10.0.26100-SP0
05:14:45  Using profiles dir at C:\Users\megha\.dbt
05:14:45  Using profiles.yml file at C:\Users\megha\.dbt\profiles.yml
05:14:45  Using dbt_project.yml file at C:\Users\megha\Documents\ResearchWork\DBTProjects\Tutorial\dbt_tutorial\dbt_project.yml
05:14:45  adapter type: sqlserver
05:14:45  adapter version: 1.8.7
05:14:46  Configuration:
05:14:46    profiles.yml file [OK found and valid]
05:14:46    dbt_project.yml file [OK found and valid]
05:14:46  Required dependencies:
05:14:46   - git [OK found]

05:14:46  Connection:
05:14:46    server: localhost\SQLEXPRESS
05:14:46    database: hr_analytics
05:14:46    schema: dbo
05:14:46    UID: None
05:14:46    client_id: None
05:14:46    authentication: Windows Login
05:14:46    encrypt: True
05:14:46    trust_cert: True
05:14:46    retries: 3
05:14:46    login_timeout: 0
05:14:46    query_timeout: 0
05:14:46    trace_flag: False
05:14:46    port: 1433
05:14:46  Registered adapter: sqlserver=1.8.7
05:14:46    Connection test: [OK connection ok]

05:14:46  All checks passed!

