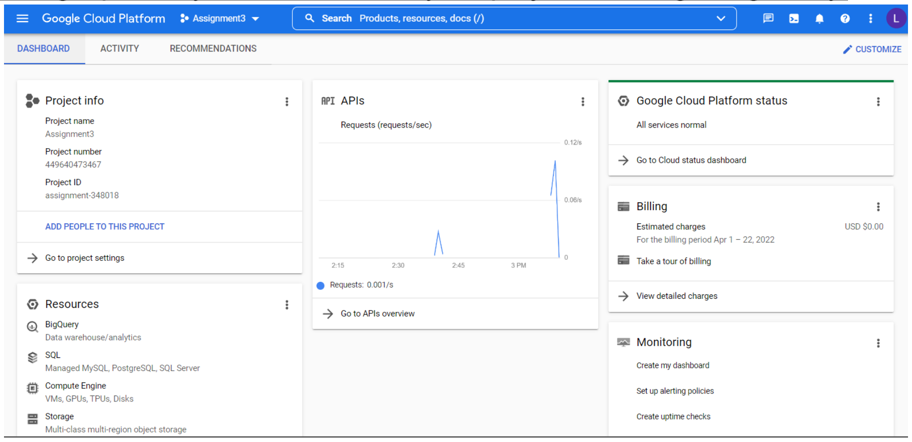
Lorena Vasquez
Computer Information Systems 4400
Homework Assignment 3

CIS_4400_assignment3_Vasquez_Lorena

Getting Started Tutorial:

a) Section: Setting Up after you have created your project in Google BigQuery.



Project info
Project name
Assignment3
Project number
449640473467
Project ID
assignment-348018
ADD PEOPLE TO THIS PROJECT
Go to project settings

Resources
BigQuery
Data warehouse/analytics
SQL
Managed MySQL, PostgreSQL, SQL Server
Compute Engine
VMs, GPUs, TPUs, Disks
Storage
Multi-class multi-region object storage

Google Cloud Platform status
All services normal
Go to Cloud status dashboard

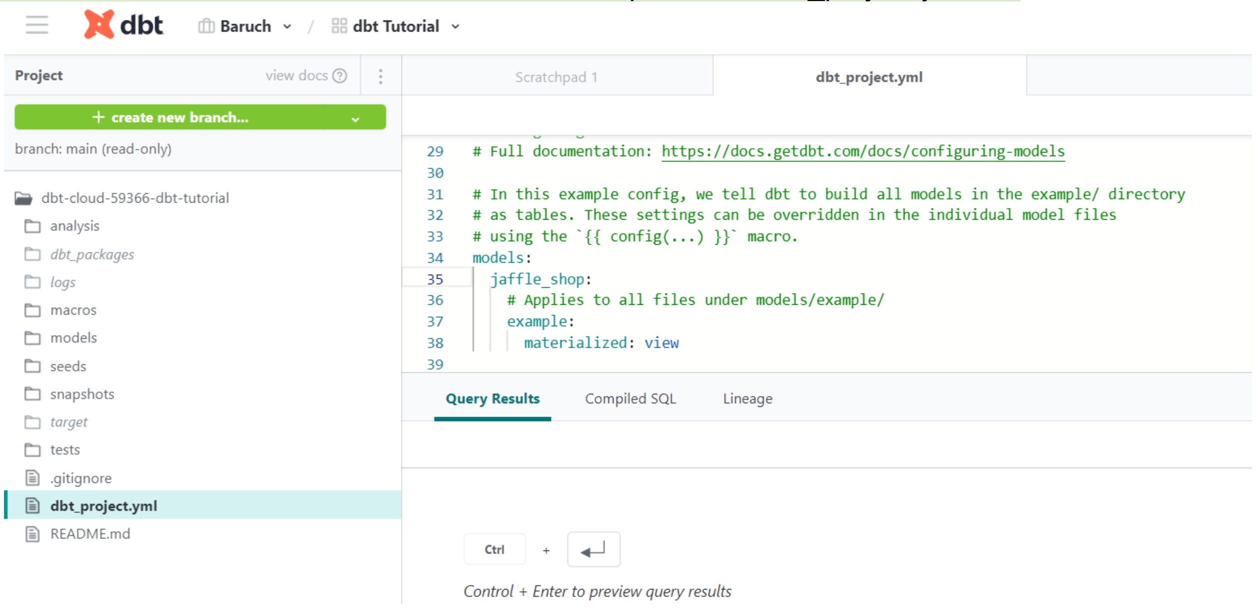
Billing
Estimated charges USD $0.00
For the billing period Apr 1 – 22, 2022
Take a tour of billing
View detailed charges

Monitoring
Create my dashboard
Set up alerting policies
Create uptime checks

Following the instructions, I first created a project in BigQuery.

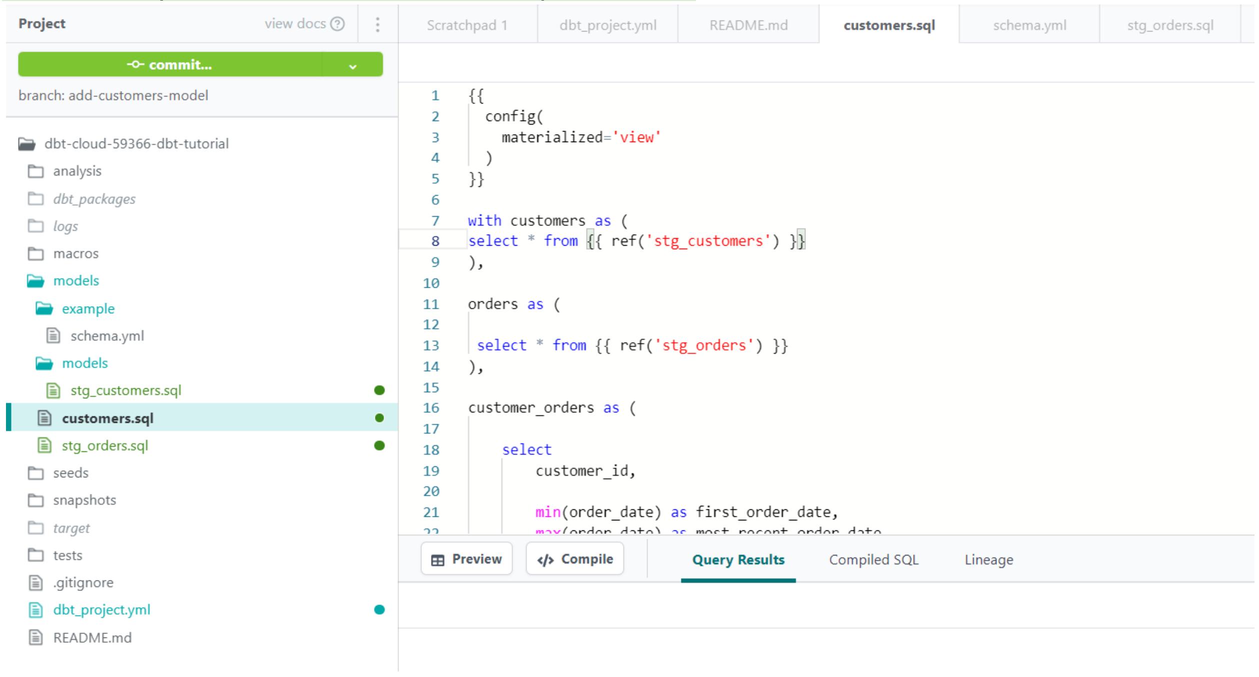
b) Section: Create a Project after you commit your changes in your dbt Project. Follow the instructions for dbt Cloud (not dbt CLI)

In this section, I created the cloud account and opened the dbt_project.yml file.



```
# 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 tables. These settings can be overridden in the individual model files
# using the `{{ config(...) }}` macro.
models:
  jaffle_shop:
    # Applies to all files under models/example/
    example:
      materialized: view

```

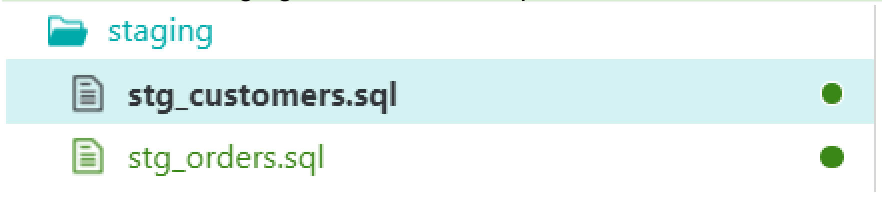
Here in this picture is where customers.sql is created.



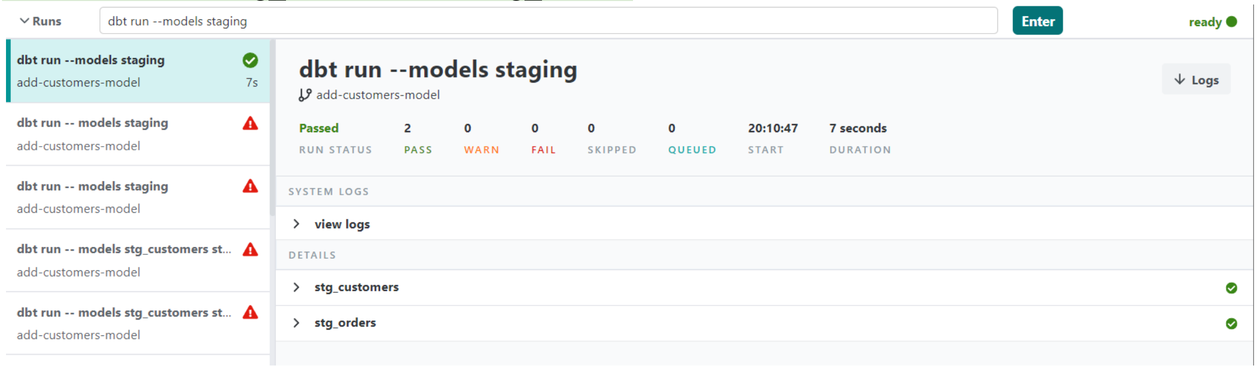
```
{{
  config(
    materialized='view'
  )
}}

with customers as (
select * from {{ ref('stg_customers') }}
),

orders as (

 select * from {{ ref('stg_orders') }}
),

customer_orders as (

    select
        customer_id,

        min(order_date) as first_order_date,
        max(order_date) as most_recent_order_date
```

c) Section: Build Your First Model after you run the staging models.

Afterwards the staging models for the sql files are then created. One for customers and orders.



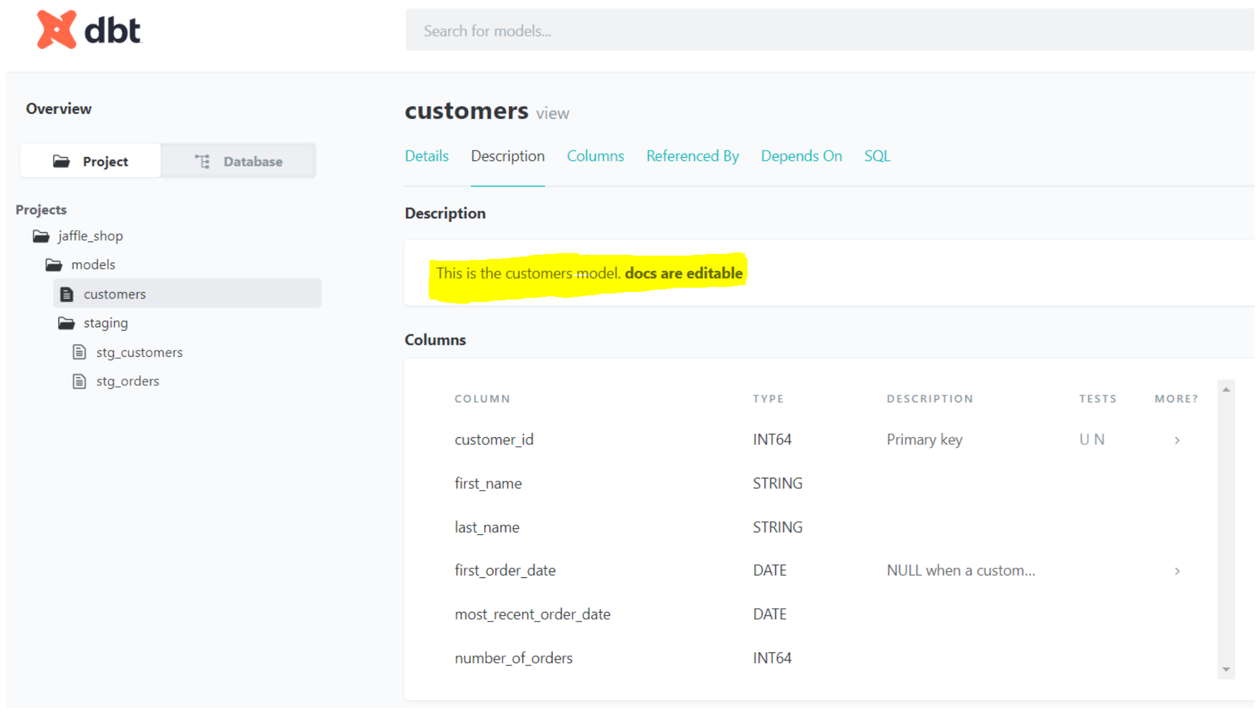
When running dbt run for the models staging, I did run into some errors, once fixed it then ran the model for the stg_customers and stg_orders.



```
dbt run --models staging

dbt run --models staging            7s
add-customers-model
dbt run -- models staging
add-customers-model
dbt run -- models staging
add-customers-model
dbt run -- models stg_customers st...
add-customers-model
dbt run -- models stg_customers st...
add-customers-model

dbt run --models staging
add-customers-model

Passed       2     0     0     0        0        20:10:47   7 seconds
RUN STATUS   PASS  WARN  FAIL  SKIPPED  QUEUED   START      DURATION

SYSTEM LOGS
> view logs
DETAILS
> stg_customers
> stg_orders
```

d) Section: Test and Document your project after you use the docs block to add a Markdown description to your model

In this section, I am able to edit the docs for the customers table, and I was able to edit it to my liking. Once running dbt docs generate, I am able to see the documentation for all the tables that I would like to see. It also enables the user to see lineage in the project, data types within the columns, any SQL code as well.
Descriptions are editable **



