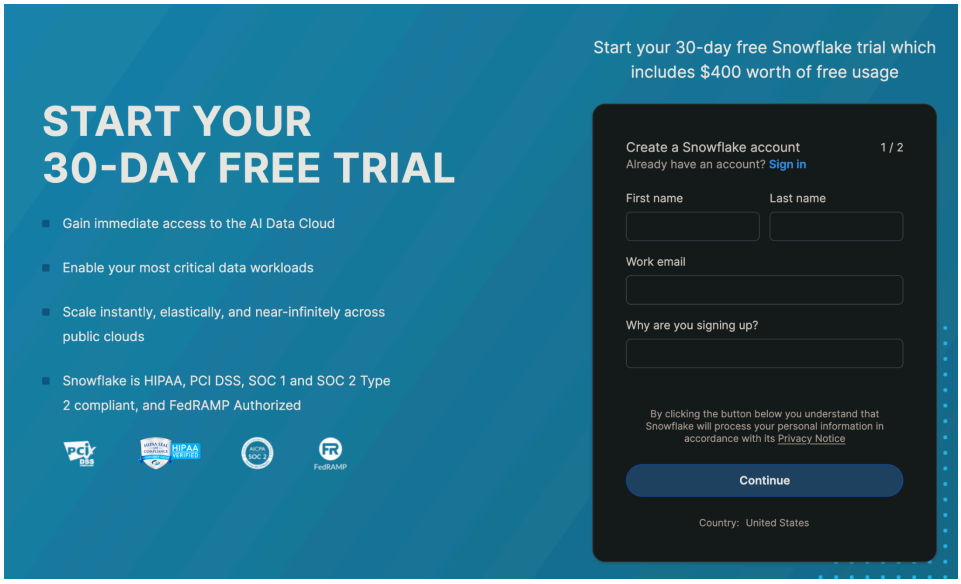
Table of Contents

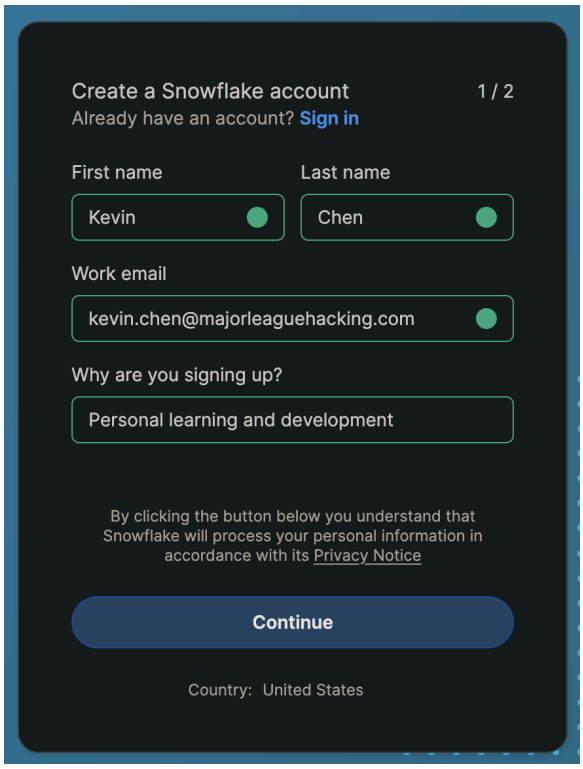
Module 1: Setting Up

Signing-up for Snowflake Trail Account

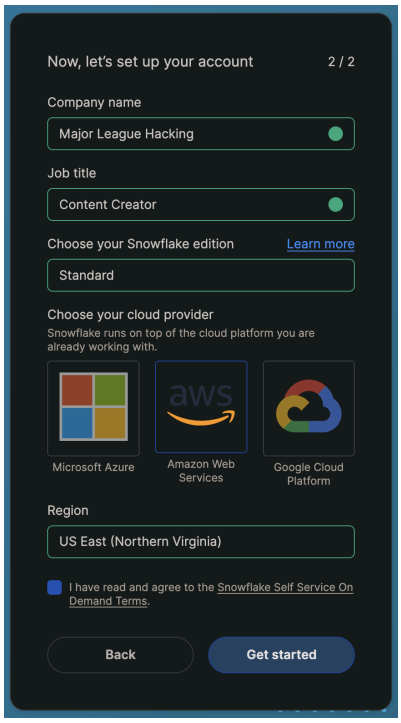
1. Visit mlh.link/snowflake-de-trial



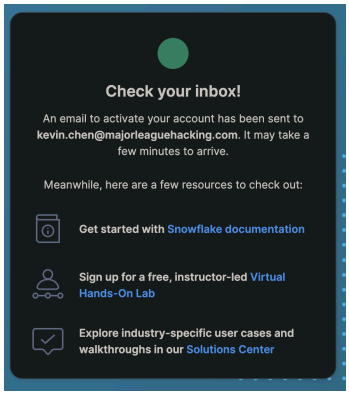
2. Fill out all the information and select “Personal learning and development”

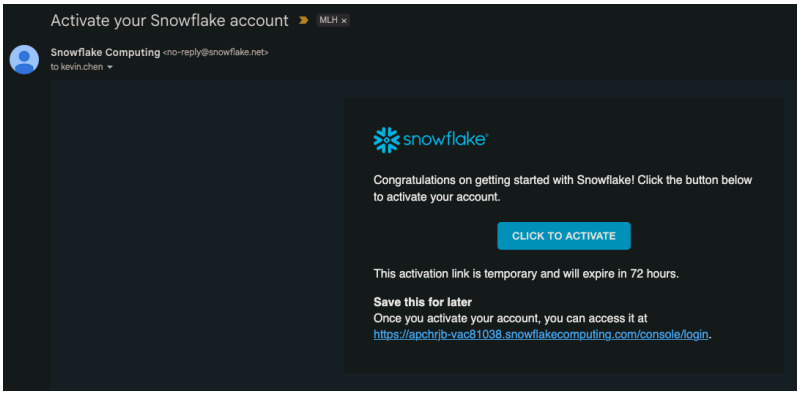


3. On the next page, enter your personal details (company name and role can be fictional). Choose the Standard edition, as it includes all AI/ML features at $2/credit, so your trial credits will stretch further, but any edition will work for this workshop. You should see your cloud provider and region pre-selected for you, with the value of AWS and US West (Oregon). If for any reason you do not see that, select AWS and US West (Oregon) manually.