C:\Users\megha\Documents\ResearchWork\DBTProjects\Tutorial\dbt_tutorial>
```

Step 3: Creating Models and testing them

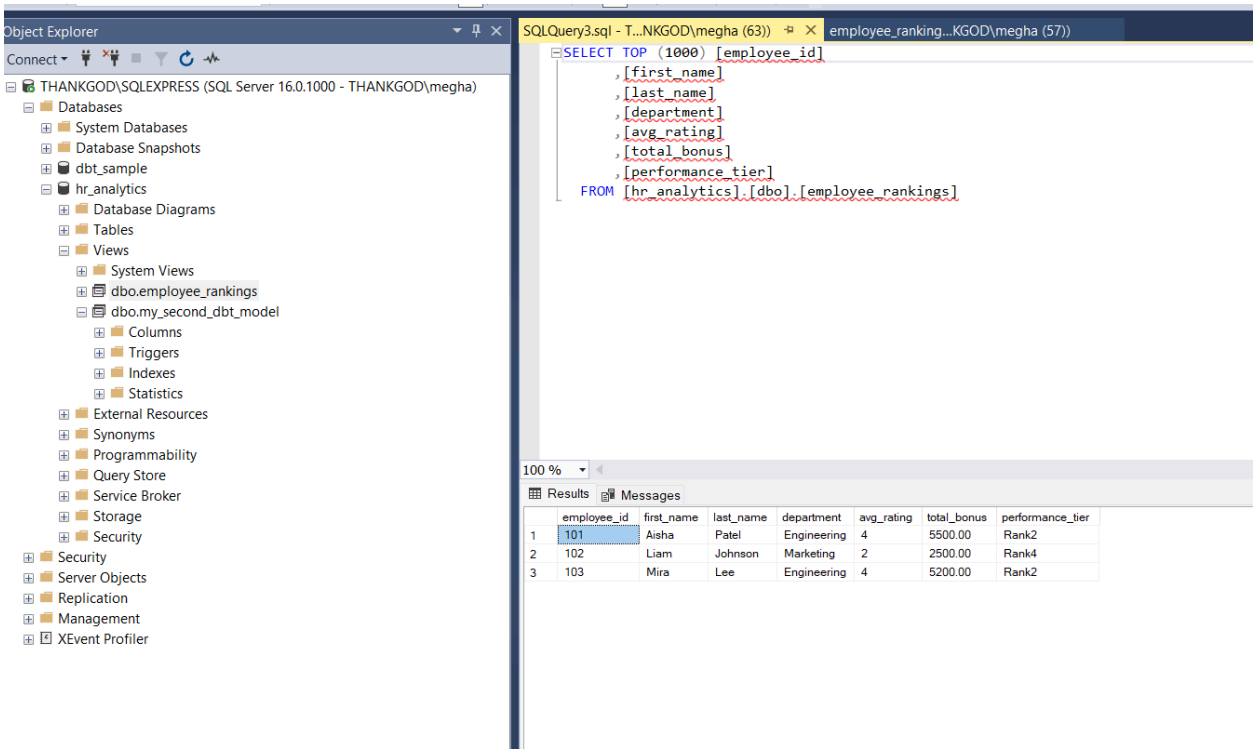
Creating a SQL script for ranking. I am placing that in the models folder. We can update the same in the schema.yml about the models.



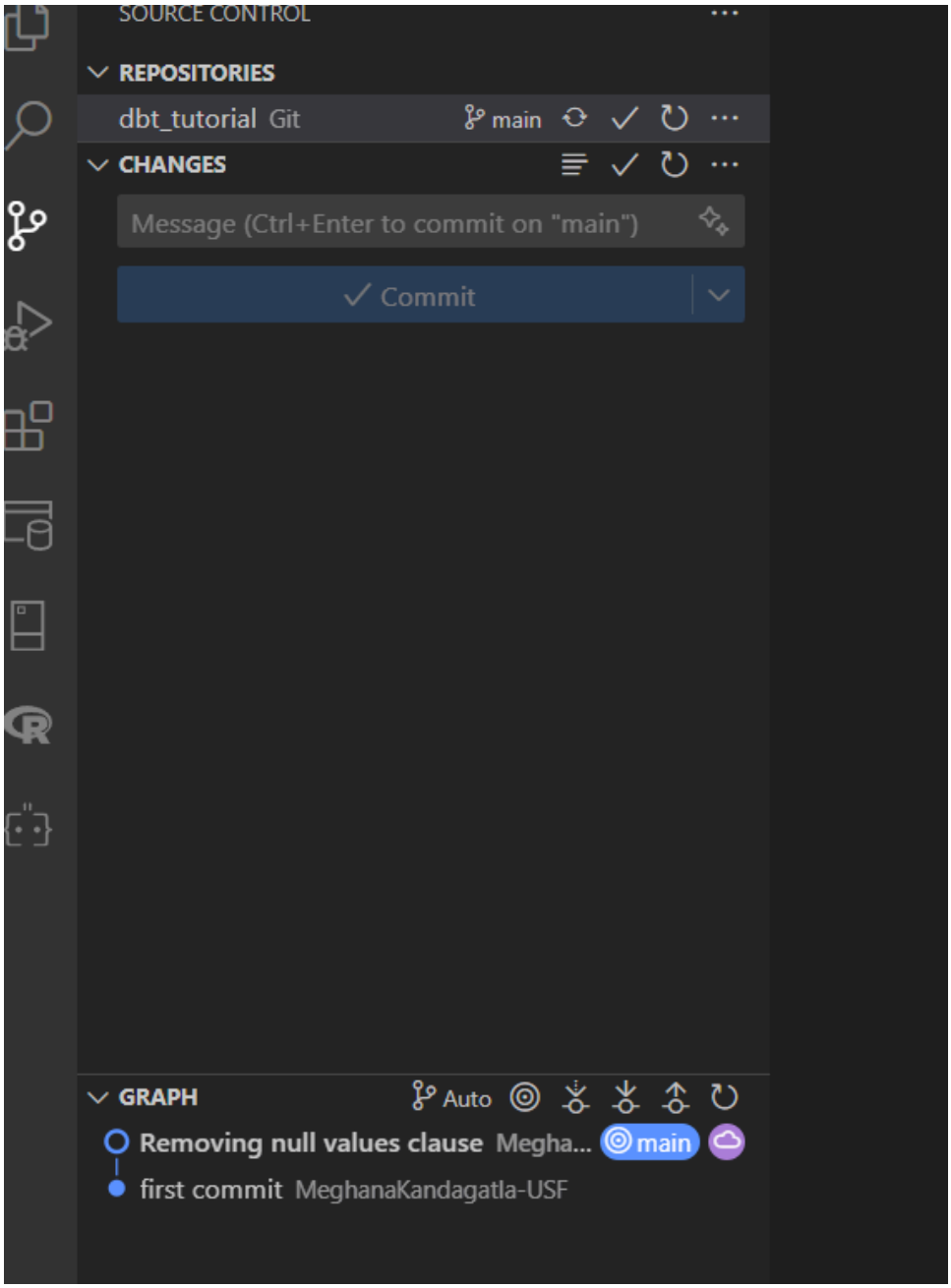
Using dbt run. I am running all the scripts in the models folder.