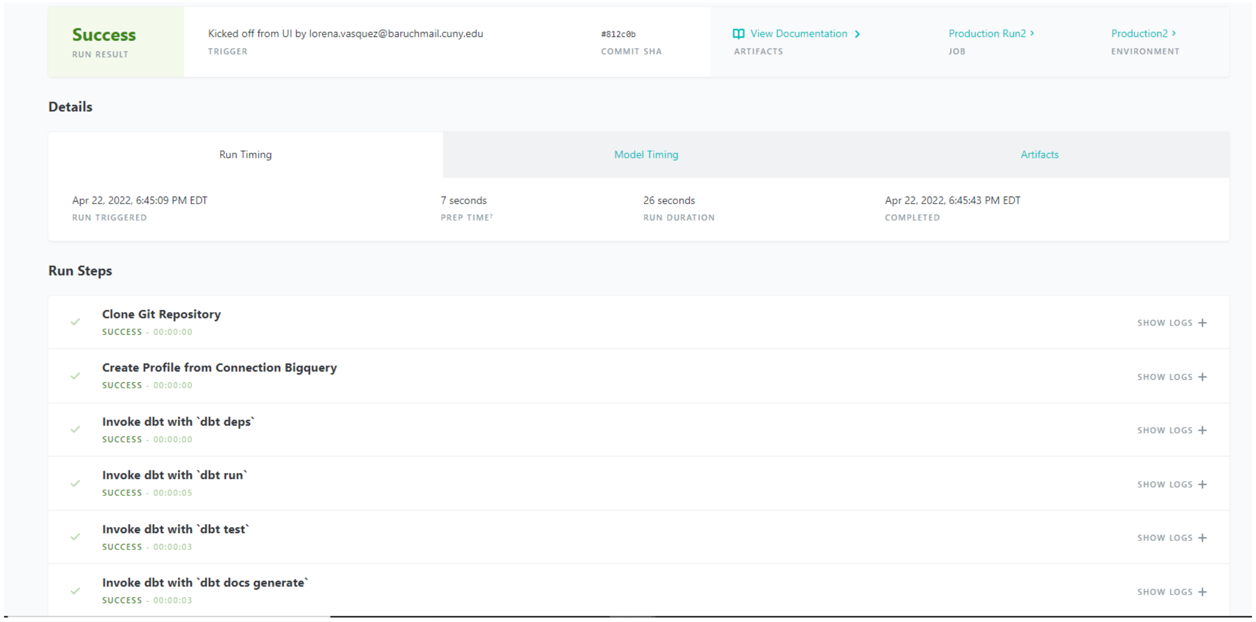
Description

This is the customers model. **docs are editable**

Columns

COLUMN	TYPE	DESCRIPTION	TESTS	MORE?
customer_id	INT64	Primary key	U N	›
first_name	STRING			
last_name	STRING			
first_order_date	DATE	NULL when a custom...		›
most_recent_order_date	DATE			
number_of_orders	INT64			

e) Section: Deploy your project after you create and run a job.
When running a job, you have to create a deployment and you want it to be different from your original repository. Running dbt in production is setting an automatic system to run the dbt job. You can enable it to do it on a schedule, but for the sake of this project we only did a preview. Within there I can change the setting to then run the dbt commands that I would like:
In this project we did:
Dbt test
Dbt run


Success
RUN RESULT
Kicked off from UI by lorena.vasquez@baruchmail.cuny.edu
TRIGGER
#812c0b
COMMIT SHA
View Documentation
ARTIFACTS
Production Run2
JOB
Production2
ENVIRONMENT

Details

Apr 22, 2022, 6:45:09 PM EDT
RUN TRIGGERED
7 seconds
PREP TIME
26 seconds
RUN DURATION
Apr 22, 2022, 6:45:43 PM EDT
COMPLETED

Run Steps

Clone Git Repository
SUCCESS - 00:00:00
Create Profile from Connection Bigquery
SUCCESS - 00:00:00
Invoke dbt with `dbt deps`
SUCCESS - 00:00:00
Invoke dbt with `dbt run`
SUCCESS - 00:00:05
Invoke dbt with `dbt test`
SUCCESS - 00:00:03
Invoke dbt with `dbt docs generate`
SUCCESS - 00:00:03

BigQuery

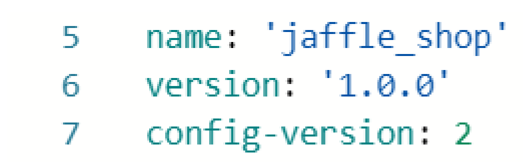
BigQuery supports public data sets that can be directly queried. The data is publicly available with the following select statements. This will be important for reference in the `models` and `sources` modules.

```
select * from `dbt-tutorial.jaffle_shop.customers`;
select * from `dbt-tutorial.jaffle_shop.orders`;
select * from `dbt-tutorial.stripe.payment`;
```

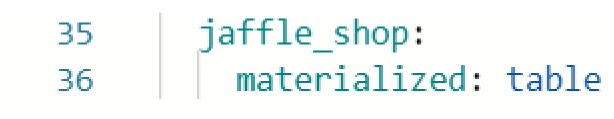
First Course: dbt Fundamentals Course

a) COURSE 1 // Models – Practice and Exemplar

❖ **Practice:**

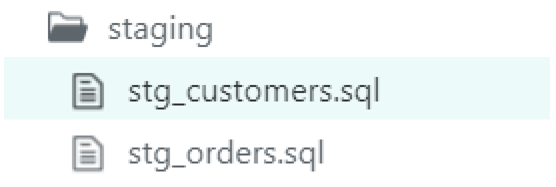
- **Quick Project Polishing:**
- For each line under my dbt_project.yml file that the practice said to change, I changed line 5 and 35 to 'jaffle_shop'. This is then telling the file that it will be the jaffle_shop that we are searching for.
    - 

```
5    name: 'jaffle_shop'
6    version: '1.0.0'
7    config-version: 2
```

    - 

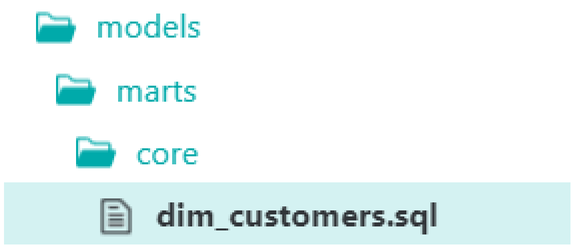
```
35      jaffle_shop:
36        materialized: table
```

- **Staging models:**
- For this part of the practice, I created a new folder under the models folder, and queried the data for each file.



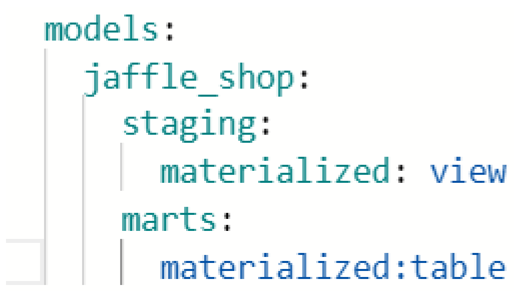
```
staging
  stg_customers.sql
  stg_orders.sql
```

- **Mart Models:**
- To be able to do this, you first have to create a new branch. When creating a new branch this allows you to be able to add, delete files and folders under the directory that you have. From there I created a folder into models, called marts, a folder called core into marts and lastly the file that they wanted.



```
models
  marts
    core
      dim_customers.sql
```

- **Configure your materializations:**



```
models:
  jaffle_shop:
    staging:
      materialized: view
    marts:
      materialized:table
```

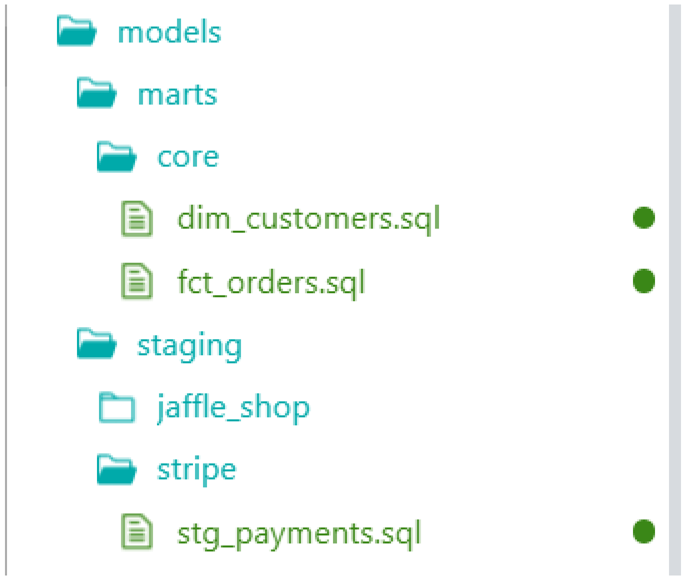
- ➢ Here, I went into the dbt_project.yml file and edited the lines and made sure of the indentation. That way the ones I want to be in view are in view and the ones to be in tables are materialized by table. Since marts is not in the staging folder but is under the models folder, there should be a difference in indentation in comparison of the staging and marts folder.
- ➢ Changing materialized from a table to a view is very important. You can see the changes within BigQuery whenever you refresh it as well!

❖ **Exemplar:**

➢ **In the files:**

- **Stg_payment.sql**
- **Fct_orders.sql**
- **Dim_customers.sql**

➢ I then recalled the lines of data to call the stripe data, and then was able to use the code. I then changed it from 'raw' to 'dbt-tutorial'. From there I was able to run the code, and it resulted into an error but the error then said that I needed to run a dbt run --full-refresh

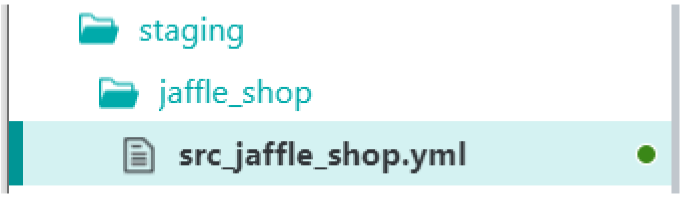


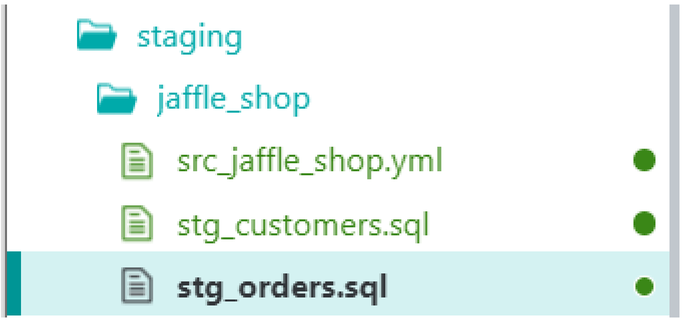
➢

b) COURSE 1 // Sources – Practice and Exemplar

❖ **Practice:**

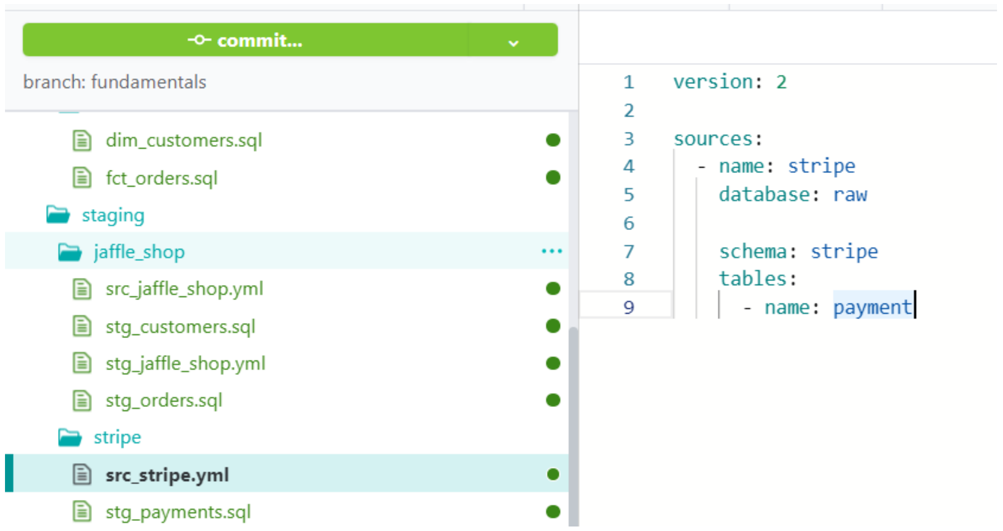
➢ **Configure Sources:**

- ➢ To configure the sources all I had to do was to make the changes within the yml file and that way it can then be applied to the rest of the jaffle_shop.
  - 

- ➢ **Refactoring staging models:**
- ➢ For both files, the sql files will query it directly from the source without needing to run it from the dbt we can source it from within the develop.
  - 

❖ **Exemplar: Self-check src_stripe and stg_payments**

- ➢ I then added the following codes into both of the files, and then the files were updated.



```
version: 2

sources:
  - name: stripe
    database: raw

    schema: stripe
    tables:
      - name: payment
```

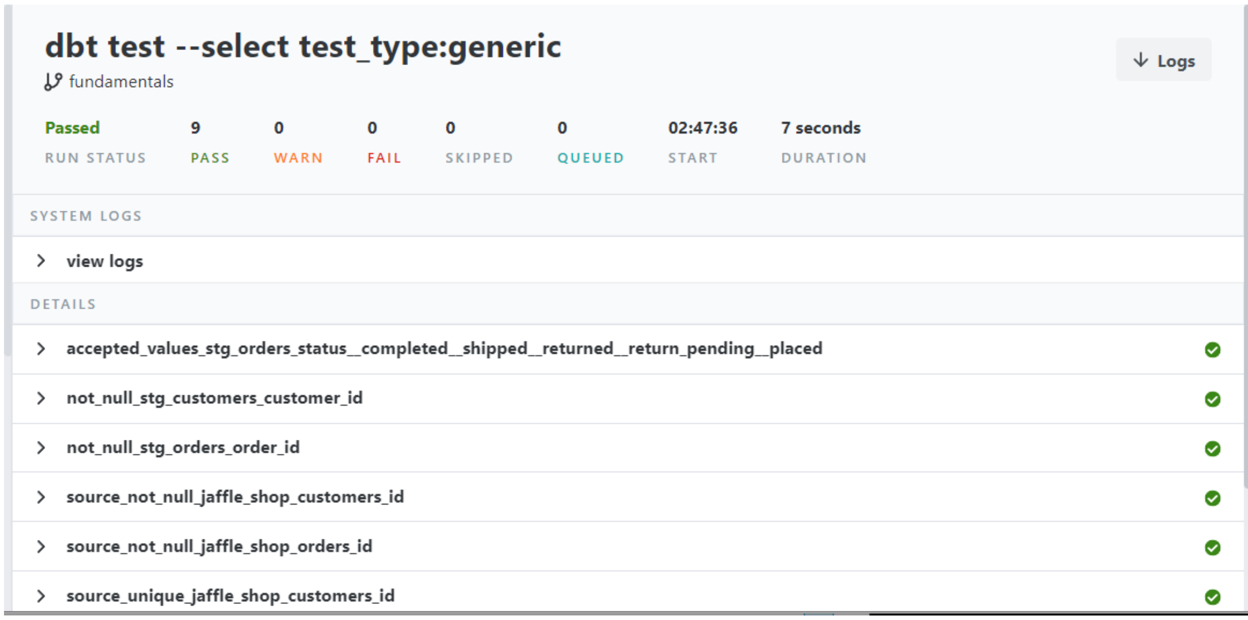
- ➢

c) COURSE 1 // **Tests – Practice and Exemplar**

❖ **Practice:**

- ➢ **Generic Tests:** This allows us to configure our tests. It will add unique and not null tests to the keys of each of the tables.
- ➢ **Singular Tests:** This adds it to the file of the stg_payments model, that is why it is a singular test.