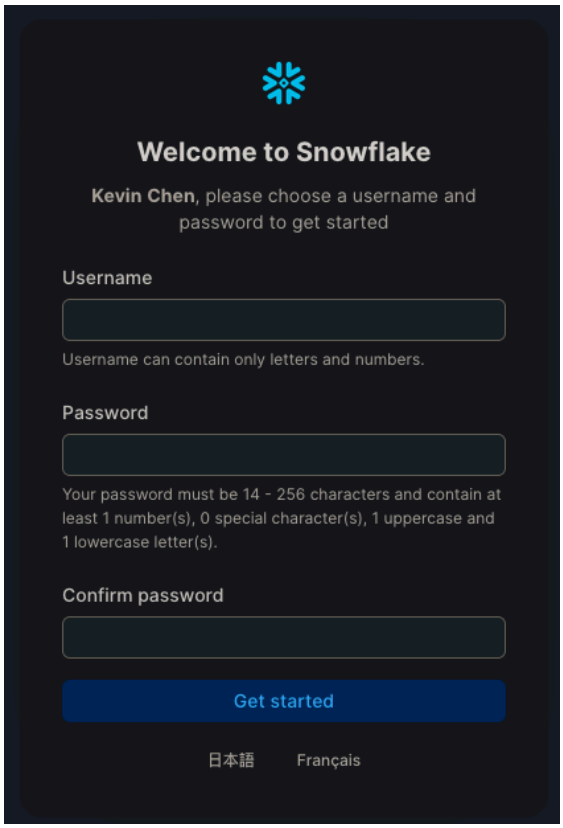


4. Answer the next two optional pages if desired
5. Check for an email and activate your account!

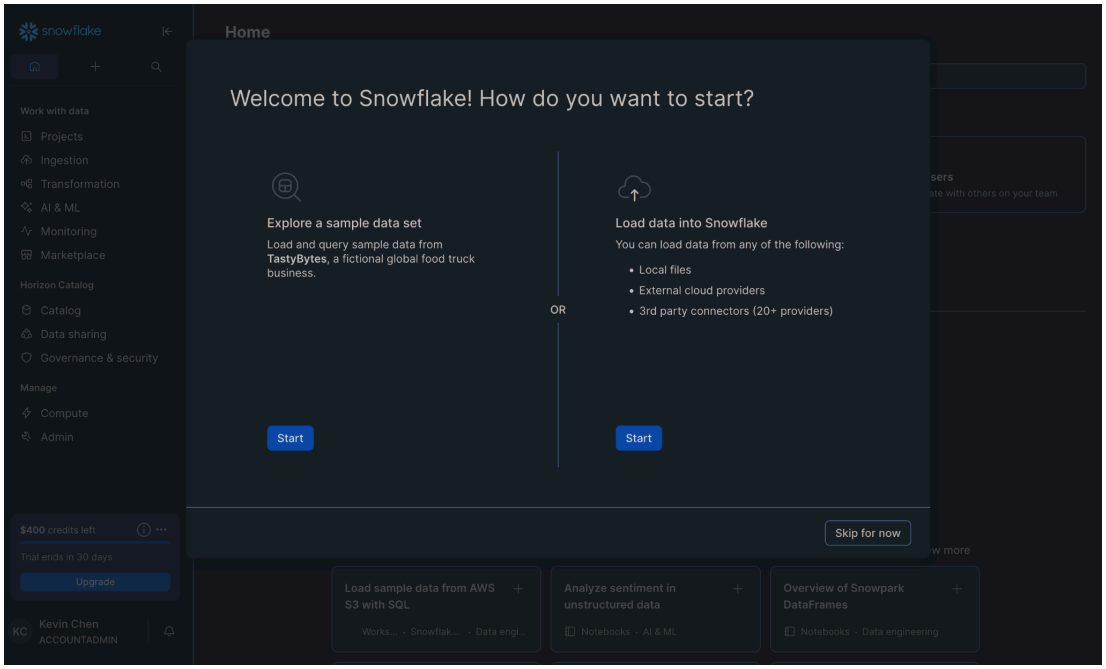




6. Set up your user account for your new Snowflake account.



7. Perfect, leave that page up, and you are all setup to continue back to the slides!



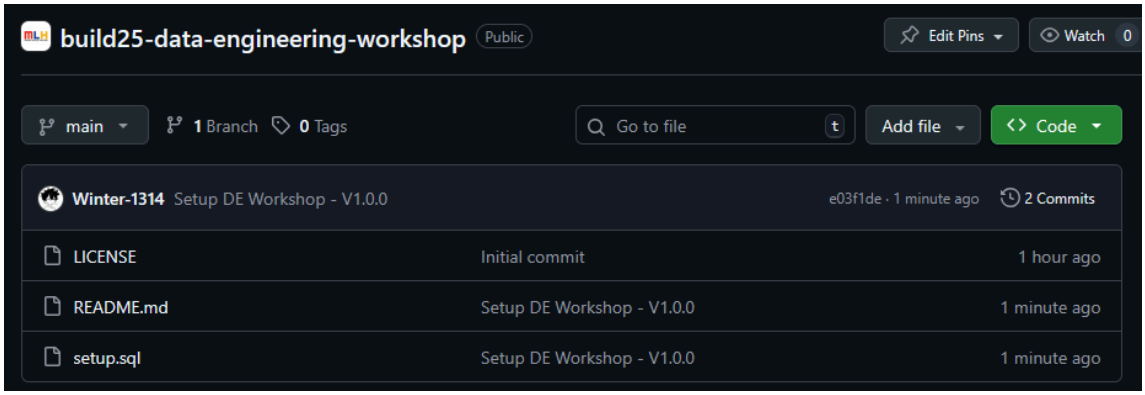
Loading in the Data

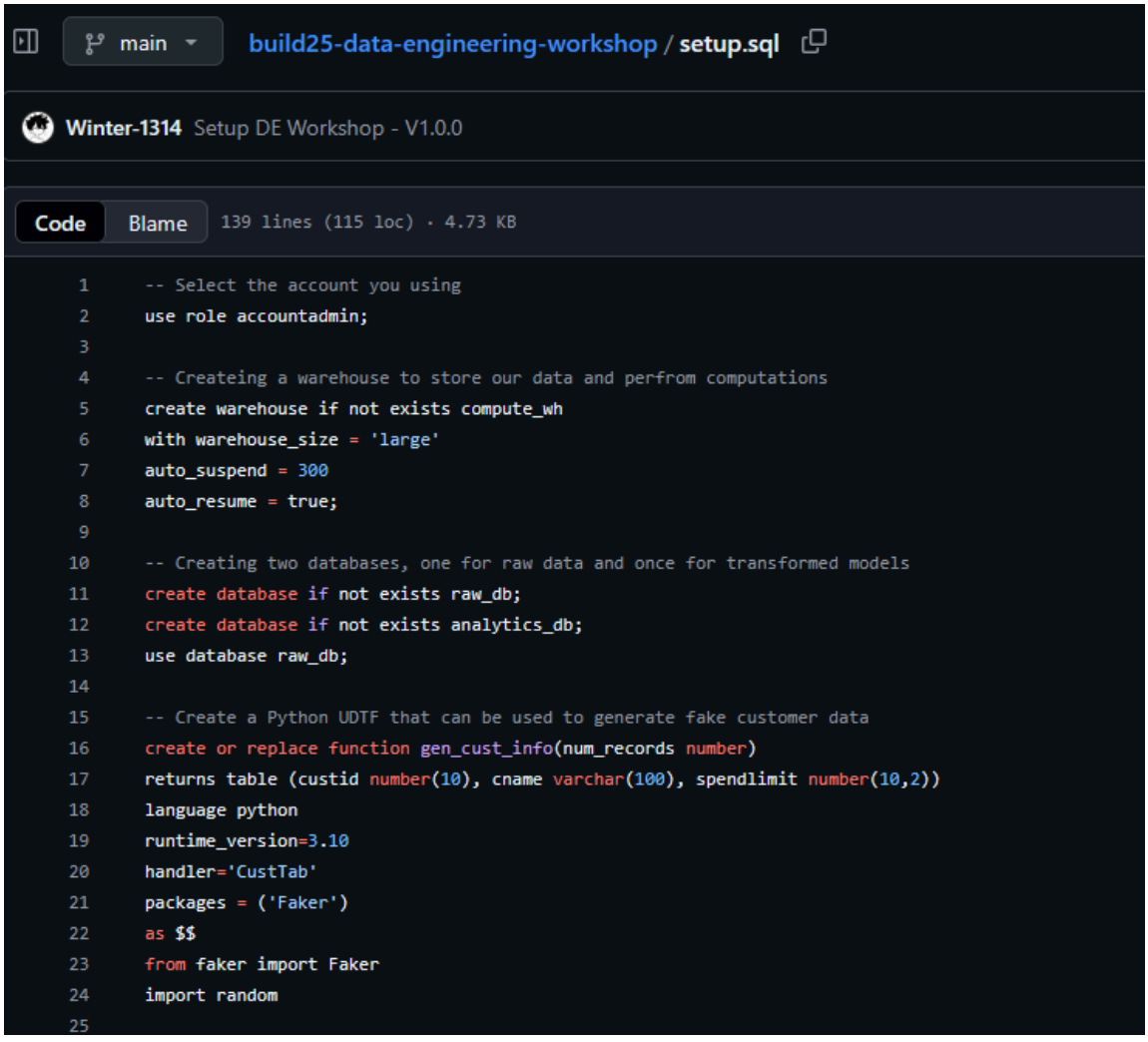
https://github.com/MLH/build25-data-engineering-workshop