```
C:\Users\megha\Documents\ResearchWork\DBTProjects\Tutorial\dbt_tutorial>dbt run
05:53:07  Running with dbt=1.8.9
05:53:08  Registered adapter: sqlserver=1.8.7
05:53:08  Unable to do partial parsing because saved manifest not found. Starting full parse.
05:53:10  [WARNING]: Deprecated functionality
The `tests` config has been renamed to `data_tests`. Please see
https://docs.getdbt.com/docs/build/data-tests#new-data_tests-syntax for more
information.
05:53:10  Found 3 models, 7 data tests, 504 macros
05:53:10
05:53:11  Concurrency: 1 threads (target='dev')
05:53:11
05:53:11  1 of 3 START sql view model dbo.employee_rankings ............................... [RUN]
05:53:11  1 of 3 OK created sql view model dbo.employee_rankings .......................... [OK in 0.45s]
05:53:11  2 of 3 START sql table model dbo.my_first_dbt_model ............................. [RUN]
05:53:11  2 of 3 OK created sql table model dbo.my_first_dbt_model ......................... [OK in 0.23s]
05:53:11  3 of 3 START sql view model dbo.my_second_dbt_model ............................. [RUN]
05:53:11  3 of 3 OK created sql view model dbo.my_second_dbt_model ......................... [OK in 0.07s]
05:53:11
05:53:11  Finished running 2 view models, 1 table model in 0 hours 0 minutes and 0.97 seconds (0.97s).
05:53:12
05:53:12  Completed successfully
05:53:12
05:53:12  Done. PASS=3 WARN=0 ERROR=0 SKIP=0 TOTAL=3

C:\Users\megha\Documents\ResearchWork\DBTProjects\Tutorial\dbt_tutorial>
```

Now, I am checking ssms to check if the new table is created. It's saved as view.



Now testing the models using dbt run. I am creating a git repo.



```
C:\Users\megha\Documents\ResearchWork\DBTProjects\Tutorial\dbt_tutorial>dbt test
06:08:48  Running with dbt=1.8.9
06:08:49  Registered adapter: sqlserver=1.8.7
06:08:50  Found 3 models, 7 data tests, 504 macros
06:08:50
06:08:50  Concurrency: 1 threads (target='dev')
06:08:50
06:08:50  1 of 7 START test accepted_values_employee_rankings_performance_tier__Rank1__Rank2__Rank3__Rank4  [RUN]
06:08:50  1 of 7 PASS accepted_values_employee_rankings_performance_tier__Rank1__Rank2__Rank3__Rank4  [PASS in 0.10s]
06:08:50  2 of 7 START test not_null_employee_rankings_employee_id ..................... [RUN]
06:08:50  2 of 7 PASS not_null_employee_rankings_employee_id ........................... [PASS in 0.04s]
06:08:50  3 of 7 START test not_null_my_first_dbt_model_id ............................ [RUN]
06:08:50  3 of 7 PASS not_null_my_first_dbt_model_id ................................... [PASS in 0.02s]
06:08:50  4 of 7 START test not_null_my_second_dbt_model_id ........................... [RUN]
06:08:50  4 of 7 PASS not_null_my_second_dbt_model_id ................................... [PASS in 0.03s]
06:08:50  5 of 7 START test unique_employee_rankings_employee_id ..................... [RUN]
06:08:50  5 of 7 PASS unique_employee_rankings_employee_id ........................... [PASS in 0.04s]
06:08:50  6 of 7 START test unique_my_first_dbt_model_id ............................ [RUN]
06:08:50  6 of 7 PASS unique_my_first_dbt_model_id ..................................... [PASS in 0.04s]
06:08:50  7 of 7 START test unique_my_second_dbt_model_id ........................... [RUN]
06:08:50  7 of 7 PASS unique_my_second_dbt_model_id ..................................... [PASS in 0.03s]
06:08:50
06:08:50  Finished running 7 data tests in 0 hours 0 minutes and 0.44 seconds (0.44s).
06:08:50
06:08:50  Completed successfully
06:08:50
06:08:50  Done. PASS=7 WARN=0 ERROR=0 SKIP=0 TOTAL=7