- ➢ I had to first update the stg_jaffle_shop.yml file and then run the generic test.



dbt test --select test_type:generic

fundamentals

Logs

Passed	9	0	0	0	0	02:47:36	7 seconds
RUN STATUS	PASS	WARN	FAIL	SKIPPED	QUEUED	START	DURATION

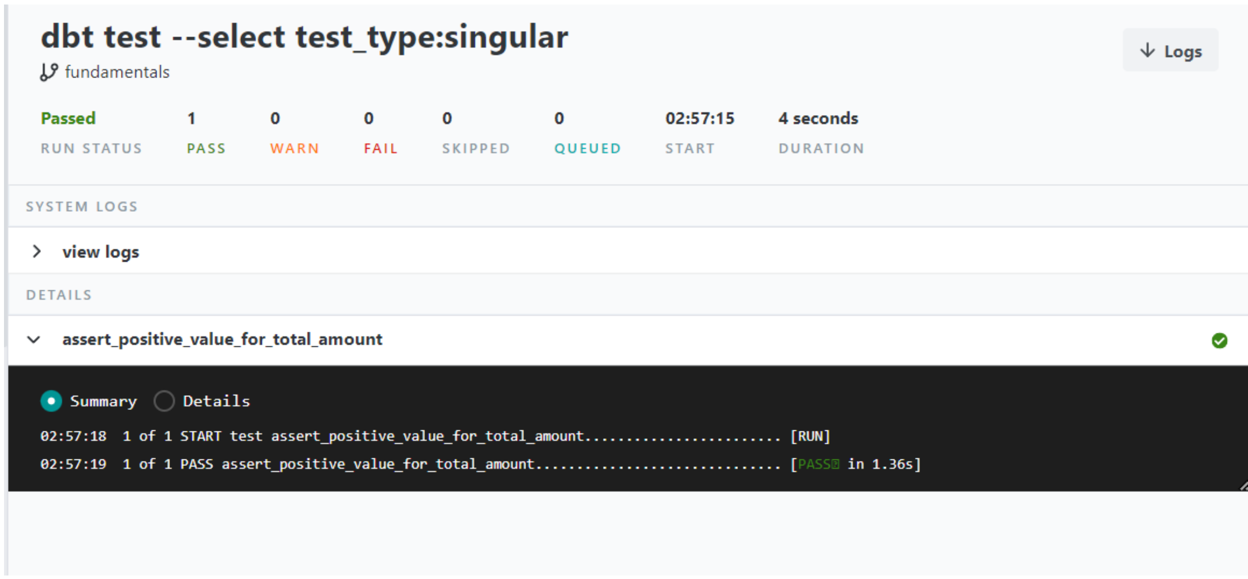
SYSTEM LOGS

view logs

DETAILS

- accepted_values_stg_orders_status__completed__shipped__returned__return_pending__placed ✓
- not_null_stg_customers_customer_id ✓
- not_null_stg_orders_order_id ✓
- source_not_null_jaffle_shop_customers_id ✓
- source_not_null_jaffle_shop_orders_id ✓
- source_unique_jaffle_shop_customers_id ✓

- ➢ For the singular test I had to configure the assert_positive…sql file to be able to run the singular test on stg_payments. As it runs the test for the total amounts.



dbt test --select test_type:singular

fundamentals

Logs

Passed	1	0	0	0	0	02:57:15	4 seconds
RUN STATUS	PASS	WARN	FAIL	SKIPPED	QUEUED	START	DURATION

SYSTEM LOGS

view logs

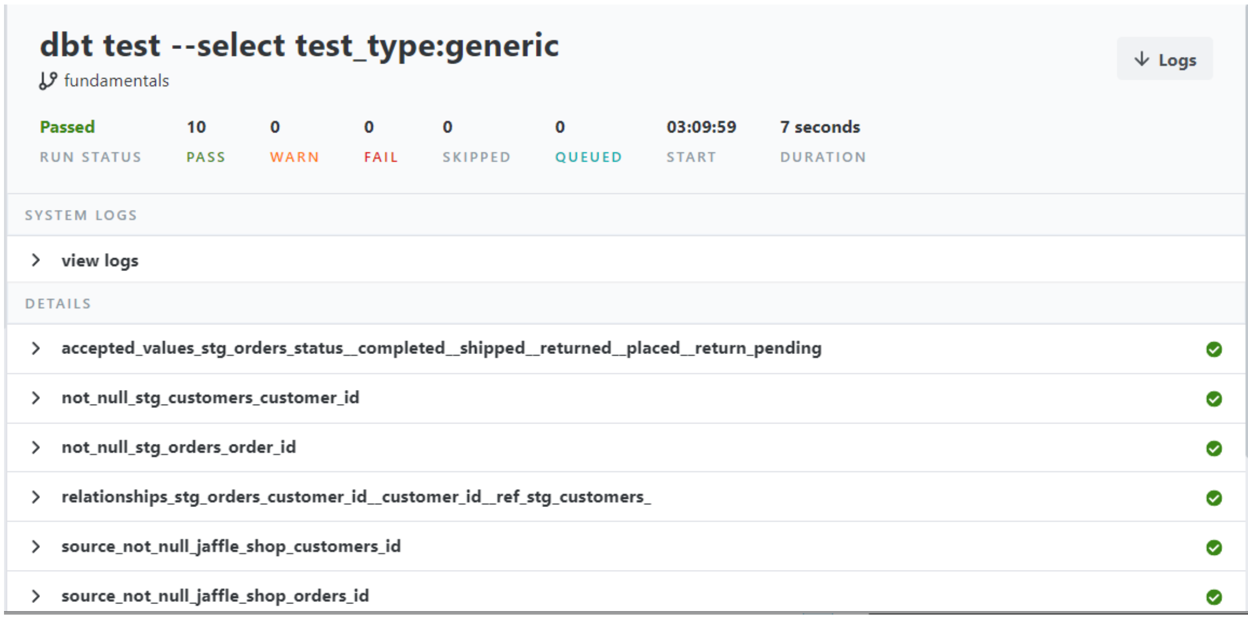
DETAILS

assert_positive_value_for_total_amount ✓

● Summary ○ Details

```
02:57:18  1 of 1 START test assert_positive_value_for_total_amount........................ [RUN]
02:57:19  1 of 1 PASS assert_positive_value_for_total_amount............................. [PASS in 1.36s]
```

❖ **Exemplar:**
Adding a relationship to the stg_orders model. That way the customer_id in stg_customers.



dbt test --select test_type:generic

fundamentals

Logs

```
Passed       10     0      0      0        0        03:09:59   7 seconds
RUN STATUS   PASS   WARN   FAIL   SKIPPED  QUEUED   START      DURATION
```

SYSTEM LOGS

view logs

DETAILS

```
accepted_values_stg_orders_status__completed__shipped__returned__placed__return_pending   ✓
not_null_stg_customers_customer_id   ✓
not_null_stg_orders_order_id   ✓
relationships_stg_orders_customer_id__customer_id__ref_stg_customers_   ✓
source_not_null_jaffle_shop_customers_id   ✓
source_not_null_jaffle_shop_orders_id   ✓
```

➢

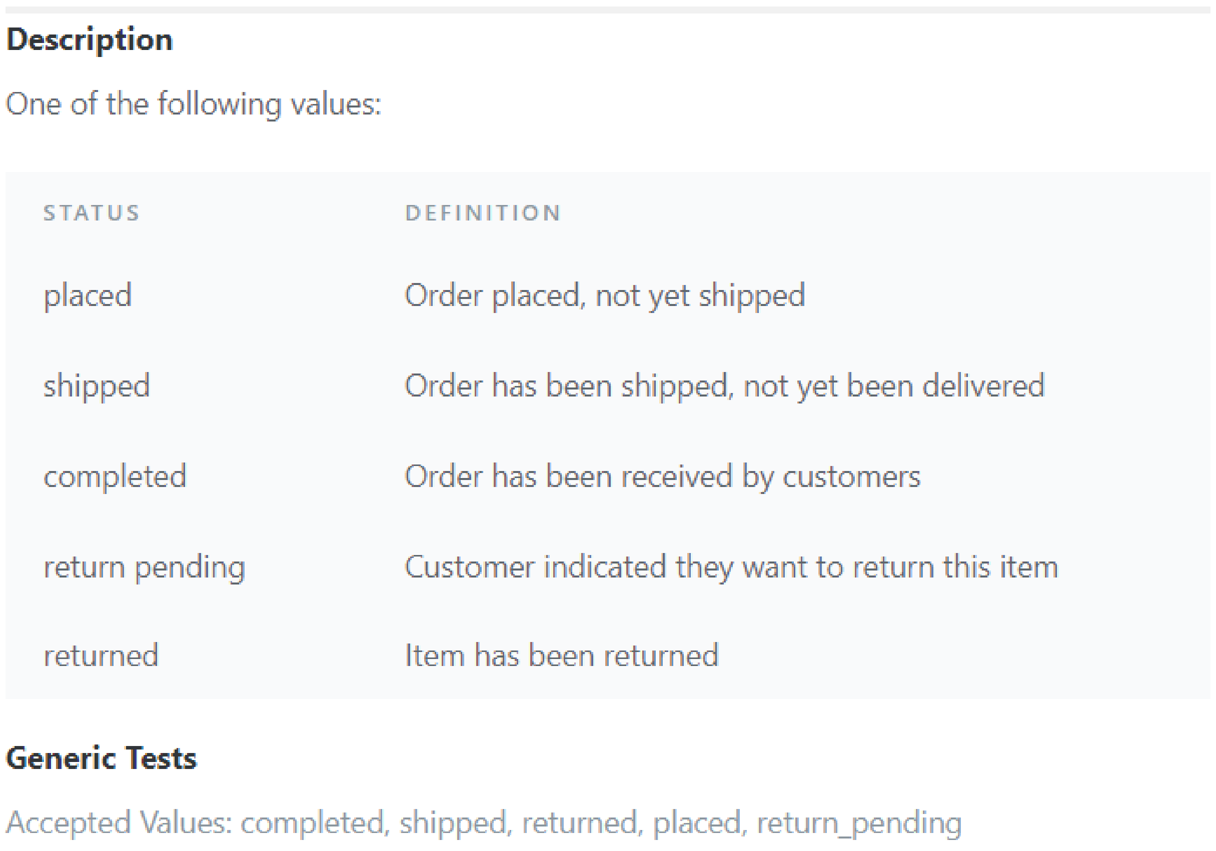
➢ **Instead of nine tests it will now test for 9 tests. Extra test is to check for the reference of the customer id.**

d) COURSE 1 // Documentation – Practice

❖ **Practice:**

❖ **Write Documentation & Create:** a reference to a doc block, I basically just copied the code onto the .yml file and the .md file. Then when creating the doc block for the orders model I wanted it to focus on the status column which is why the .md file explains each one of the entries for the status column.

❖ **Generate and view documentation:** After the files I then ran dbt docs generate. Updating the stg_orders and having the .md files which describes each of it.



Description

One of the following values:

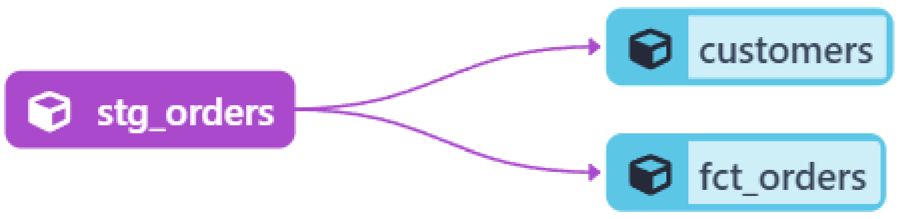
STATUS	DEFINITION
placed	Order placed, not yet shipped
shipped	Order has been shipped, not yet been delivered
completed	Order has been received by customers
return pending	Customer indicated they want to return this item
returned	Item has been returned

Generic Tests

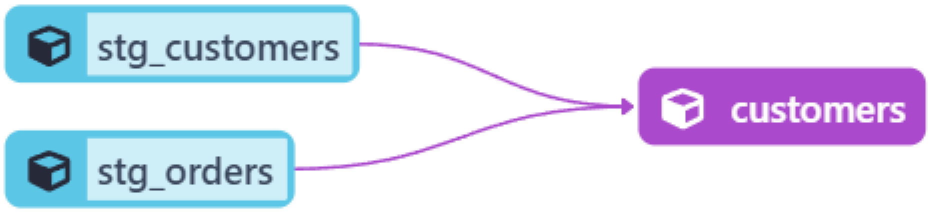
Accepted Values: completed, shipped, returned, placed, return_pending

❖

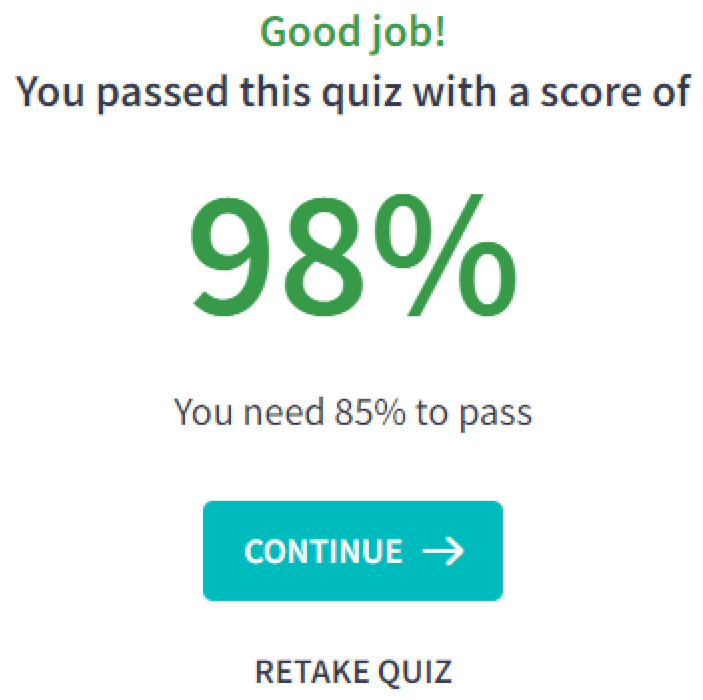
❖ **Lineage Graph: The lineage graph of the orders model.**



❖



❖

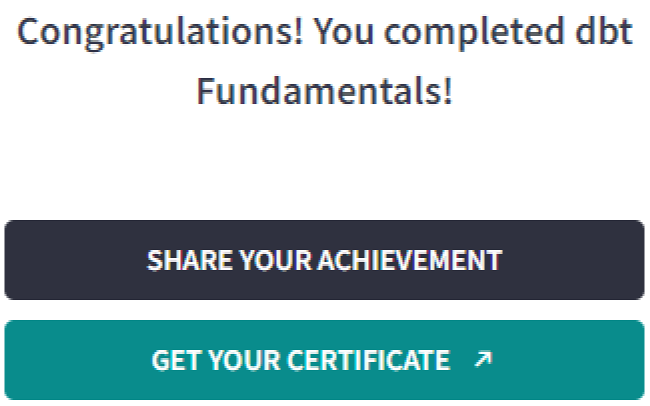


Good job!

You passed this quiz with a score of

98%

You need 85% to pass


Congratulations! You completed dbt Fundamentals!