Setting Up Account

`Showcase setup.sql`

1. Show the attendees the `setup.sql` script

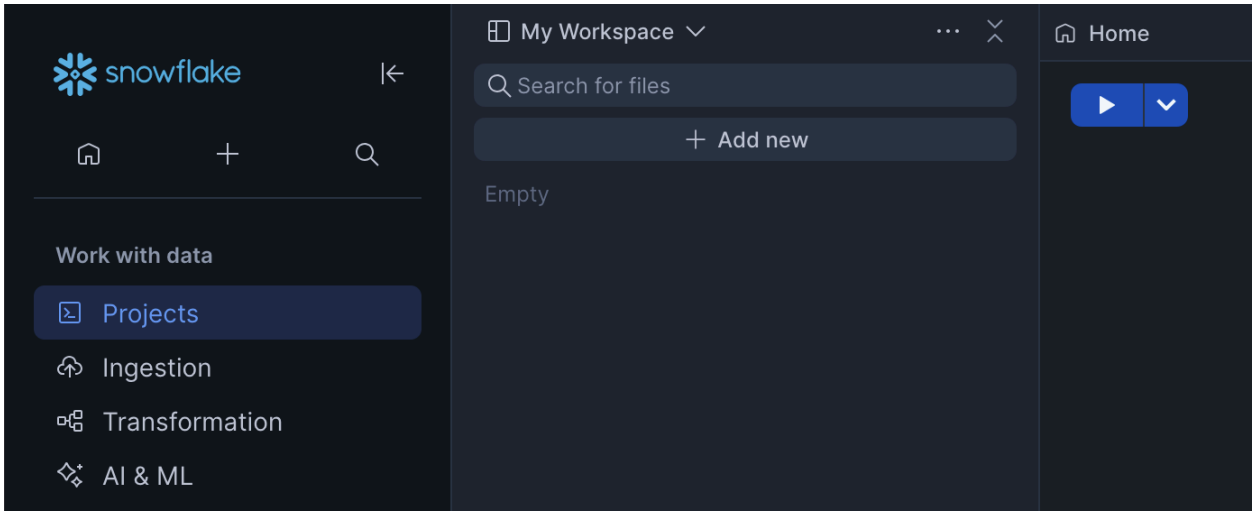




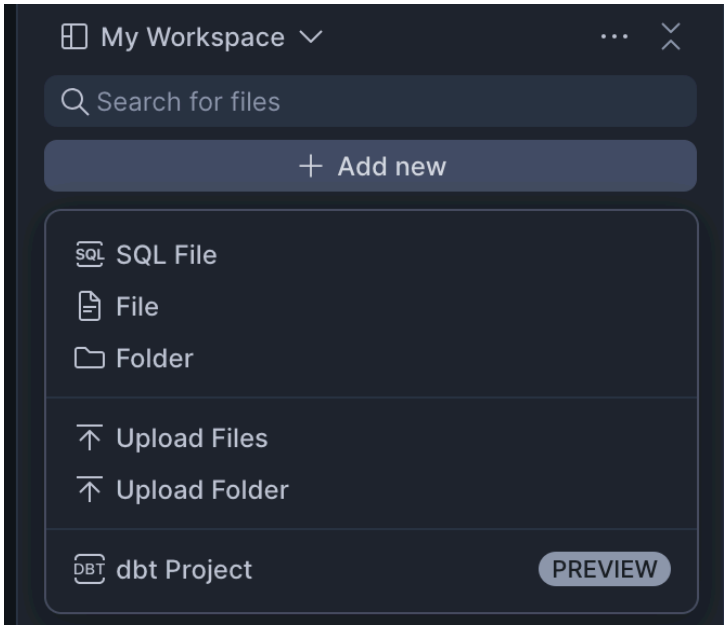
2. Scroll through and explain what the script is doing.
    a. 1: Switch to the ACCOUNTADMIN role
    b. 5-8: Create the Warehouse that will be used for compute/storage
    c. 10-13: Create a database called RAW_DB for raw data and one called ANALYTICS_DB for transformed models
    d. 16-122: Create three different User Defined Table Function that will generate mock data for a customer, products, and orders tables
    e. 125-129: Create the three tables using the UDTF that was just created
    f. 132-136: Allows you to run and validate the mock data in each of the three tables
    g. 139: Display a setup completion confirmation message.

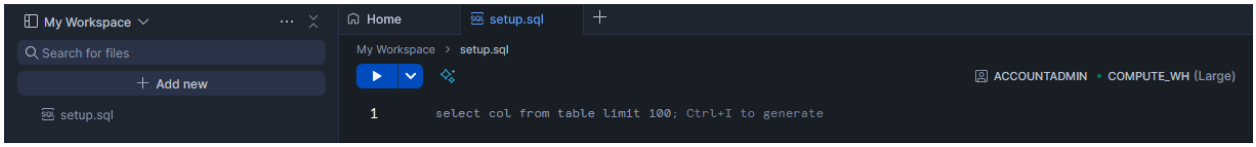
Creating and Executing a SQL Files in Snowflake Workspace

1. Go to “Projects” which should take you to “My Workspace”

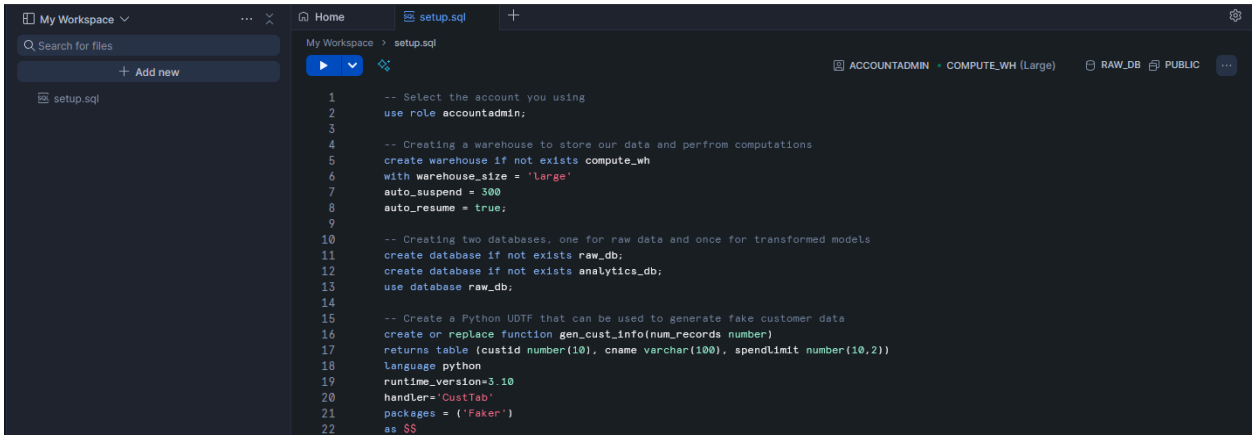


2. Click on "+ Add new" and select "SQL File" and name it `setup.sql`



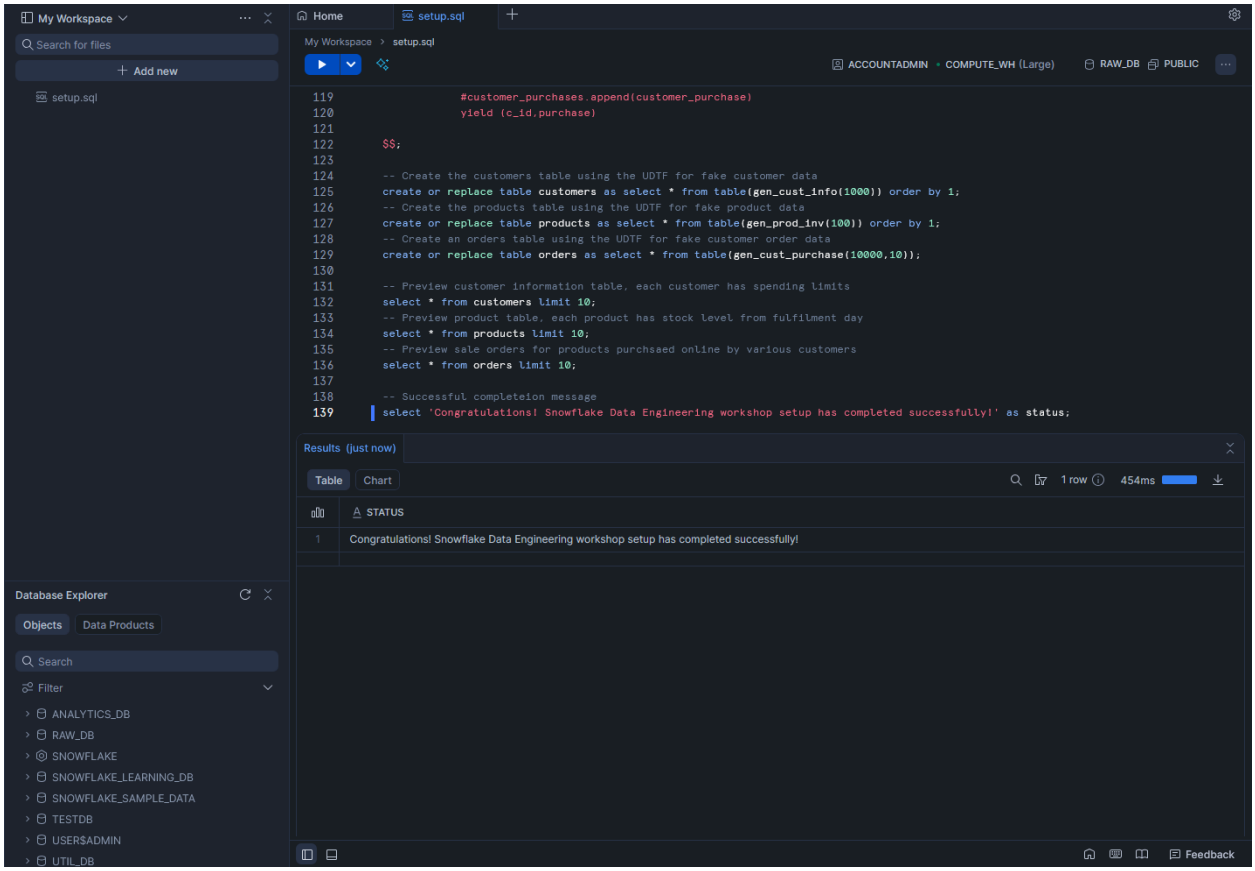


3. Navigate back to the GitHub repo, open `setup.sql`, copy its contents, and paste them into the SQL file you created in your workspace.