C:\Users\megha\Documents\ResearchWork\DBTProjects\Tutorial\dbt_tutorial>
```

Step 4: Generating the documents

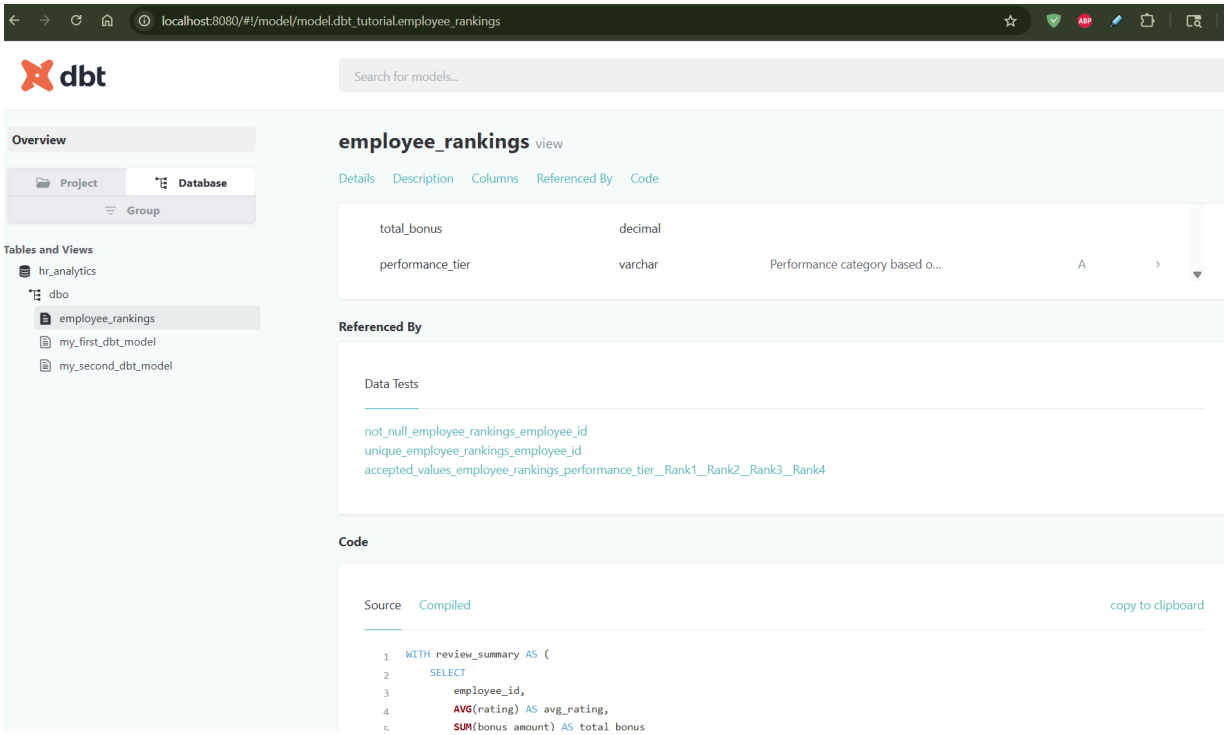
Using dbt commands we can create.

dbt docs generate
dbt docs serve

```
C:\Users\megha\Documents\ResearchWork\DBTProjects\Tutorial\dbt_tutorial>dbt docs generate
06:17:11  Running with dbt=1.8.9
06:17:12  Registered adapter: sqlserver=1.8.7
06:17:12  Found 3 models, 7 data tests, 504 macros
06:17:12
06:17:13  Concurrency: 1 threads (target='dev')
06:17:13
06:17:13  Building catalog
06:17:13  Catalog written to C:\Users\megha\Documents\ResearchWork\DBTProjects\Tutorial\dbt_tutorial\target\catalog.json

C:\Users\megha\Documents\ResearchWork\DBTProjects\Tutorial\dbt_tutorial>dbt docs serve
06:17:30  Running with dbt=1.8.9
Serving docs at 8080
To access from your browser, navigate to: http://localhost:8080