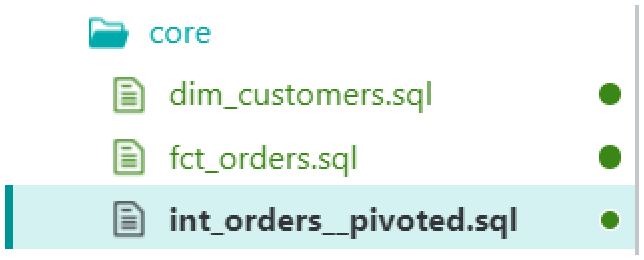
Second Course: Jinja, Macros and Packages Course

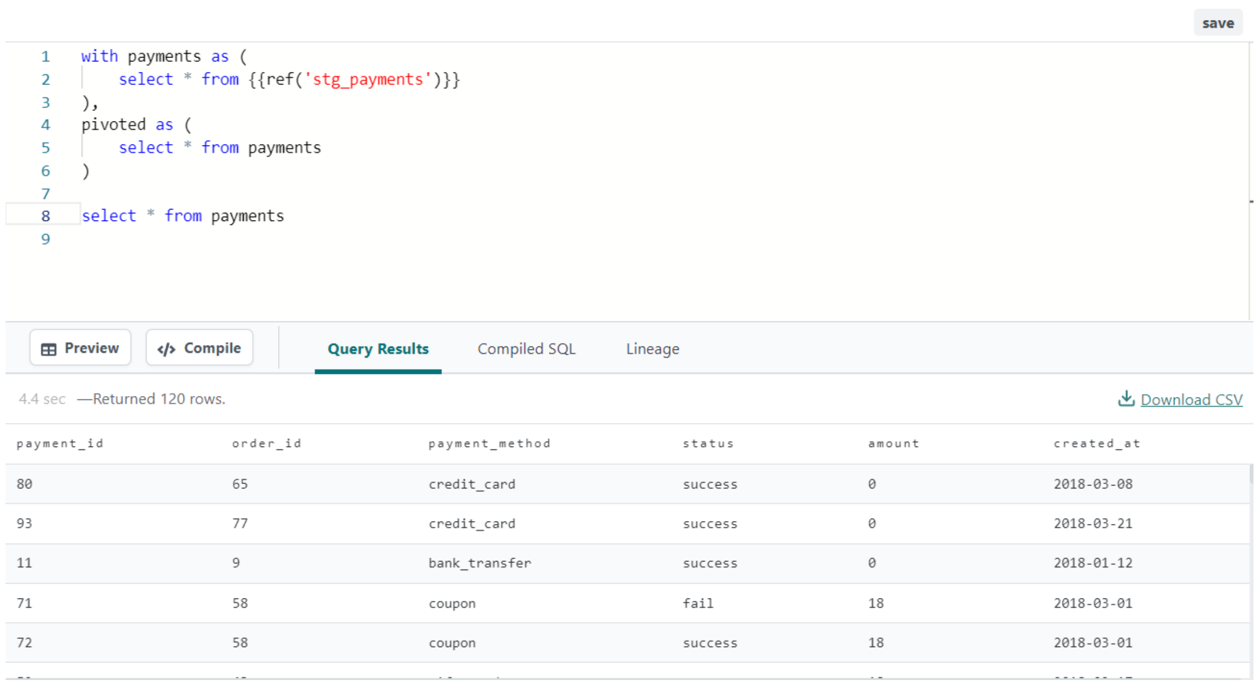
e) COURSE 2 //Jinja Primer

Jinja Primer - Practice and Exemplar (do not include the Advanced Jinja + Macros grant_select_macro part)

- ❖ **Practice:**
- ❖ A new file was created int_orders__pivoted.sql, where I will be able to follow along with the video to produce the following queries.
- ❖ After creating the file, I ran through the first query and got the results below.

➢





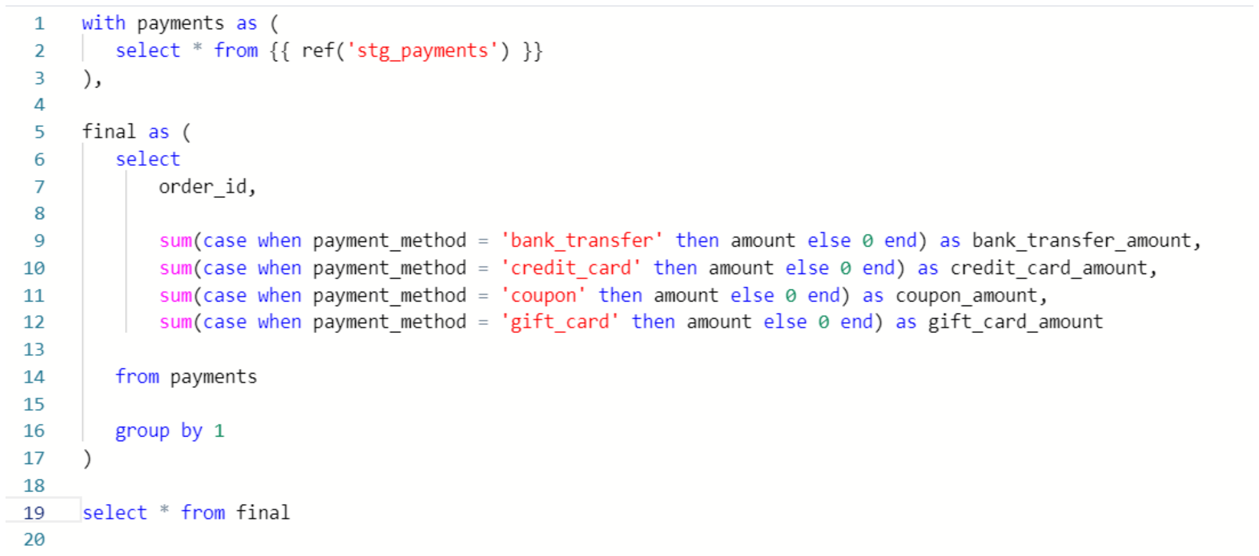
```sql
with payments as (
    select * from {{ref('stg_payments')}}
),
pivoted as (
    select * from payments
)

select * from payments
```

4.4 sec —Returned 120 rows.

payment_id	order_id	payment_method	status	amount	created_at
80	65	credit_card	success	0	2018-03-08
93	77	credit_card	success	0	2018-03-21
11	9	bank_transfer	success	0	2018-01-12
71	58	coupon	fail	18	2018-03-01
72	58	coupon	success	18	2018-03-01

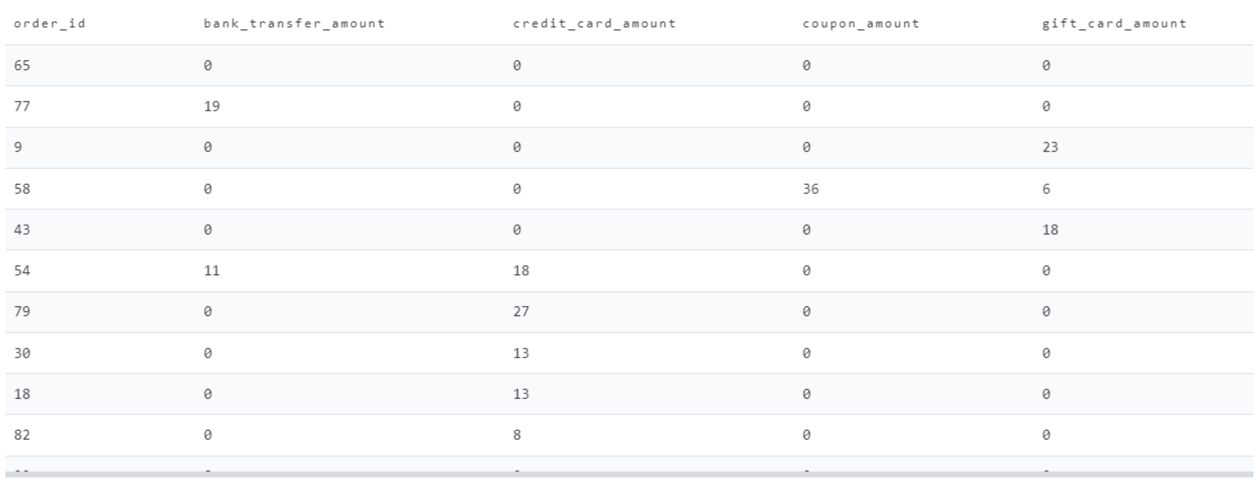
❖

❖ I then used the original SQL code: To just see and play around with the data. This allowed me to see that the SQL works and is able to be manipulated.



```sql
with payments as (
    select * from {{ ref('stg_payments') }}
),

final as (
    select
        order_id,

        sum(case when payment_method = 'bank_transfer' then amount else 0 end) as bank_transfer_amount,
        sum(case when payment_method = 'credit_card' then amount else 0 end) as credit_card_amount,
        sum(case when payment_method = 'coupon' then amount else 0 end) as coupon_amount,
        sum(case when payment_method = 'gift_card' then amount else 0 end) as gift_card_amount

    from payments

    group by 1
)

select * from final
```

❖



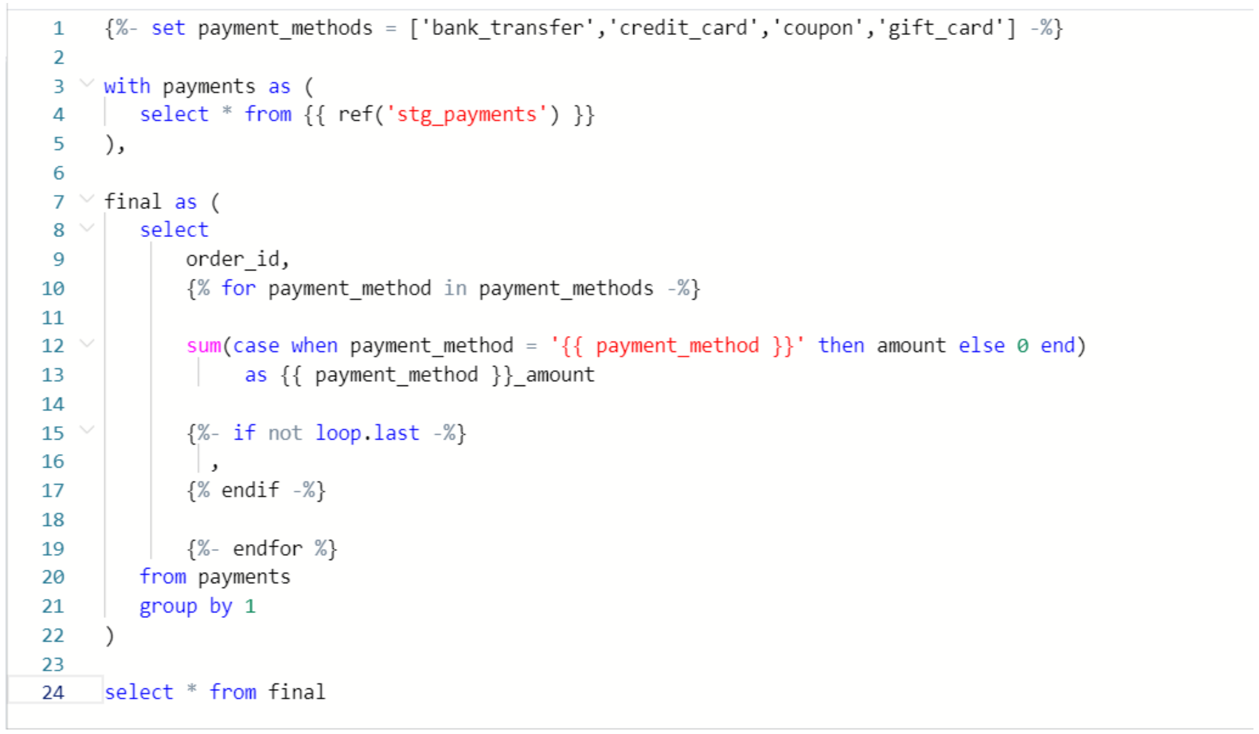
order_id	bank_transfer_amount	credit_card_amount	coupon_amount	gift_card_amount
65	0	0	0	0
77	19	0	0	0
9	0	0	0	23
58	0	0	36	6
43	0	0	0	18
54	11	18	0	0
79	0	27	0	0
30	0	13	0	0
18	0	13	0	0
82	0	8	0	0

❖

❖ **Exemplar:**

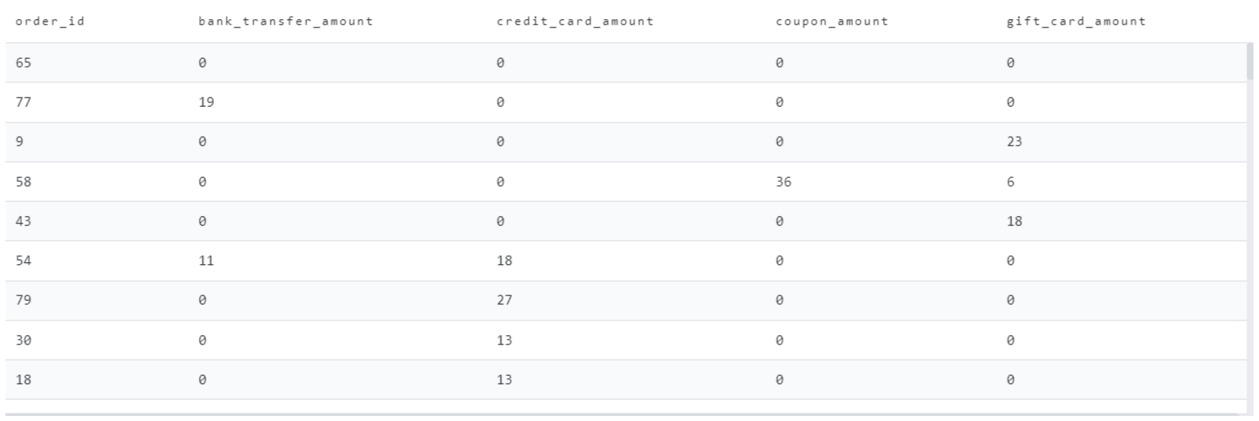
❖ The results then indicated the order id related to the amount and whether it was a bank transfer, credit card, coupon or a gift card.