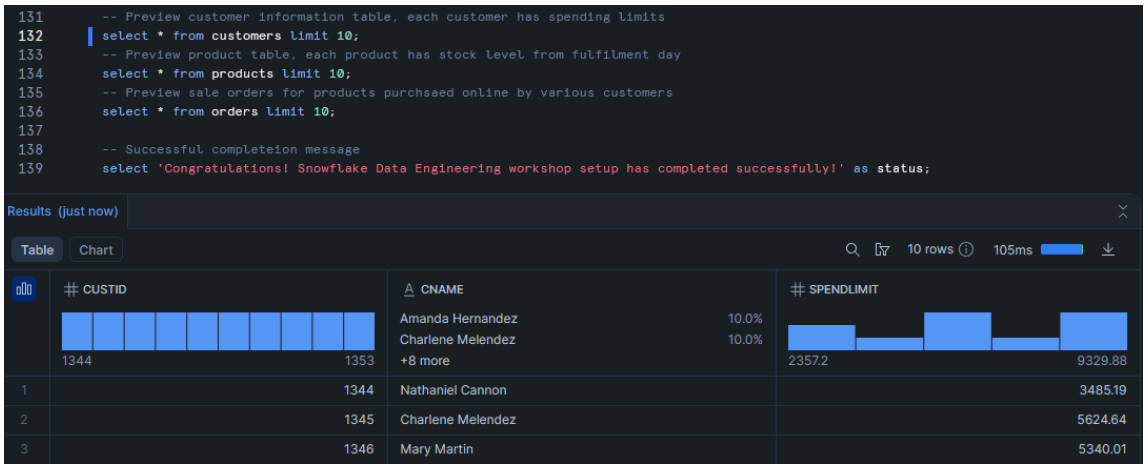


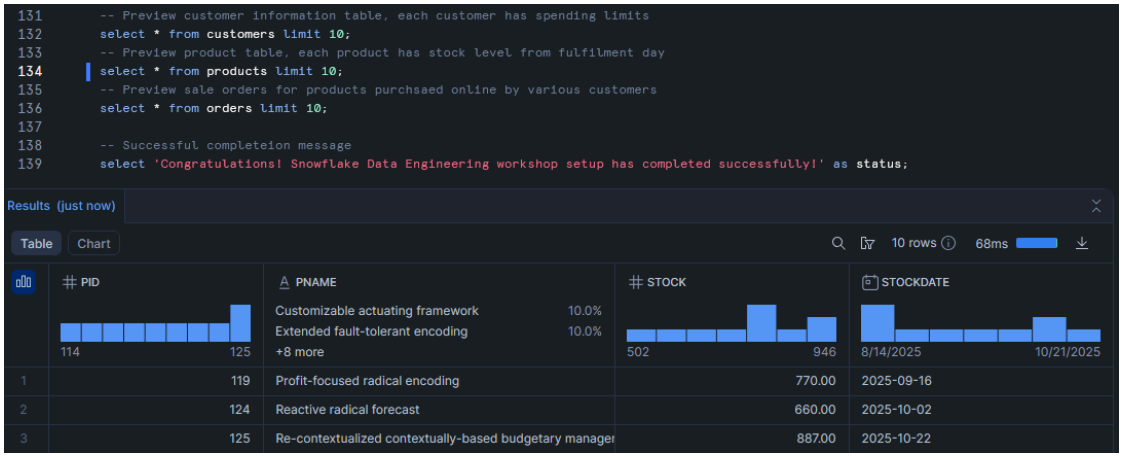
4. If you haven't yet, briefly explain the purpose of `setup.sql`: it automates role, warehouse, database, UDTF, and table creation. Click "Run All" to execute the entire script. You should see: "Congratulations! Snowflake Data Engineering workshop

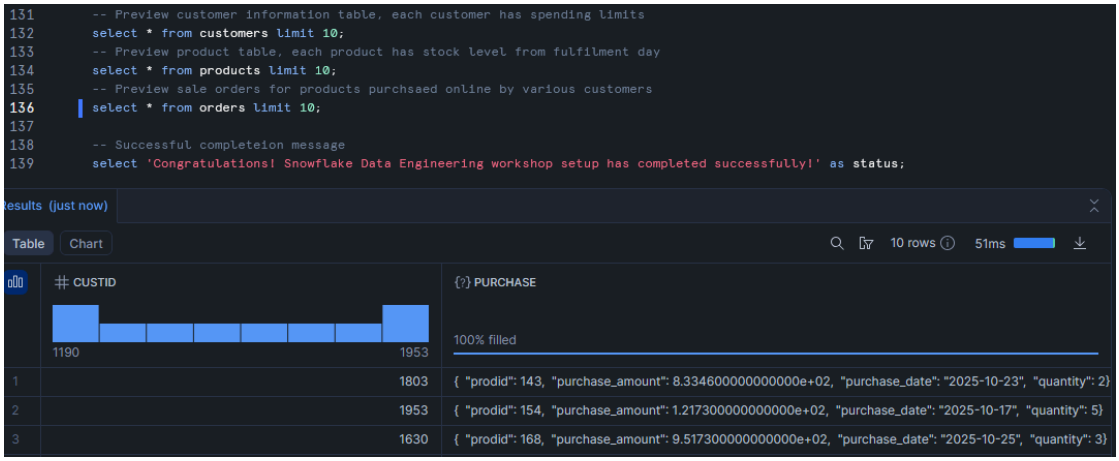
setup has completed successfully!"



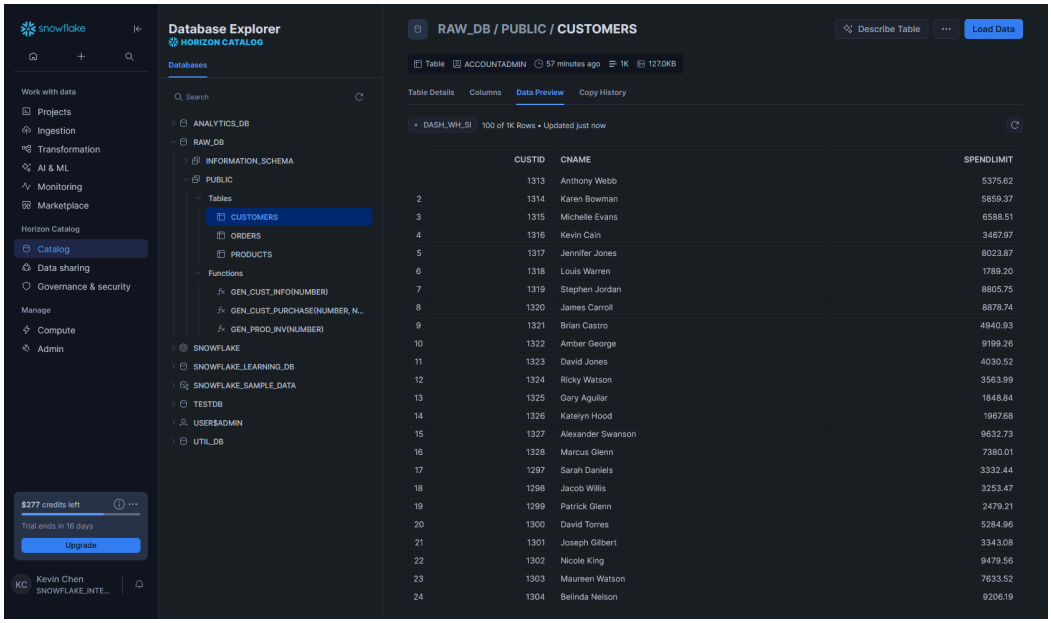
5. Go back and run lines 132, 134, and 136 to preview all the data that was created for each of the three tables: CUSTOMERS, PRODUCTS, and ORDERS