Press Ctrl+C to exit.
127.0.0.1 - - [18/May/2025 02:17:31] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [18/May/2025 02:17:32] "GET /manifest.json?cb=1747549052393 HTTP/1.1" 200 -
127.0.0.1 - - [18/May/2025 02:17:32] "GET /catalog.json?cb=1747549052393 HTTP/1.1" 200 -
```

see all of the models that are used to build, or are built from, the model you're exploring.

This generated me



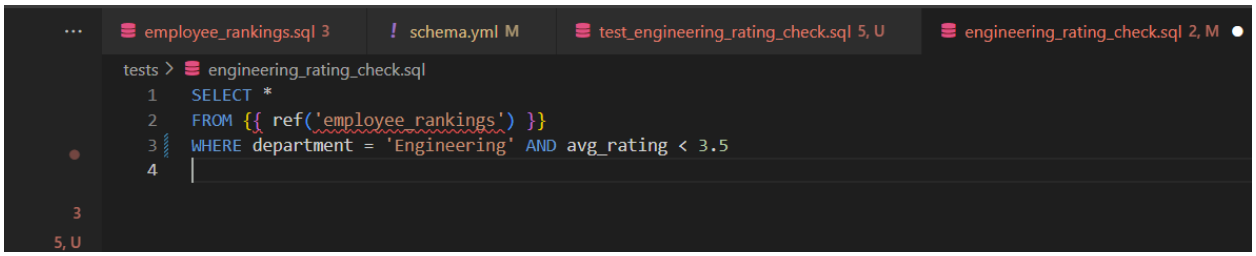
Here I can navigate through the database and the tables. All the data tests can be reused again using the commands in the terminal.

Step 5: Creating a custom test

how dbt tests are designed to work:

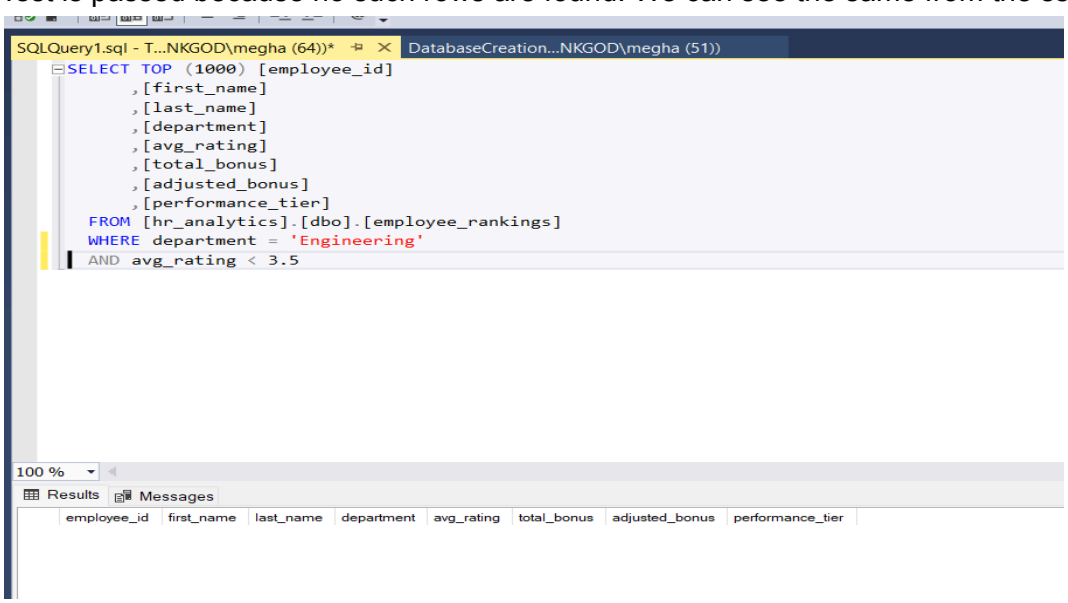
- Tests pass when they return zero rows (no violations found)
- Tests fail when they return one or more rows (violations found)

I am creating a custom test that checks if the department is engineering or not and also if avg_rating < 3.5 or not. If such rows are found then test is failed or else test is passed