❖ I then did the last of the SQL code, just that this is refactored jinja and sql combined. Where we can use the “python” language with SQL.



```
{%- set payment_methods = ['bank_transfer','credit_card','coupon','gift_card'] -%}

with payments as (
   select * from {{ ref('stg_payments') }}
),

final as (
   select
       order_id,
       {% for payment_method in payment_methods -%}

       sum(case when payment_method = '{{ payment_method }}' then amount else 0 end)
            as {{ payment_method }}_amount

       {%- if not loop.last -%}
         ,
       {% endif -%}

       {%- endfor %}
   from payments
   group by 1
)

select * from final
```

❖ **The results:**



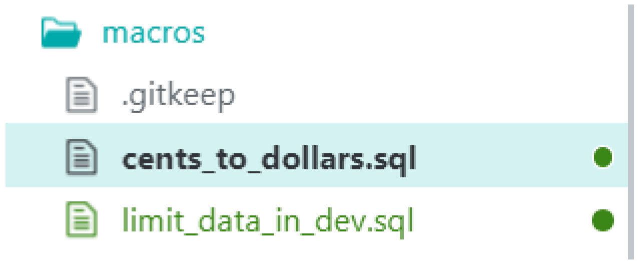
order_id	bank_transfer_amount	credit_card_amount	coupon_amount	gift_card_amount
65	0	0	0	0
77	19	0	0	0
9	0	0	0	23
58	0	0	36	6
43	0	0	0	18
54	11	18	0	0
79	0	27	0	0
30	0	13	0	0
18	0	13	0	0

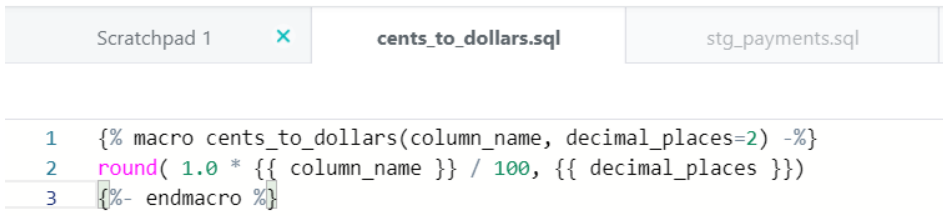
❖

f) COURSE 2 // Macros

Macros – Practice and Exemplar

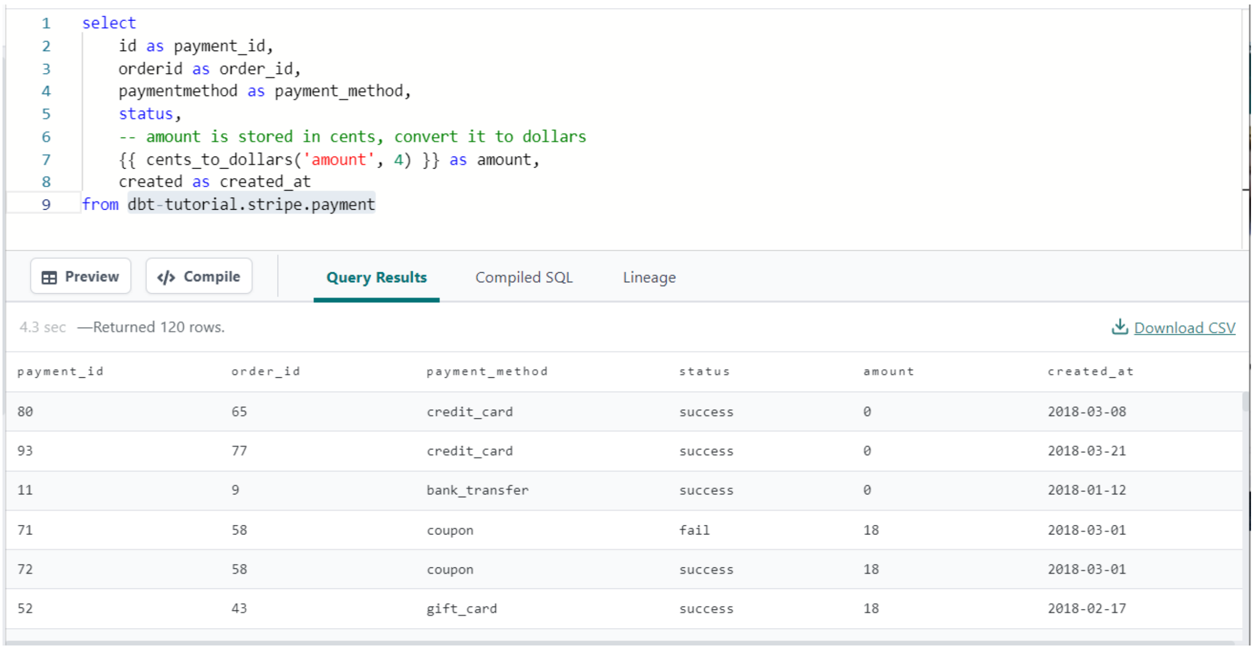
- ❖ **Practice:**
- ❖ **Cents_to_dollars.sql**
- ❖ Here we created a new sql file and then inserted the code for that file.





```
{% macro cents_to_dollars(column_name, decimal_places=2) -%}
round( 1.0 * {{ column_name }} / 100, {{ decimal_places }})
{%- endmacro %}
```

- ❖ **Exemplar:**
- The stg_payments sql has now been updated. The macro that we previously created cents_to_dollars.sql will then be against the stg_payments.sql.

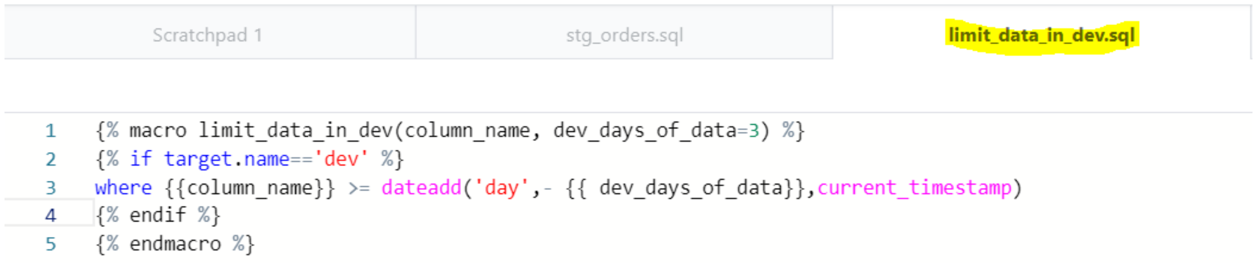


```
select
    id as payment_id,
    orderid as order_id,
    paymentmethod as payment_method,
    status,
    -- amount is stored in cents, convert it to dollars
    {{ cents_to_dollars('amount', 4) }} as amount,
    created as created_at
from dbt-tutorial.stripe.payment
```

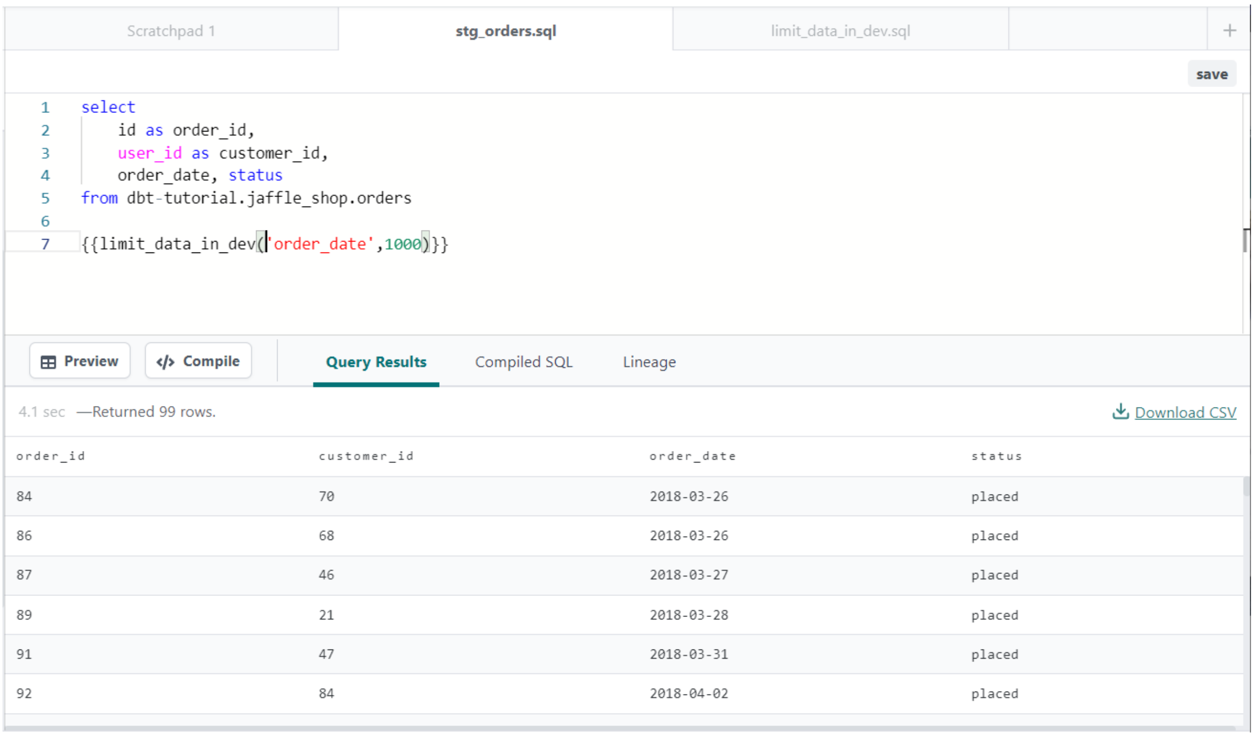
4.3 sec —Returned 120 rows.

payment_id	order_id	payment_method	status	amount	created_at
80	65	credit_card	success	0	2018-03-08
93	77	credit_card	success	0	2018-03-21
11	9	bank_transfer	success	0	2018-01-12
71	58	coupon	fail	18	2018-03-01
72	58	coupon	success	18	2018-03-01
52	43	gift_card	success	18	2018-02-17

- ❖ **Check out the limit data in dev macro video and implement this in your project.***
- ❖ sql file and The limit_data_in_dev.sql, has now been updated. This allows us to limit the macro file that we created before..



```
{% macro limit_data_in_dev(column_name, dev_days_of_data=3) %}
{% if target.name=='dev' %}
where {{column_name}} >= dateadd('day',- {{ dev_days_of_data}},current_timestamp)
{% endif %}
{% endmacro %}
```

- 

Scratchpad 1 | stg_orders.sql | limit_data_in_dev.sql

```sql
select
    id as order_id,
    user_id as customer_id,
    order_date, status
from dbt-tutorial.jaffle_shop.orders

{{limit_data_in_dev('order_date',1000)}}
```

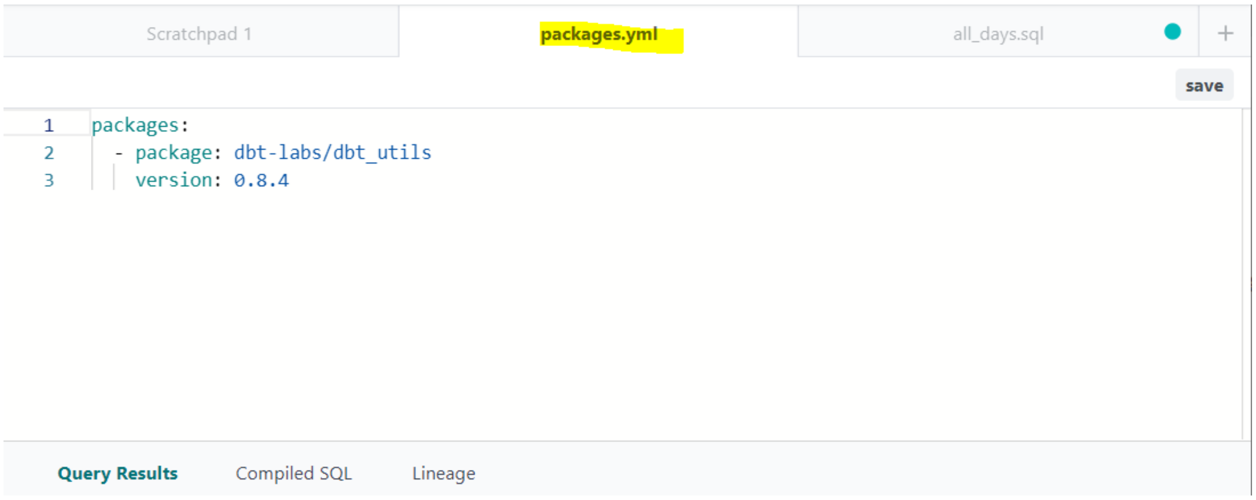
Preview | Compile | Query Results | Compiled SQL | Lineage

4.1 sec —Returned 99 rows. Download CSV

order_id	customer_id	order_date	status
84	70	2018-03-26	placed
86	68	2018-03-26	placed
87	46	2018-03-27	placed
89	21	2018-03-28	placed
91	47	2018-03-31	placed
92	84	2018-04-02	placed

g) COURSE 2 // Packages

Packages – Practice and Exemplar

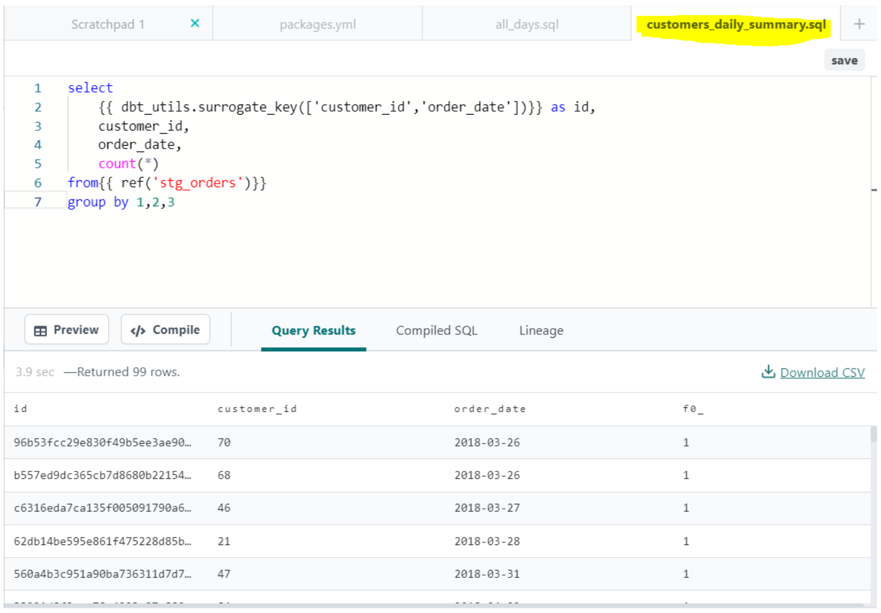
- ❖ **Practice:** Here all we needed to do is to create a new file under the models folder, and submit into it the new code that we wanted to use.
- ❖ I then downloaded the new version of the packages as I went over to the website.
- ❖ https://hub.getdbt.com/dbt-labs/dbt_utils/latest/
- ❖ From there I then submitted the new code to be able to run the first query.
- ❖ 

Scratchpad 1 | packages.yml | all_days.sql

```yaml
packages:
  - package: dbt-labs/dbt_utils
    version: 0.8.4
```

Query Results | Compiled SQL | Lineage

- ❖ The first query is just generating all of the dates from 2020-2021

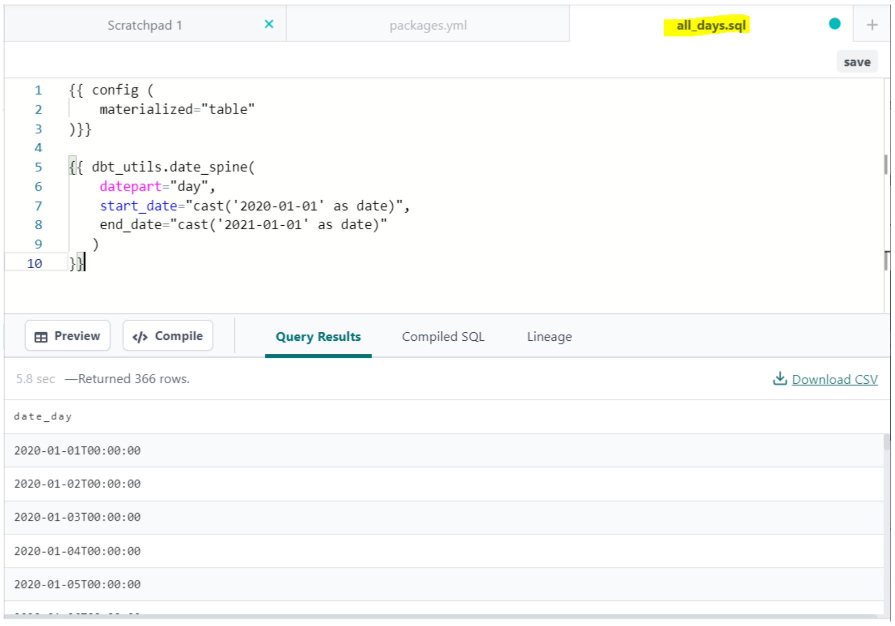


```sql
select
    {{ dbt_utils.surrogate_key(['customer_id','order_date'])}} as id,
    customer_id,
    order_date,
    count(*)
from{{ ref('stg_orders')}}
group by 1,2,3
```