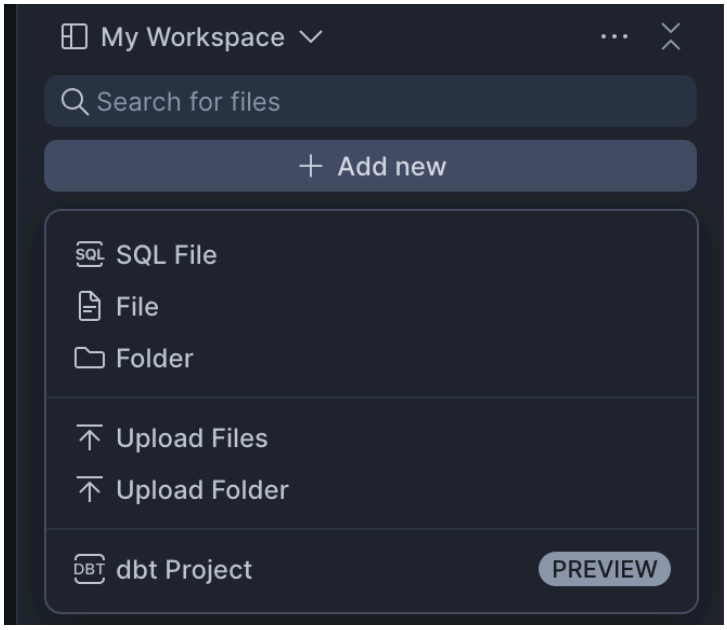
6. Finally, verify the script ran successfully by going to “Catalog” and checking for “RAW_DB/PUBLIC” DB which should list three tables and three functions populated.



Module 2: Dynamic Tables

Creating Dynamic Tables

1. Click on "+ Add new" and select "SQL File" and name it `create-dt.sql`



2. Type in the following lines to select the account, warehouse, and database you are using.

```
a. use role accountadmin;
b. use warehouse compute_wh;
c. use database analytics_db;
```

3. Type in the following lines to display what type of data and example data in the customers table. The goal here is to identify columns that we can perform light transformations like renaming columns and basic type casting.

```
a. desc table raw_db.public.customers;
b. select * from raw_db.public.customers;
```

c.

Results (just now)

Table | Chart | 3 rows | 77ms

	name	type	kind	null?	default	primary key	unique key	check	expression
1	CUSTID	NUMBER(10,0)	COLUMN	Y	null	N	N	null	null
2	CNAME	VARCHAR(100)	COLUMN	Y	null	N	N	null	null
3	SPENDLIMIT	NUMBER(10,2)	COLUMN	Y	null	N	N	null	null

	CUSTID	CNAME	SPENDLIMIT
	1001 – 2000	Jessica Scott 0.2%, Luis Wilson 0.2%, +98 more	1002.97 – 9993.53
1	1407	Jasmine Phillips	9529.00
2	1408	Mr. Todd Watkins	7682.46

4. Type in the following lines to create a Dynamic Table for `CUSTOMERS` table with the transformations that we selected.

```
a. create or replace dynamic table stg_customers_dt
b.     target_lag=downstream
c.     warehouse=compute_wh
d.     as select
e.         custid as customer_id,
f.         cname as customer_name,
g.         cast(spendlimit as float) as spend_limit
h.     from raw_db.public.customers;
```

i.

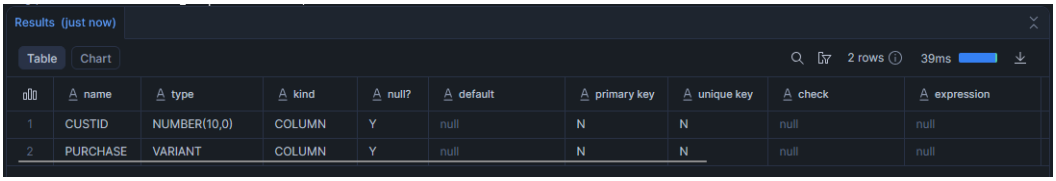
Results (just now)

Table | Chart

	status
1	Dynamic table STG_CUSTOMERS_DT successfully created.

5. Type in the following lines to display what type of data and example data in the orders table. For this table, you will notice that the purchase column holds a JSON value. This means when we are doing the transformations, we can pull those values into their own columns

```
a. desc table raw_db.public.orders;
b. select* from raw_db.public.orders;
```

c. 

Results (just now)

Table | Chart — 2 rows — 39ms

	A name	A type	A kind	A null?	A default	A primary key	A unique key	A check	A expression
1	CUSTID	NUMBER(10,0)	COLUMN	Y	null	N	N	null	null
2	PURCHASE	VARIANT	COLUMN	Y	null	N	N	null	null

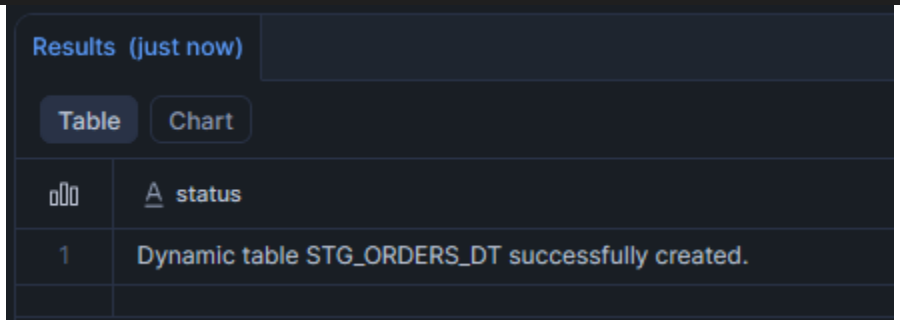
Results (just now)

Table | Chart — 10,000 rows — 61ms

	# CUSTID	{?} PURCHASE
1	1803	{ "prodid": 143, "purchase_amount": 8.334600000000000e+02, "purchase_date": "2025-10-23", "quantity": 2}
2	1953	{ "prodid": 154, "purchase_amount": 1.217300000000000e+02, "purchase_date": "2025-10-17", "quantity": 5}
3	1630	{ "prodid": 168, "purchase_amount": 9.517300000000000e+02, "purchase_date": "2025-10-25", "quantity": 3}

6. Type in the following lines to create a dynamic table for orders table. Note the unpacking of the JSON. The following format is used COLUMN:"VAR_NAME"::TYPE_CASE as NEW_COL_NAME

```
a. create or replace dynamic table stg_orders_dt
b.     target_lag=downstream
c.     warehouse=compute_wh
d.     as select
e.         custid as customer_id,
f.         purchase:"prodid"::number(5) as product_id,
g.         purchase:"purchase_amount"::float(10) as order_price,
h.         purchase:"quantity"::number(5) as quantity,
i.         purchase:"purchase_date"::date as order_date
j.     from raw_db.public.orders;
```

k. 

Results (just now)

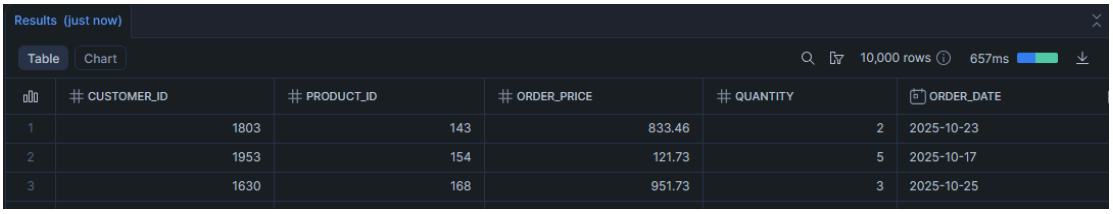
Table | Chart

	A status
1	Dynamic table STG_ORDERS_DT successfully created.

7. Type in the following lines to query the new Dynamic Tables that you created. You should see the new columns.

```
a. select * from analytics_db.public.stg_customers_dt;
b. select * from analytics_db.public.stg_orders_dt;
```

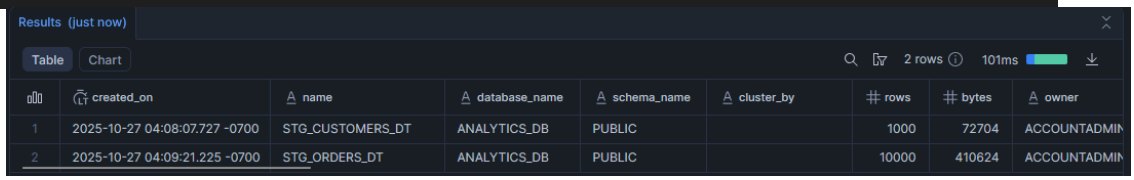
c.