```
PS C:\Users\megha\Documents\ResearchWork\DBTProjects\Tutorial\dbt_tutorial> dbt test
15:21:23  Running with dbt=1.8.9
15:21:23  Registered adapter: sqlserver=1.8.7
15:21:24  Found 3 models, 9 data tests, 506 macros
15:21:24
15:21:24  Concurrency: 1 threads (target='dev')
15:21:24
15:21:24  1 of 9 START test accepted_values_employee_rankings_performance_tier__Rank1__Rank2__Rank3__Rank4  [RUN]
15:21:24  1 of 9 PASS accepted_values_employee_rankings_performance_tier__Rank1__Rank2__Rank3__Rank4  [PASS in 0.08s]
15:21:24  2 of 9 START test engineering_rating_check ..................................... [RUN]
15:21:24  2 of 9 PASS engineering_rating_check ........................................... [PASS in 0.02s]
15:21:24  3 of 9 START test engineering_rating_check_macro_employee_rankings_ ........... [RUN]
15:21:24  3 of 9 PASS engineering_rating_check_macro_employee_rankings_ .................. [PASS in 0.06s]
15:21:24  4 of 9 START test not_null_employee_rankings_employee_id ....................... [RUN]
15:21:24  4 of 9 PASS not_null_employee_rankings_employee_id ............................ [PASS in 0.02s]
15:21:24  5 of 9 START test not_null_my_first_dbt_model_id .............................. [RUN]
15:21:24  5 of 9 PASS not_null_my_first_dbt_model_id ..................................... [PASS in 0.02s]
15:21:24  6 of 9 START test not_null_my_second_dbt_model_id ............................. [RUN]
15:21:24  6 of 9 PASS not_null_my_second_dbt_model_id ..................................... [PASS in 0.02s]
15:21:24  7 of 9 START test unique_employee_rankings_employee_id ......................... [RUN]
15:21:24  7 of 9 PASS unique_employee_rankings_employee_id ............................... [PASS in 0.03s]
15:21:24  8 of 9 START test unique_my_first_dbt_model_id ................................ [RUN]
15:21:24  8 of 9 PASS unique_my_first_dbt_model_id ....................................... [PASS in 0.03s]
15:21:24  9 of 9 START test unique_my_second_dbt_model_id ............................... [RUN]
15:21:24  9 of 9 PASS unique_my_second_dbt_model_id ..................................... [PASS in 0.03s]
15:21:24
15:21:24  Finished running 9 data tests in 0 hours 0 minutes and 0.41 seconds (0.41s).
15:21:24
15:21:24  Completed successfully
15:21:24
15:21:24  Done. PASS=9 WARN=0 ERROR=0 SKIP=0 TOTAL=9
PS C:\Users\megha\Documents\ResearchWork\DBTProjects\Tutorial\dbt_tutorial>
```

Test is passed because no such rows are found. We can see the same from the ssms as well.

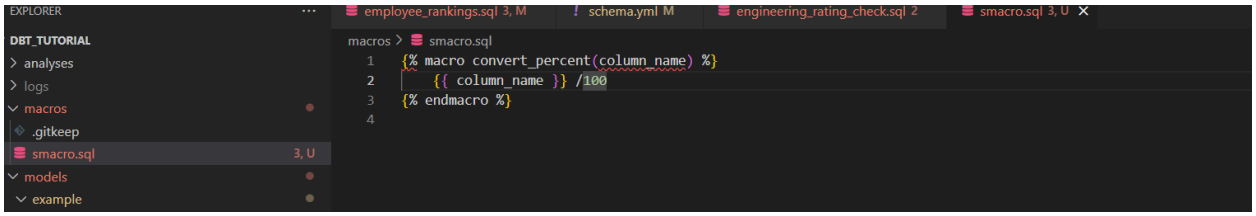


As this file is in the tests folder, it automatically performs this test. If a test has to be included in the schema it has to be defined as a macro.

Step 6: Macros

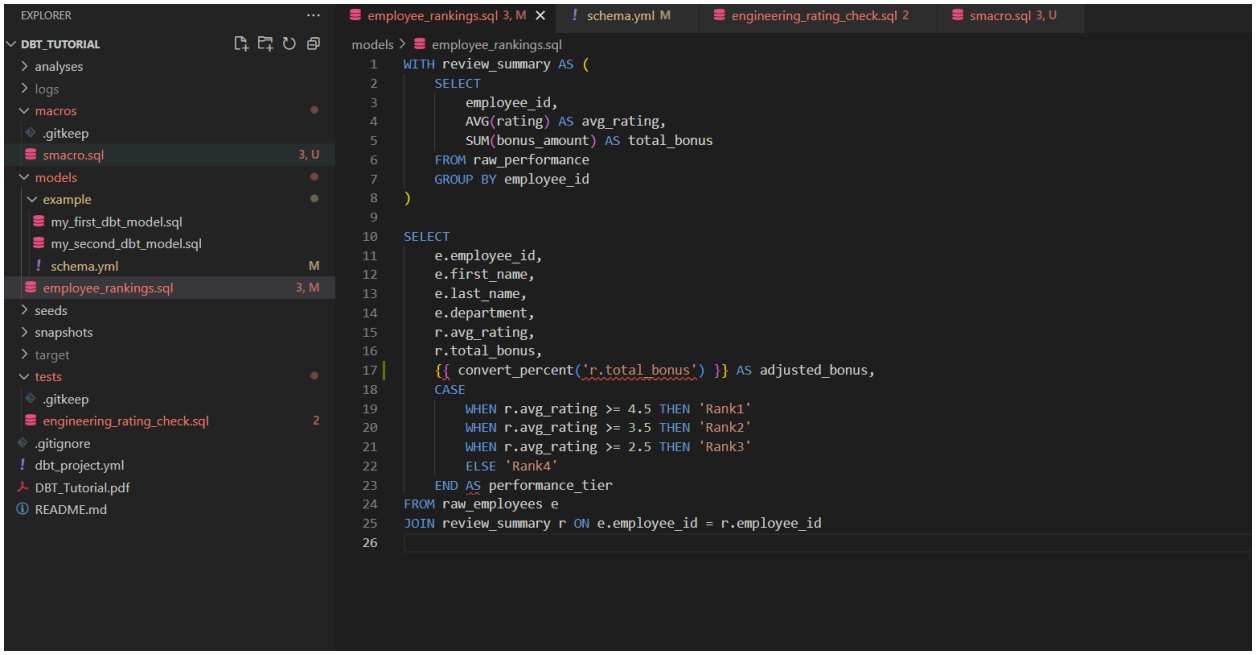
Macro is a reusable block of logic.

Here I am adding a simple macro that converts into percent to the given column.



As we can see all of these macros are kept in macros folder

Now let's look into how we are going to use this macro in the models.



Here we can see that we have replaced the column name as r.total_bonus in the macro and saved it as adjusted_bonus

SQLQuery1.sql - T...NKGOD\megha (58))