3.9 sec —Returned 99 rows.

id	customer_id	order_date	f0_
96b53fcc29e830f49b5ee3ae90…	70	2018-03-26	1
b557ed9dc365cb7d8680b22154…	68	2018-03-26	1
c6316eda7ca135f005091790a6…	46	2018-03-27	1
62db14be595e861f475228d85b…	21	2018-03-28	1
560a4b3c951a90ba736311d7d7…	47	2018-03-31	1

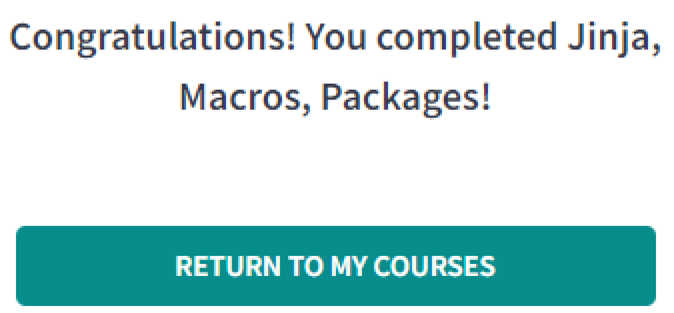
- Once adding the dbt_utils package to the project, we then are able to use the data spine macro to build a spine model for the all_dates file. By automatic the table will be in view, so we configure the block to make the table materialize.
- **Exemplar:**



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

{{ dbt_utils.date_spine(
    datepart="day",
    start_date="cast('2020-01-01' as date)",
    end_date="cast('2021-01-01' as date)"
   )
}}
```

5.8 sec —Returned 366 rows.

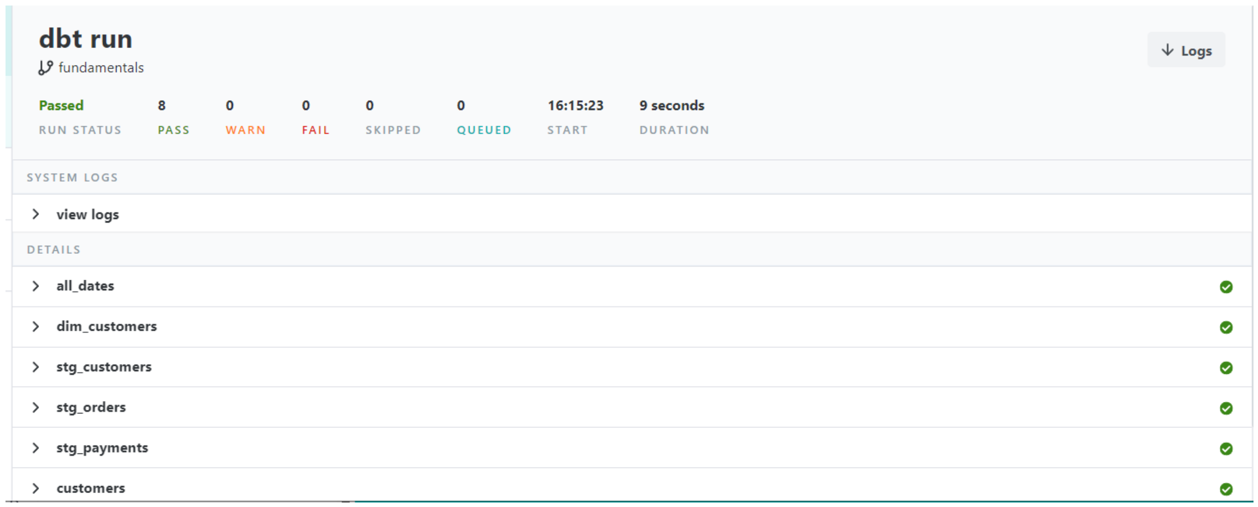
date_day
2020-01-01T00:00:00
2020-01-02T00:00:00
2020-01-03T00:00:00
2020-01-04T00:00:00
2020-01-05T00:00:00


Congratulations! You completed Jinja, Macros, Packages!

Third Course: Advanced Materialization Course

h) COURSE 3 // Materializations
Materializations – Practice (Skip the section on "Incremental Models")

❖ **Practice:** dbt run to the updated code and configuring the views of the tables.


```
dbt run
fundamentals

Passed       8      0      0      0         0        16:15:23   9 seconds
RUN STATUS   PASS   WARN   FAIL   SKIPPED   QUEUED   START      DURATION

SYSTEM LOGS
> view logs
DETAILS
> all_dates
> dim_customers
> stg_customers
> stg_orders
> stg_payments
> customers
```

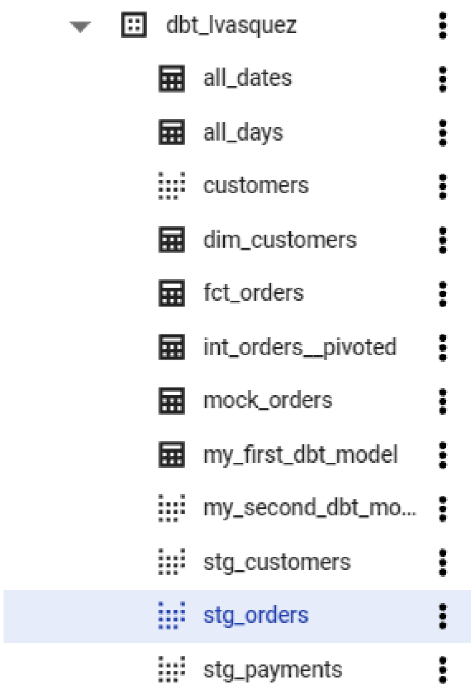
❖

❖ After running dbt run, I was not able to further advance into the practice as it required snowplow which is not available on Google BigQuery.

❖ **Snapshots:**

Snapshots are difficult to practice without genuine type 2, slowly changing dimension data. For this exercise, use the following code snippets to practice snapshots. You may need to adjust the Snowflake snippets based on your data warehouse.

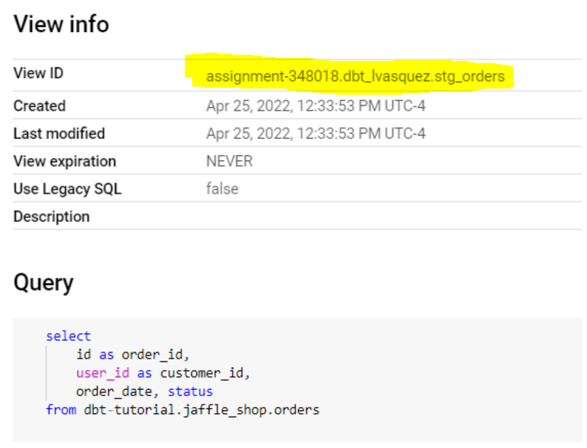
- (In Snowflake) Create a table called mock_orders in your development schema. You will have to replace dbt_kcoapman in the snippet below.



Instead use these codes in **Google BigQuery:**

```
create or replace table google-cloud-project.bigquery-dataset.mock_orders (
   order_id integer,
   status varchar (100),
   created_at date,
   updated_at date
);


insert into google-cloud-project.bigquery-dataset.mock_orders (order_id, status,
created_at, updated_at)
values (1, 'delivered', '2020-01-01', '2020-01-04'),
       (2, 'shipped', '2020-01-02', '2020-01-04'),
       (3, 'shipped', '2020-01-03', '2020-01-04'),
       (4, 'processed', '2020-01-04', '2020-01-04');
```

Figuring this code out was a challenge! First, I wasn't sure as to what was my project name or the dataset,



View info

View ID	assignment-348018.dbt_lvasquez.stg_orders
Created	Apr 25, 2022, 12:33:53 PM UTC-4
Last modified	Apr 25, 2022, 12:33:53 PM UTC-4
View expiration	NEVER
Use Legacy SQL	false
Description	

Query

```
select
    id as order_id,
    user_id as customer_id,
    order_date, status
from dbt-tutorial.jaffle_shop.orders
```

Google-cloud-project.bigquery-dataset.mock_orders

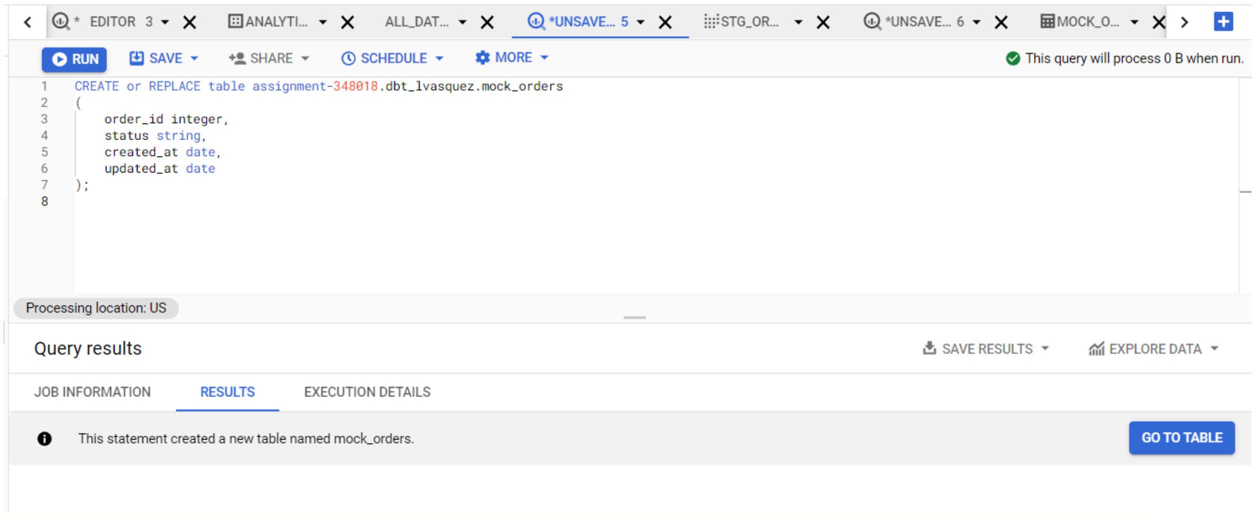
assignment-348018.dbt_lvasquez.mock_orders

```
CREATE or REPLACE table assignment-348018.dbt_lvasquez.mock_orders
(
    order_id integer,
    status string,
    created_at date,
    updated_at date
);
```

For the status I had to change the data type from a VARCHAR into a string. For some unknown reason, VARCHAR was not working within BigQuery.
From repeatedly trying to work with the Varchar: I quickly did a quick google search and found this information below:

> VARCHAR datatype in Bigquery
>
> **STRING is the equivalent for VARCHAR datatype in Bigquery**. While migrating the code from Oracle you will need to rewrite this datatype in Bigquery.

Afterwards, this actually ran through and produced and created the table:



```
CREATE or REPLACE table assignment-348018.dbt_lvasquez.mock_orders
(
    order_id integer,
    status string,
    created_at date,
    updated_at date
);
```

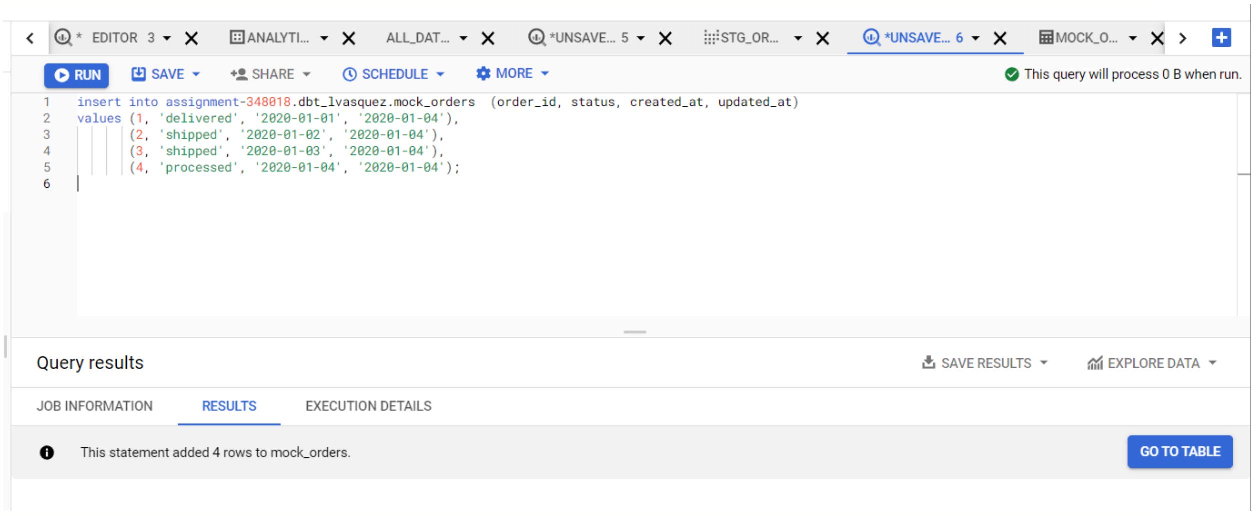
Processing location: US

This statement created a new table named mock_orders.

- (In Snowflake) Insert values into the mock_orders table in your development schema. You will have to replace dbt_kcoapman in the snippet below.

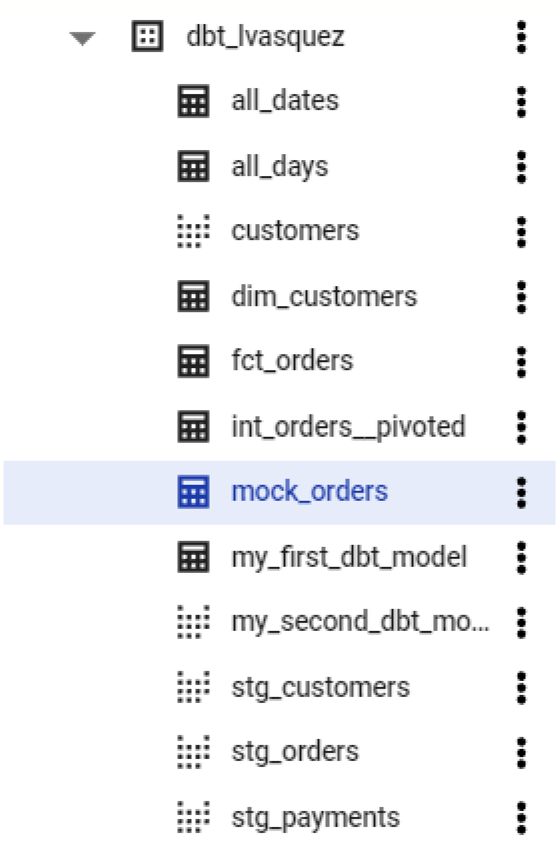
```
insert into assignment-348018.dbt_lvasquez.mock_orders  (order_id, status,
created_at, updated_at)
values (1, 'delivered', '2020-01-01', '2020-01-04'),
       (2, 'shipped', '2020-01-02', '2020-01-04'),
       (3, 'shipped', '2020-01-03', '2020-01-04'),
       (4, 'processed', '2020-01-04', '2020-01-04');
```

For this code, all I had to mainly change was the first line. Minor changes and it worked seamlessly!

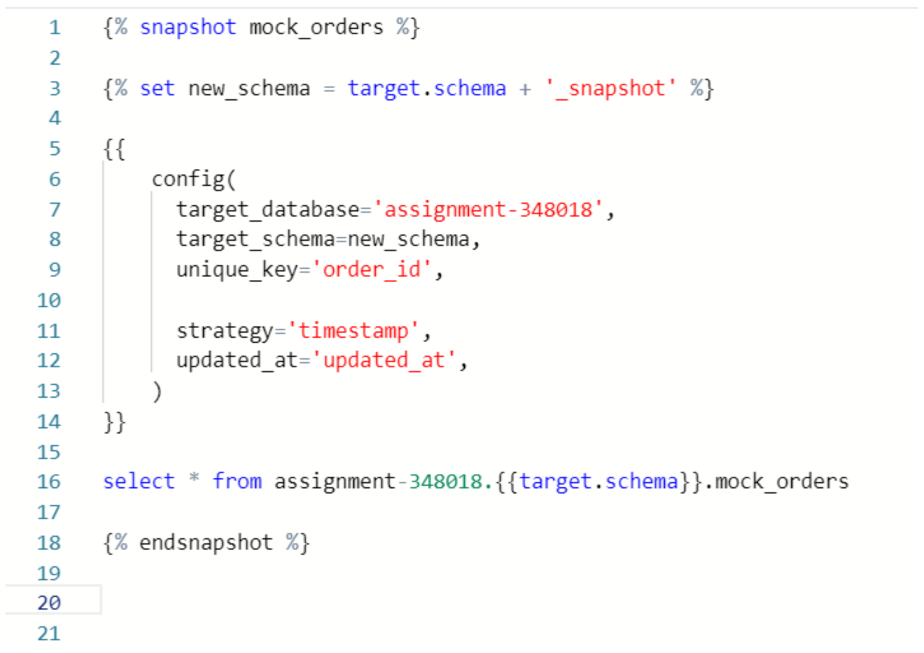