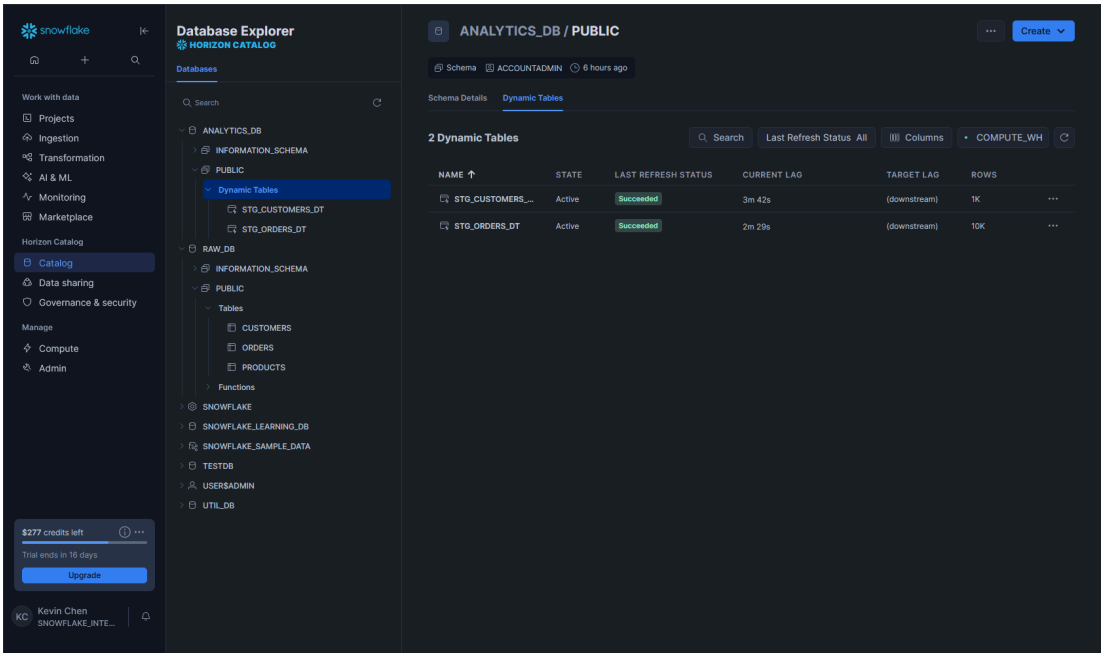


8. Type in the following lines to displays all the Dynamic Tables that we have created. Scroll to the right and find `TARGET_LAG` can make sure to call out that this how often the Dynamic Tables will refresh its data. In our case, it's set to `DOWNSTREAM`, which means whenever the downstream table is changed, our Dynamic Tables will refresh. You can also go to “Catalog” and see the two new Dynamic Tables.

a.
```
show dynamic tables;
```

b.

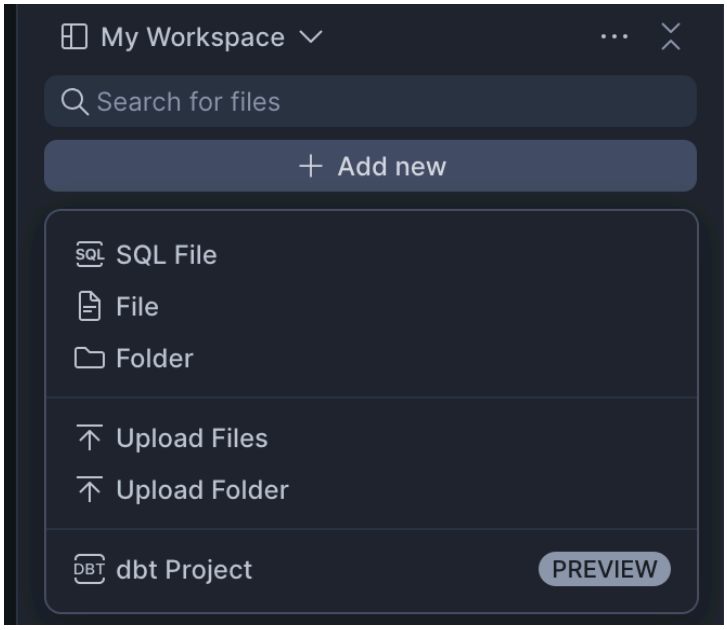




Module 3: Chaining Dynamic Tables

Creating a Fact Table

1. Click on "+ Add new" and select "SQL File" and name it `chaining-dt.sql`



2. Type in the following lines to select the account, warehouse, and database you are using.

```
a. use role accountadmin;
b. use warehouse compute_wh;
c. use database analytics_db;
```

3. Type in the following lines to check the values in each Dynamic Tables. You are doing this do you can decide what values you want to carry over into your Fact Table.

```
a. select * from analytics_db.public.stg_customers_dt;
b. select * from analytics_db.public.stg_orders_dt;
```

4. Type in the following lines to create a fact dynamic table for customer orders. We are going to take the `customer_id` and `customer_name` from the `STG_CUSTOMERS_DT` Dynamic Table and then the `product_id`, `order_price`, `quantity`, and `order_date` from the `STG_ORDERS_DT` Dynamic Table. Make sure to talk about how all you needed to do is simply reference the upstream Dynamic Tables by name and Snowflake automatically discovers the dependency needed.

```
a. create or replace dynamic table fct_customer_orders_dt
b.     target_lag=downstream
c.     warehouse=compute_wh
d.     as select
e.         c.customer_id,
f.         c.customer_name,
g.         o.product_id,
h.         o.order_price,
i.         o.quantity,
j.         o.order_date
k.     from stg_customers_dt c
l.     left join stg_orders_dt o
m.         on c.customer_id = o.customer_id;
```

5. Type in the following lines to query the new Fact Model Dynamic Table that you created.

```
a. select * from analytics_db.public.fct_customer_orders_dt;
```

b.

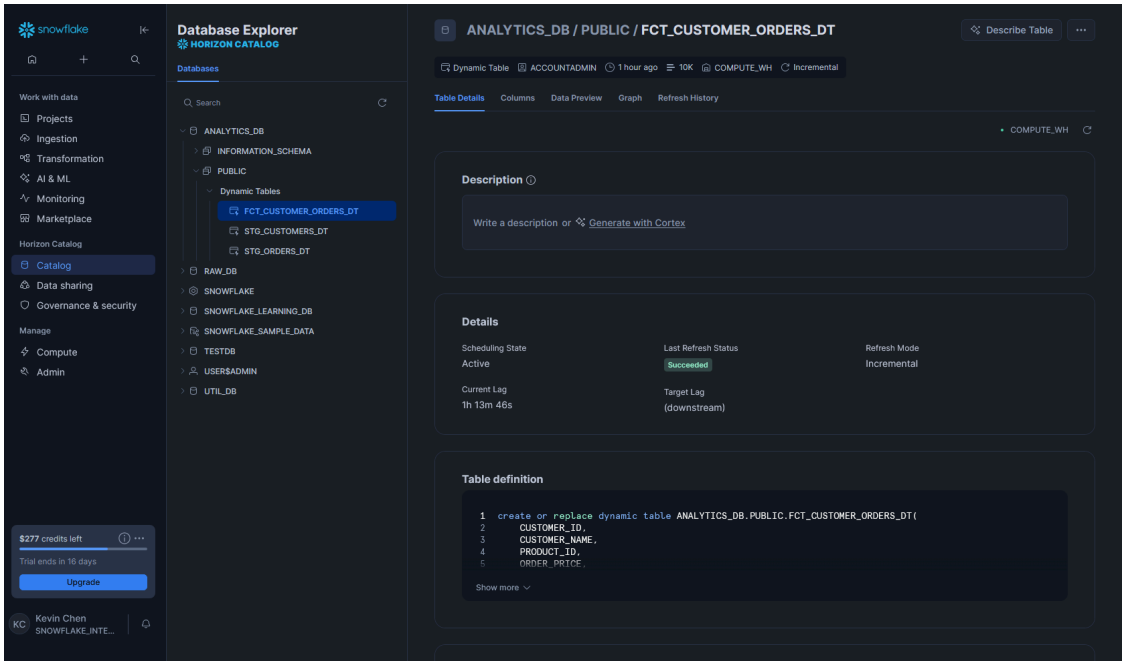
Results (just now)