```sql
SELECT TOP (1000) [employee_id]
      ,[first_name]
      ,[last_name]
      ,[department]
      ,[avg_rating]
      ,[total_bonus]
      ,[adjusted_bonus]
      ,[performance_tier]
  FROM [hr_analytics].[dbo].[employee_rankings]
```

100 %

Results | Messages

	employee_id	first_name	last_name	department	avg_rating	total_bonus	adjusted_bonus	performance_tier
1	101	Aisha	Patel	Engineering	4	5500.00	55.000000	Rank2
2	102	Liam	Johnson	Marketing	2	2500.00	25.000000	Rank4
3	103	Mira	Lee	Engineering	4	5200.00	52.000000	Rank2

Here we can see that adjusted_bonus is created.

Step 7 : Macros for Tests

EXPLORER

DBT_TUTORIAL
- analyses
- logs
- macros
  - .gitkeep
  - smacro.sql 3
  - test_engineering_rating_check.sql 5, U
- models

employee_rankings.sql 3 | schema.yml M | test_engineering_rating_check.sql 5, U | eng

macros > test_engineering_rating_check.sql

```sql
{% test engineering_rating_check_macro(model) %}
    SELECT *
    FROM {{ model }}
    WHERE department = 'Engineering' AND avg_rating < 3.5
{% endtest %}
```

We can define tests like this in the macros folder and then use in the schema.yml file.

```
employee_rankings.sql 3   ! schema.yml M ×   test_engineering_rating_check.sql 5, U
models > example > ! schema.yml
models:
  - name: my_second_dbt_model

  - name: employee_rankings
    description: "Aggregates employee performance and categorizes them
    columns:
      - name: employee_id
        description: "Unique employee ID"
        tests:
          - not_null
          - unique

      - name: avg_rating
        description: "Average of all yearly performance ratings"

      - name: performance_tier
        description: "Performance category based on average rating"
        tests:
          - accepted_values:
              values: ['Rank1', 'Rank2', 'Rank3', 'Rank4']