```
insert into assignment-348018.dbt_lvasquez.mock_orders  (order_id, status, created_at, updated_at)
values (1, 'delivered', '2020-01-01', '2020-01-04'),
       (2, 'shipped', '2020-01-02', '2020-01-04'),
       (3, 'shipped', '2020-01-03', '2020-01-04'),
       (4, 'processed', '2020-01-04', '2020-01-04');
```

This statement added 4 rows to mock_orders.



- (In dbt Cloud) Create a new snapshot in the folder snapshots with the filename mock_orders.sql with the following code snippet. Note: Jinja is being used here to create a new, dedicated schema.

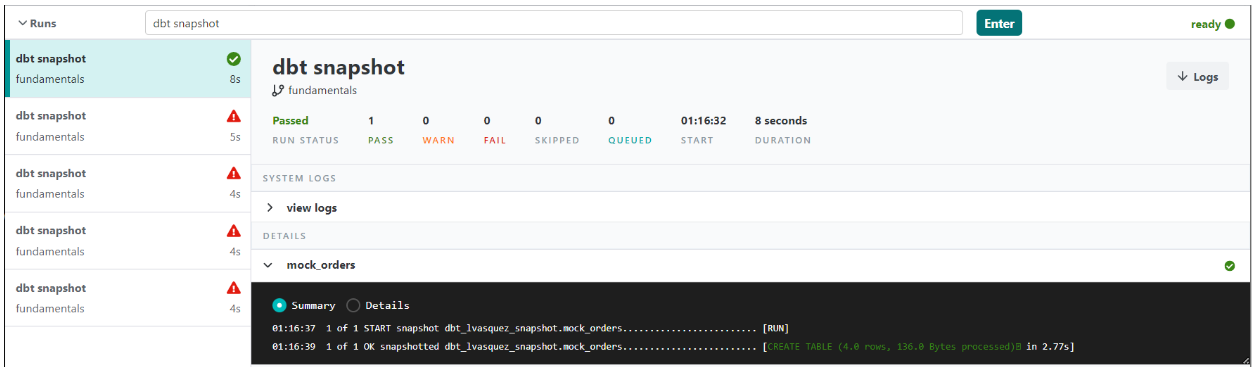


```
{% snapshot mock_orders %}

{% set new_schema = target.schema + '_snapshot' %}

{{
    config(
      target_database='assignment-348018',
      target_schema=new_schema,
      unique_key='order_id',

      strategy='timestamp',
      updated_at='updated_at',
    )
}}

select * from assignment-348018.{{target.schema}}.mock_orders

{% endsnapshot %}
```

-

After creating the mock_orders.sql file, I then needed to insert the code. Once I inserted the code and adjusted the code to configure my dataset and or project, everything worked.

I then ran a dbt snapshot.



Runs | dbt snapshot | Enter | ready

dbt snapshot / fundamentals	✓	8s
dbt snapshot / fundamentals	⚠	5s
dbt snapshot / fundamentals	⚠	4s
dbt snapshot / fundamentals	⚠	4s
dbt snapshot / fundamentals	⚠	4s

dbt snapshot
fundamentals

RUN STATUS	PASS	WARN	FAIL	SKIPPED	QUEUED	START	DURATION
Passed	1	0	0	0	0	01:16:32	8 seconds

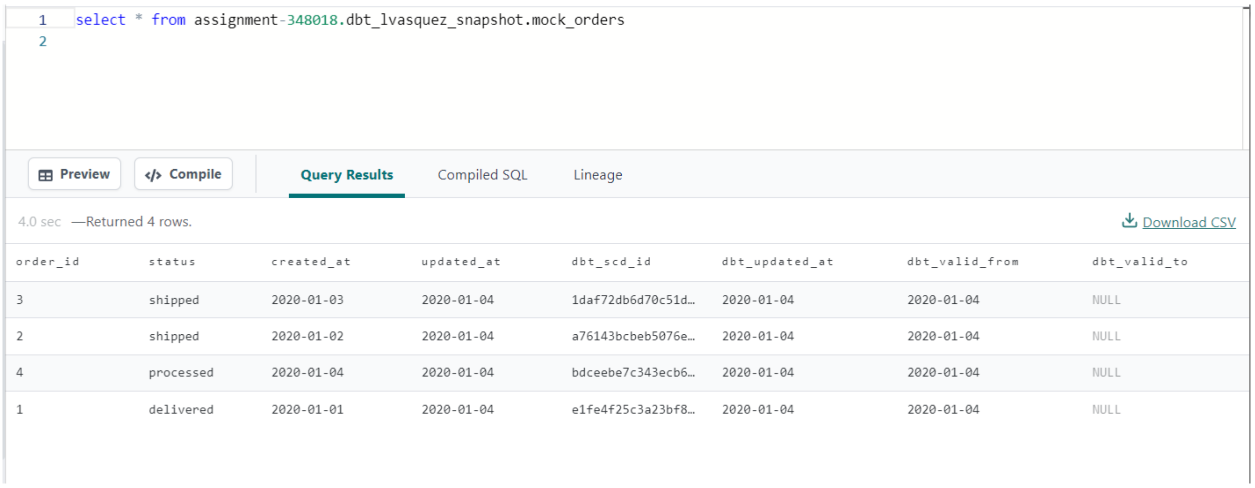
SYSTEM LOGS
view logs
DETAILS
mock_orders

```
01:16:37  1 of 1 START snapshot dbt_lvasquez_snapshot.mock_orders........................ [RUN]
01:16:39  1 of 1 OK snapshotted dbt_lvasquez_snapshot.mock_orders........................ [CREATE TABLE (4.0 rows, 136.0 Bytes processed) in 2.77s]
```

After several tries and errors, I finally got it to work!

- (In dbt Cloud) Run snapshots by executing dbt snapshot.

- (In dbt Cloud) Run the following snippet in a statement tab to see the current snapshot table. You will have to replace dbt_kcoapman with your development schema. Take note of how dbt has added three columns.

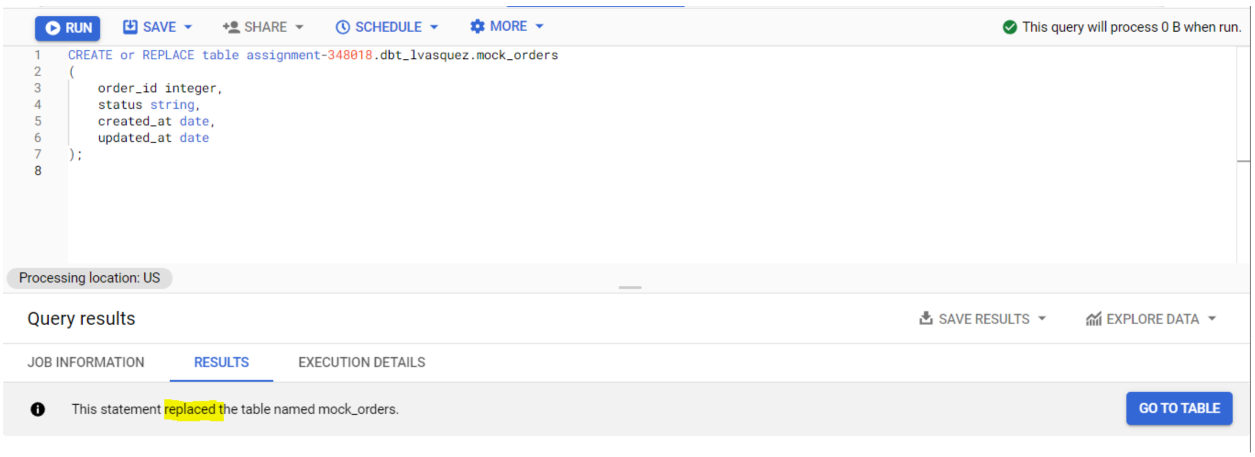


```
select * from assignment-348018.dbt_lvasquez_snapshot.mock_orders
```

4.0 sec —Returned 4 rows.

order_id	status	created_at	updated_at	dbt_scd_id	dbt_updated_at	dbt_valid_from	dbt_valid_to
3	shipped	2020-01-03	2020-01-04	1daf72db6d70c51d…	2020-01-04	2020-01-04	NULL
2	shipped	2020-01-02	2020-01-04	a76143bcbeb5076e…	2020-01-04	2020-01-04	NULL
4	processed	2020-01-04	2020-01-04	bdceebe7c343ecb6…	2020-01-04	2020-01-04	NULL
1	delivered	2020-01-01	2020-01-04	e1fe4f25c3a23bf8…	2020-01-04	2020-01-04	NULL

- (In Snowflake) Create a table called mock_orders in your development schema. You will have to replace dbt_kcoapman in the snippet below.



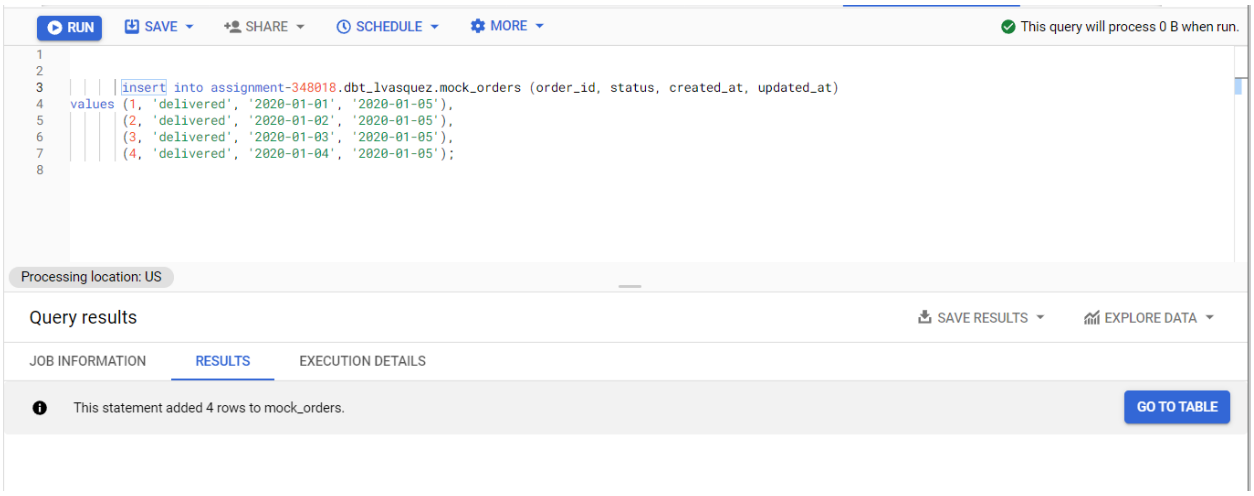
This query will process 0 B when run.

```
CREATE or REPLACE table assignment-348018.dbt_lvasquez.mock_orders
(
    order_id integer,
    status string,
    created_at date,
    updated_at date
);
```

Processing location: US

This statement replaced the table named mock_orders.

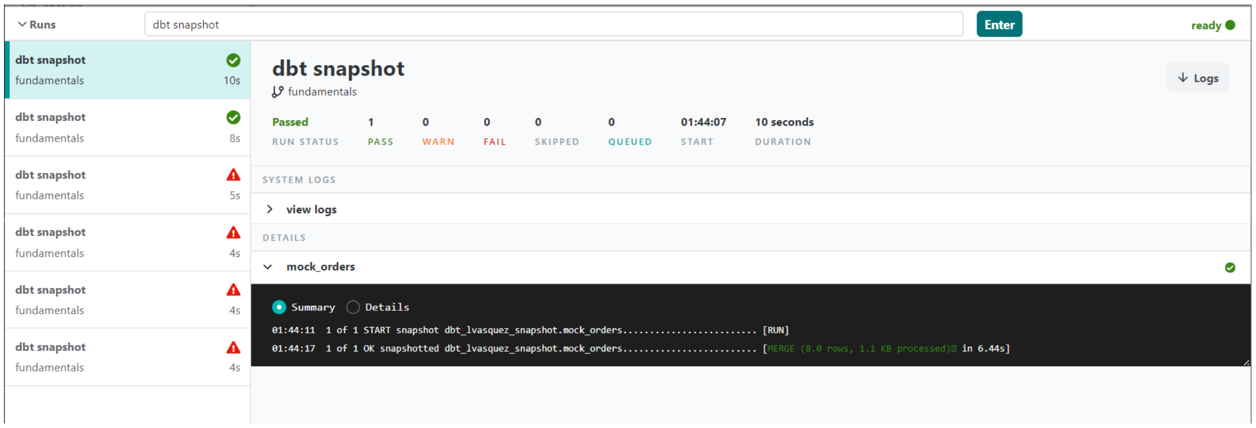
- (In Snowflake) Insert values into the mock_orders table in your development schema. You will have to replace dbt_kcoapman in the snippet below.