Table | Chart | 10,001 rows | 679ms

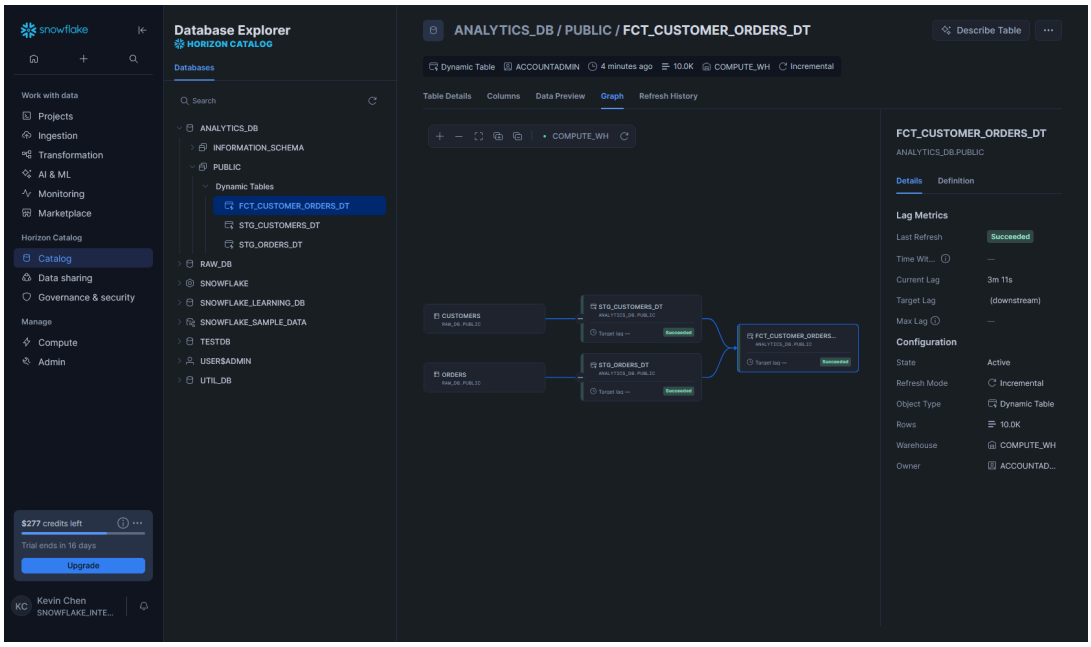
	CUSTOMER_ID	CUSTOMER_NAME	PRODUCT_ID	ORDER_PRICE	QUANTITY	ORDER_DATE
1	2000	Andrew Lara	null	null	null	null
2	1803	Erin Warren	143	833.46	2	2025-10-23
3	1953	Christopher Armstrong PhD	154	121.73	5	2025-10-17
4	1630	Rebecca Wright	168	951.73	3	2025-10-25
5	1190	James Gaines	161	409.26	1	2025-10-22

Visualize the Pipeline

1. Navigate to “Catalog” and under “Dynamic Table” select FCT_CUSTOMER_ORDER_DT.



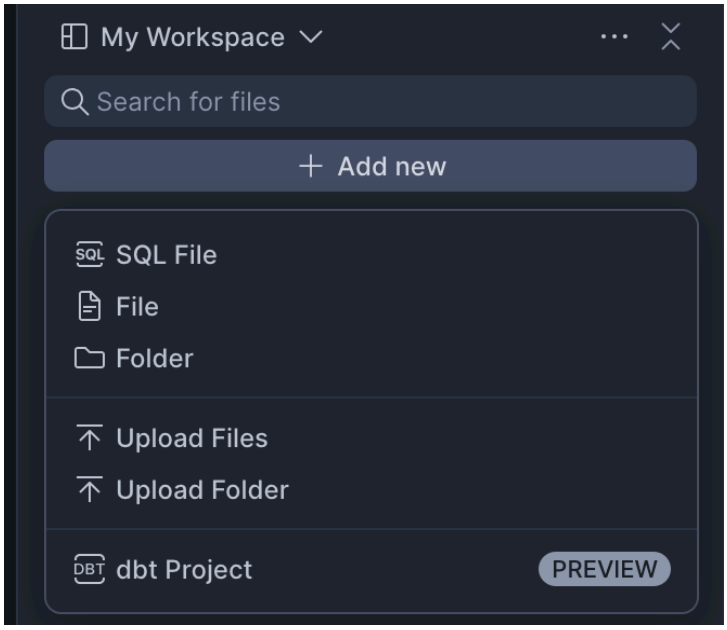
2. Click on “Graph”. You should now see a visual DAG (Directed Acyclic Graph) of the pipeline: Raw Tables -> Staging Dynamic Tables -> Fact Dynamic Table. If you need to click the + and zoom out.



Module 4: Pipeline Monitoring

Showcase Pipeline Monitoring Techniques

1. Click on "+ Add new" and select "SQL File" and name it `pipeline.sql`



2. Type in the following lines to select the account, warehouse, and database you are using.

```
a. use role accountadmin;
b. use warehouse compute_wh;
c. use database analytics_db;
```

3. Type in the following lines to displays all the Dynamic Tables that we have created. We just want to show the attendees that the current value for TARGET_LAG is `DOWNSTREAM`.

```
a. show dynamic tables;
```

4. Type in the following lines to adjust the freshness, meaning the `TARGET_LAG` for `STG_ORDERS_DT`. Explain that any downstream tables will automatically adjust to this new cadence, since they update whenever changes occur upstream.

```
a. alter dynamic table stg_orders_dt set target_lag = '5
   minutes';
```

5. Type in the following lines to check for the altered Dynamic Table's new `TARGET_LAG` which now should say 5 minutes.

```
show dynamic tables;
```

6. Type in the following lines to monitoring pipeline health, this is how you inspect the history of refreshes, showing execution times, data changes, and potential errors. This is your built-in observability tool in SQL.

```
select * from
table(information_schema.dynamic_table_refresh_history());
```

b.

Results (just now)

Table | Chart | 75 rows | 361ms

	NAME	SCHEMA_NAME	DATABASE_NAME	STATE	STATE_CODE	STATE_MESSAGE	QUERY_ID
	STG_ORDERS_DT 68.0% STG_CUSTOMERS_DT 20.0% +1 more	PUBLIC 100.0%	ANALYTICS... 98.7% RAW_DB 1.3%	SUCCE... 100.0%	SUCCESS 100.0%	All values are null	01bffc15-0000-4214-00 01bffc23-0000-4214-00 +73 more
1	FCT_CUSTOMER_ORDERS_DT	PUBLIC	ANALYTICS_DB	SUCCEEDED	SUCCESS	null	01bffd21-0000-4231-00
2	STG_CUSTOMERS_DT	PUBLIC	ANALYTICS_DB	SUCCEEDED	SUCCESS	null	01bffd21-0000-4231-00
3	STG_ORDERS_DT	PUBLIC	ANALYTICS_DB	SUCCEEDED	SUCCESS	null	01bffd21-0000-4231-00
4	STG_ORDERS_DT	PUBLIC	ANALYTICS_DB	SUCCEEDED	SUCCESS	null	01bffd1d-0000-4214-00
5	STG_ORDERS_DT	PUBLIC	ANALYTICS_DB	SUCCEEDED	SUCCESS	null	01bffd1c-0000-41d4-00

7. Type in the following lines to query the Fact Dynamic Table and check for potential issues, we are looking to see if there is any null order, sometimes there is, sometimes there isn't. If there is, call it out, if not move to the next step but call out that there could be null for `PRODUCT_ID` if the user didn't purchase anything.

```
select * from analytics_db.public.fct_customer_orders_dt;
```

8. Type in the following lines to integrated data quality to remove null orders from the `FCT_CUSTOMER_ORDERS_DT`.

```
create or replace dynamic table fct_customer_orders_dt
    target_lag=downstream
    warehouse=compute_wh
    as select
        c.customer_id,
        c.customer_name,
        o.product_id,
        o.order_price,
        o.quantity,
        o.order_date
    from stg_customers_dt c
    left join stg_orders_dt o
        on c.customer_id = o.customer_id
```

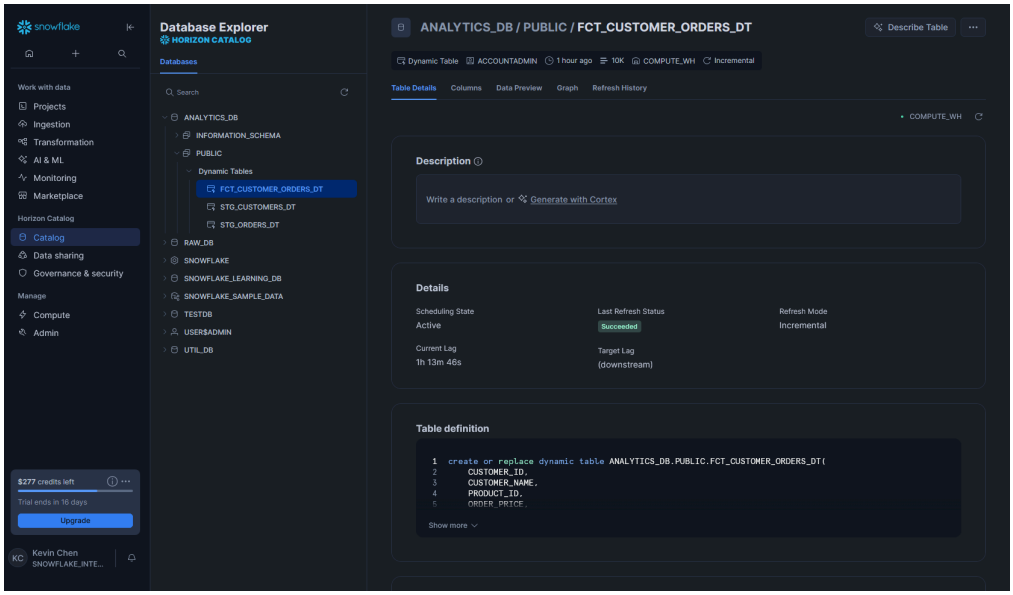
```
n.      where o.product_id is not null;
```