    data_tests:
      - engineering_rating_check_macro: {}
```

```
15:21:23  Registered adapter: sqlserver=1.8.7
15:21:24  Found 3 models, 9 data tests, 506 macros
15:21:24
15:21:24  Concurrency: 1 threads (target='dev')
15:21:24
15:21:24  1 of 9 START test accepted_values_employee_rankings_performance_tier__Rank1__Rank2__Rank3__Rank4  [RUN]
15:21:24  1 of 9 PASS accepted_values_employee_rankings_performance_tier__Rank1__Rank2__Rank3__Rank4  [PASS in 0.08s]
15:21:24  2 of 9 START test engineering_rating_check ..................................... [RUN]
15:21:24  2 of 9 PASS engineering_rating_check ..........................................  [PASS in 0.02s]
15:21:24  3 of 9 START test engineering_rating_check_macro_employee_rankings_ ........... [RUN]
15:21:24  3 of 9 PASS engineering_rating_check_macro_employee_rankings_ .................  [PASS in 0.06s]
15:21:24  4 of 9 START test not_null_employee_rankings_employee_id ....................... [RUN]
15:21:24  4 of 9 PASS not_null_employee_rankings_employee_id ............................  [PASS in 0.02s]
15:21:24  5 of 9 START test not_null_my_first_dbt_model_id ............................... [RUN]
15:21:24  5 of 9 PASS not_null_my_first_dbt_model_id ....................................  [PASS in 0.02s]
15:21:24  6 of 9 START test not_null_my_second_dbt_model_id ............................... [RUN]
15:21:24  2 of 9 PASS engineering_rating_check ..........................................  [PASS in 0.02s]
15:21:24  3 of 9 START test engineering_rating_check_macro_employee_rankings_ ........... [RUN]
15:21:24  3 of 9 PASS engineering_rating_check_macro_employee_rankings_ .................  [PASS in 0.06s]
15:21:24  4 of 9 START test not_null_employee_rankings_employee_id ....................... [RUN]
15:21:24  4 of 9 PASS not_null_employee_rankings_employee_id ............................  [PASS in 0.02s]
15:21:24  5 of 9 START test not_null_my_first_dbt_model_id ............................... [RUN]
15:21:24  5 of 9 PASS not_null_my_first_dbt_model_id ....................................  [PASS in 0.02s]
```

We can see the same test performing well as a macro and the normal test from the tests folder.

Understanding Jinja

1. Setting the variables

Jinja Code	Corresponding SQL code

{% set bonus_multiplier = 1.1 %}
 	
select
  employee_id,
  round(bonus_amount * {{ bonus_multiplier }}, 2) as adjusted_bonus
from {{ source('dbt_tutorial', 'raw_performance') }}
	select
	  employee_id,
	  round(bonus_amount * 1.1, 2) as adjusted_bonus
	from dbo.raw_performance

- **set** is used to store values that you want to reuse, like constants or column lists.
- {% set %} assigns a value to a variable.
- {{ bonus_multiplier }} inserts the variable in SQL.
- {{ source('dbt_tutorial', 'raw_performance') }} refers to the database that I will be using

2. Macro with Arguments

Jinja Code	Corresponding SQL code

{% macro adjust_bonus(col) %}
  round({{ col }} * 1.1, 2)
{% endmacro %}
 	
Usage in Model:
select
  employee_id,
  {{ adjust_bonus('bonus_amount') }} as adjusted_bonus
from {{ source('dbt_tutorial', 'raw_performance') }}
	select
	  employee_id,
	  round(bonus_amount * 1.1, 2) as adjusted_bonus
	from dbo.raw_performance

These macros define a reusable function that takes a column name and applies logic to do a certain task.

- Arguments let you pass values like column names.

- Use {{ adjust_bonus('column') }} to call it.

3. If else

Jinja Code	Corresponding SQL code

{% set selected_department = 'Engineering' %}
 	
select *
from {{ source('dbt_tutorial', 'raw_employees') }}
{% if selected_department == 'Engineering' %}
  where hire_date >= '2020-01-01'
{% else %}
  where hire_date >= '2018-01-01'
{% endif %}
	select *
	from dbo.raw_employees
	where salary >
	  case
	  when department = 'Engineering' then 70000
	  else 50000
	  end

{% set selected_department = 'Engineering' %} : To set department
{{ source('dbt_tutorial', 'raw_employees') }}: To select the table from the database
{% if selected_department == 'Engineering' %} {% else %} {% endif %} : writing the if else statements

4. Loop

Jinja Code	Corresponding SQL code

{% set cols = ['employee_id', 'first_name', 'last_name', 'department'] %}
 	
select
  {% for col in cols %}
  {{ col }}{% if not loop.last %}, {% endif %}
  {% endfor %}
from {{ source('dbt_tutorial', 'raw_employees') }}
	select
	  employee_id, first_name, last_name, department
	from dbo.raw_employees

Loops over a list of columns and prints them.

- for loops : {% for col in cols %} {% endfor %}
- loop.last avoids the last comma {% if not loop.last %} {% endif %}
- To set a list of names: {% set cols = ['employee_id', 'first_name', 'last_name', 'department'] %}