```
insert into assignment-348018.dbt_lvasquez.mock_orders (order_id, status, created_at, updated_at)
values (1, 'delivered', '2020-01-01', '2020-01-05'),
       (2, 'delivered', '2020-01-02', '2020-01-05'),
       (3, 'delivered', '2020-01-03', '2020-01-05'),
       (4, 'delivered', '2020-01-04', '2020-01-05');
```

This statement added 4 rows to mock_orders.

- (In dbt Cloud) Re-run snapshots by executing dbt snapshot.



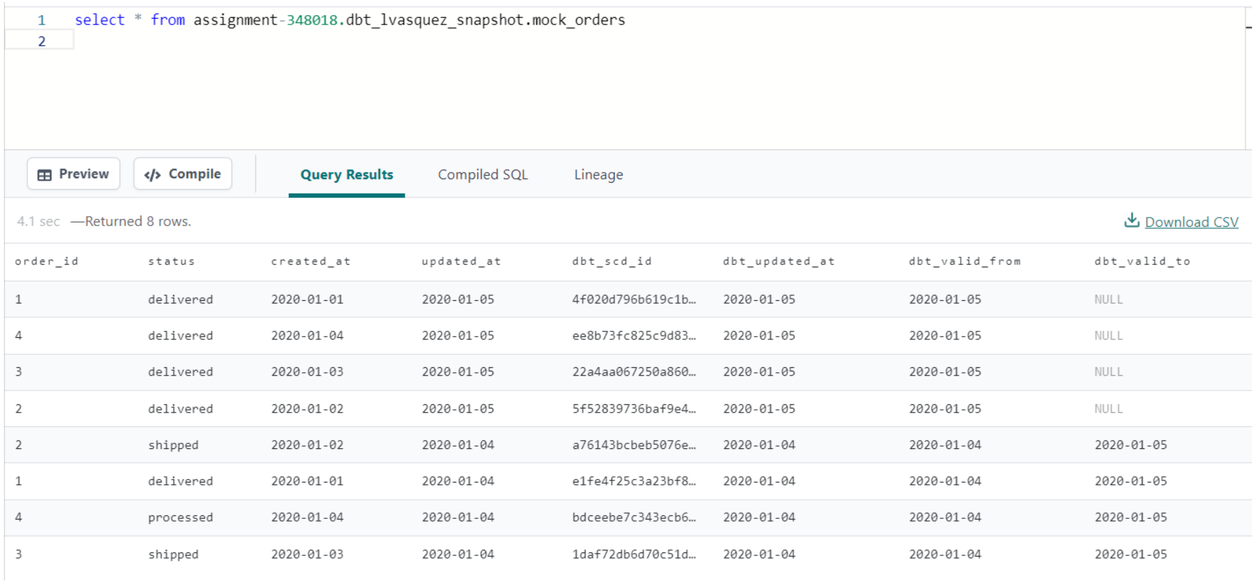
Run	Command	Duration
dbt snapshot	fundamentals	10s
dbt snapshot	fundamentals	8s
dbt snapshot	fundamentals	5s
dbt snapshot	fundamentals	4s
dbt snapshot	fundamentals	4s
dbt snapshot	fundamentals	4s

dbt snapshot
fundamentals

RUN STATUS	PASS	WARN	FAIL	SKIPPED	QUEUED	START	DURATION
Passed	1	0	0	0	0	01:44:07	10 seconds

```
01:44:11  1 of 1 START snapshot dbt_lvasquez_snapshot.mock_orders......................... [RUN]
01:44:17  1 of 1 OK snapshotted dbt_lvasquez_snapshot.mock_orders......................... [MERGE (8.0 rows, 1.1 KB processed) in 6.44s]
```

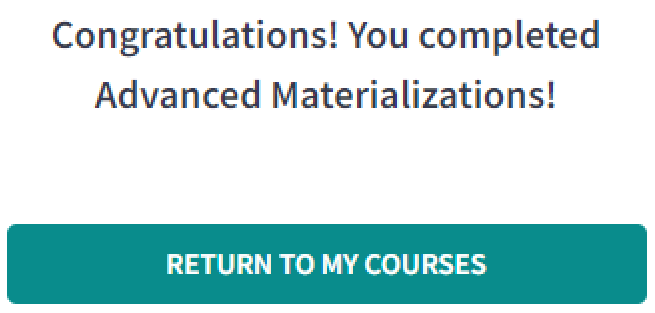
- (In dbt Cloud) Run the following snippet in a statement tab to see the current snapshot table. You will have to replace dbt_kcoapman with your development schema. Now take note of how dbt has 'snapshotted' the data to capture the changes over time!



```
select * from assignment-348018.dbt_lvasquez_snapshot.mock_orders
```

4.1 sec —Returned 8 rows.

order_id	status	created_at	updated_at	dbt_scd_id	dbt_updated_at	dbt_valid_from	dbt_valid_to
1	delivered	2020-01-01	2020-01-05	4f020d796b619c1b…	2020-01-05	2020-01-05	NULL
4	delivered	2020-01-04	2020-01-05	ee8b73fc825c9d83…	2020-01-05	2020-01-05	NULL
3	delivered	2020-01-03	2020-01-05	22a4aa067250a860…	2020-01-05	2020-01-05	NULL
2	delivered	2020-01-02	2020-01-05	5f52839736baf9e4…	2020-01-05	2020-01-05	NULL
2	shipped	2020-01-02	2020-01-04	a76143bcbeb5076e…	2020-01-04	2020-01-04	2020-01-05
1	delivered	2020-01-01	2020-01-04	e1fe4f25c3a23bf8…	2020-01-04	2020-01-04	2020-01-05
4	processed	2020-01-04	2020-01-04	bdceebe7c343ecb6…	2020-01-04	2020-01-04	2020-01-05
3	shipped	2020-01-03	2020-01-04	1daf72db6d70c51d…	2020-01-04	2020-01-04	2020-01-05



Congratulations! You completed Advanced Materializations!

Fourth Course: Analysis and Seeds Course

Practice

Analyses

Using you new knowledge of analyses, create an analysis file in the analyses folder called total_revenue.sql that uses the stg_payments model and sums the amount of successful payments. (Remember to use Jinja in this rather than the raw table name)

Seeds

Using your new knowledge of seeds, create a seed file in the seeds folder (or if you're using a dbt version prior to 1.0.0, it will be called the data folder) called employees.csv with the following values:

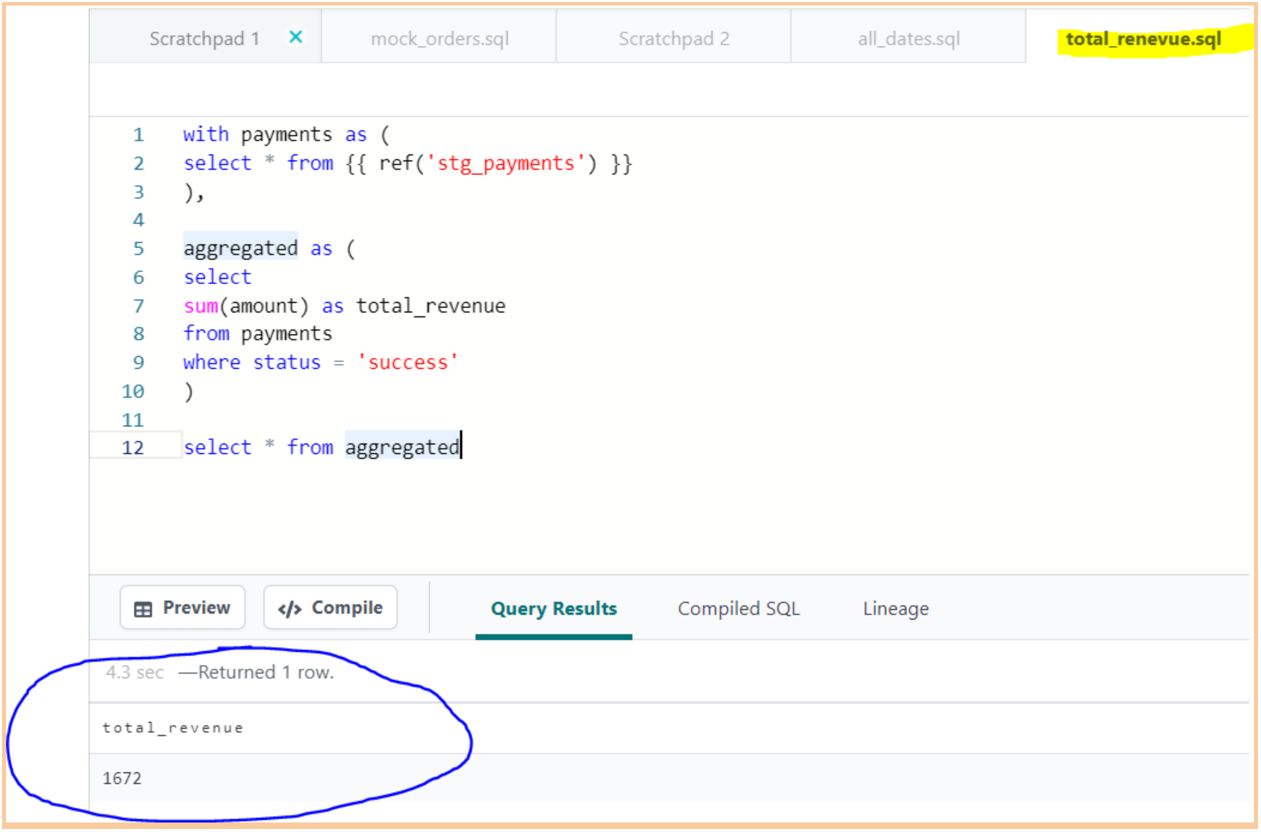
```
employee_id,email,customer_id
3425, mike@jaffleshop.com, 1
2354, sarah@jaffleshop.com, 6
2342, frank@jaffleshop.com, 8
1234, jennifer@jaffleshop.com, 9
```

Build this seed into your data warehouse by running dbt seed.

i) Analyses and Seeds –

Practice

For the practice portion, it was pretty simple.All I had to do is create the total_revenue.sql file and then add the code. At the end, I previewed the result and got one result of the total revenue from the stg_payments model.

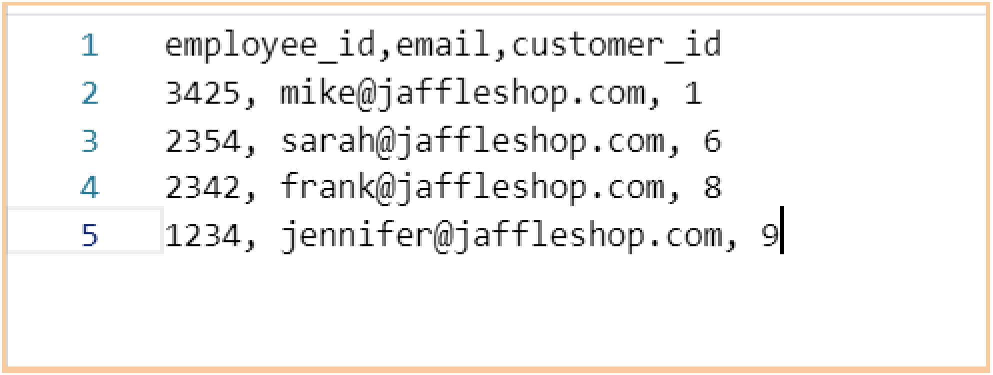


```
with payments as (
select * from {{ ref('stg_payments') }}
),

aggregated as (
select
sum(amount) as total_revenue
from payments
where status = 'success'
)

select * from aggregated
```

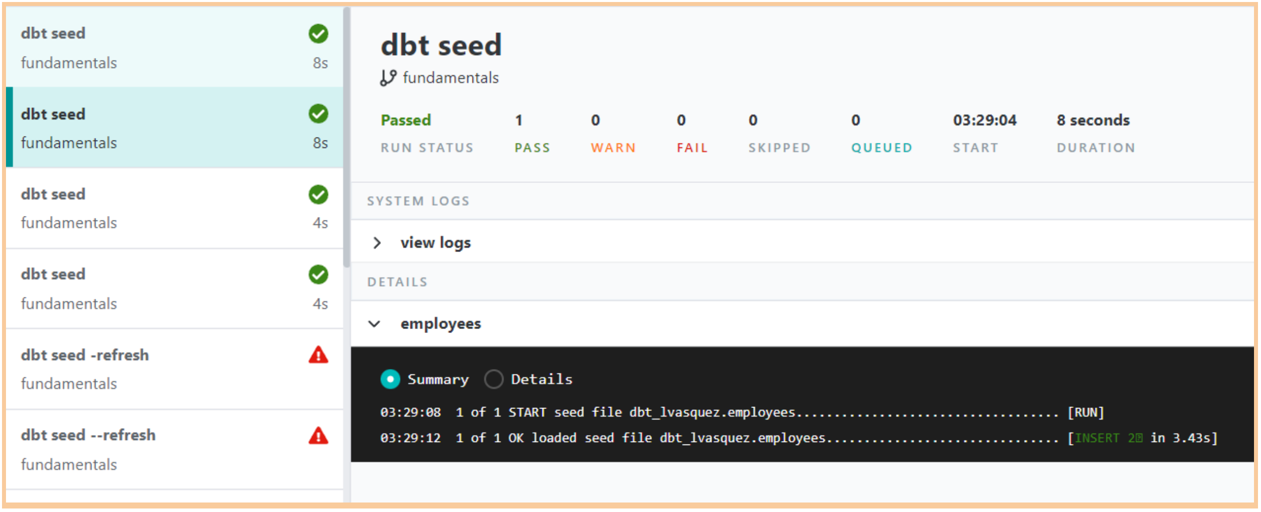
```
4.3 sec —Returned 1 row.
total_revenue
1672
```

Exemplar

For this one, I needed to play around with the employees file. For some reason, dbt seed would not work. I then read into the documentation that if dbt is above 1.0 then there might be a mishap in the naming of the folder, so I renamed the folder ‘data’ as a later version would of, ran that, and then renamed it back to the seed folder, and then conducted dbt seed.
Once renaming the folders took place, dbt seed then finally worked.



```
employee_id,email,customer_id
3425, mike@jaffleshop.com, 1
2354, sarah@jaffleshop.com, 6
2342, frank@jaffleshop.com, 8
1234, jennifer@jaffleshop.com, 9
```



dbt seed fundamentals	✓ 8s
dbt seed fundamentals	✓ 8s
dbt seed fundamentals	✓ 4s
dbt seed fundamentals	✓ 4s
dbt seed -refresh fundamentals	⚠
dbt seed --refresh fundamentals	⚠

dbt seed
fundamentals

RUN STATUS	PASS	WARN	FAIL	SKIPPED	QUEUED	START	DURATION
Passed	1	0	0	0	0	03:29:04	8 seconds

SYSTEM LOGS

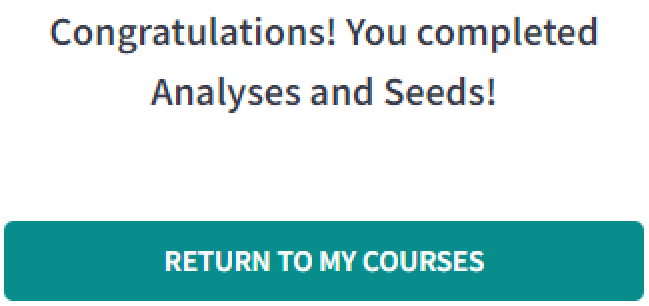
> view logs

DETAILS

employees

● Summary ○ Details

```
03:29:08  1 of 1 START seed file dbt_lvasquez.employees................................. [RUN]
03:29:12  1 of 1 OK loaded seed file dbt_lvasquez.employees............................. [INSERT 2 in 3.43s]
```



Congratulations! You completed Analyses and Seeds!

Write a 1 paragraph conclusion including:
a) How long it took you to complete the assignment (in units of hours actually working).
b) What was the most difficult part of the assignment?

Overall, I did this starting from Friday. In the span of the Friday to Monday, I would say that overall it took me about 15-25 hours. There is a gap, as most of the time I had a bug and needed to find the bug to be able to approach the project.
The most difficult part would be course 1. Course 1 everything is so new, lots of steps are taking place and one step leads to the next. I would also say that course 3 was also hard as well. It was working with Big Query, but if you didn't have the right code you were not able to succeed.