9. Type in the following lines to check if the data quality enforcement works

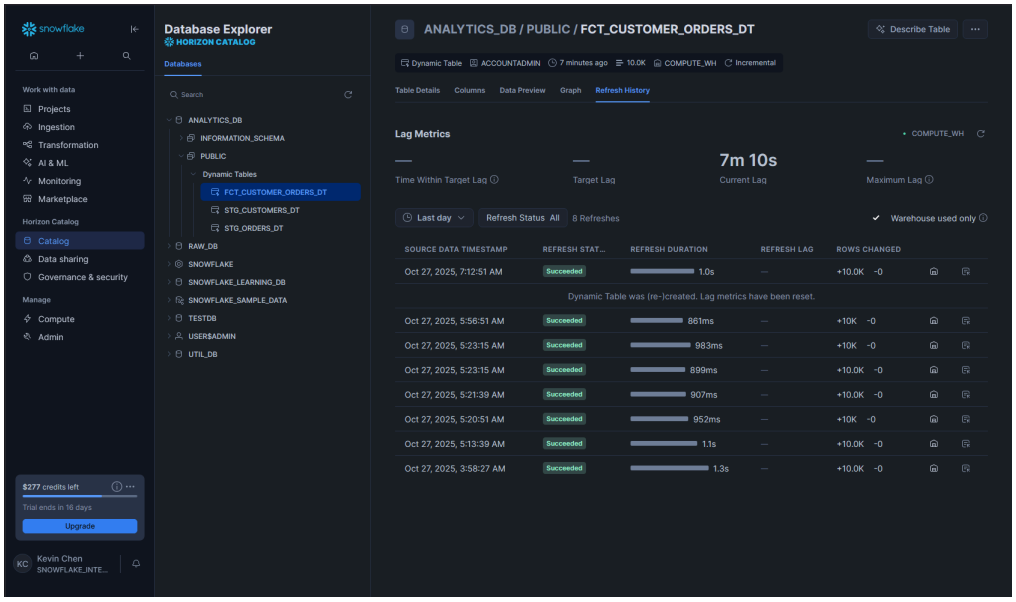
```
a. select * from analytics_db.public.fct_customer_orders_dt;
```

Pipeline Management in Snowsight

1. Navigate to “Catalog” and under “Dynamic Table” select FCT_CUSTOMER_ORDER_DT.

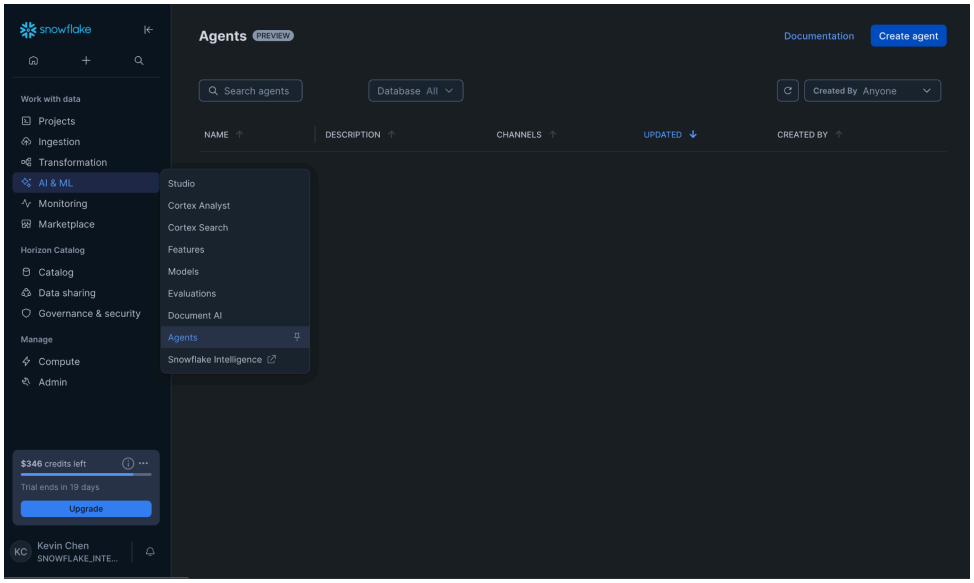


2. Click on “Refresh History”. You should now see a table of information on when the Dynamic Table was refreshed and the status of the refresh.

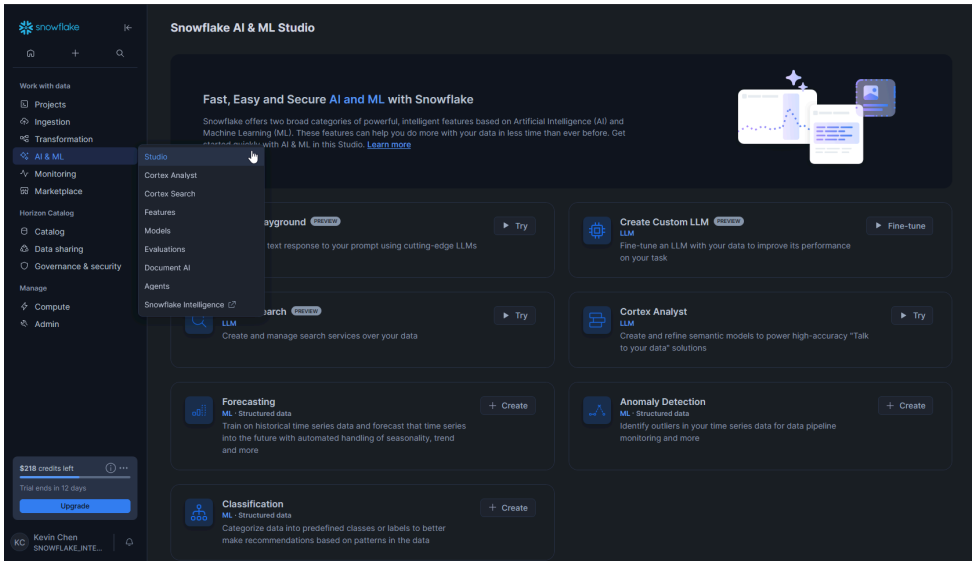


Module 5: Snowflake Intelligence

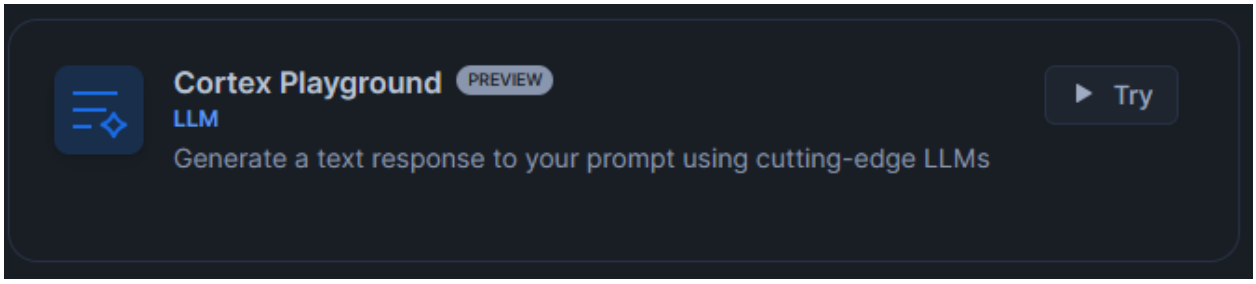
1. Resources to explore Snowflake Intelligence
    a. Snowflake Intelligence Overview
    b. Snowflake Cortex Agents
    c. Snowflake Cortex Analyst
    d. Snowflake Cortex Search
    e. Understanding Snowflake Cortex
    f. Tutorial: Getting Started with Snowflake Intelligence
2. Show the attendees the “AI/ML” section



3. Navigate to “Studio” where you can see the “Snowflake AI & ML Studio”

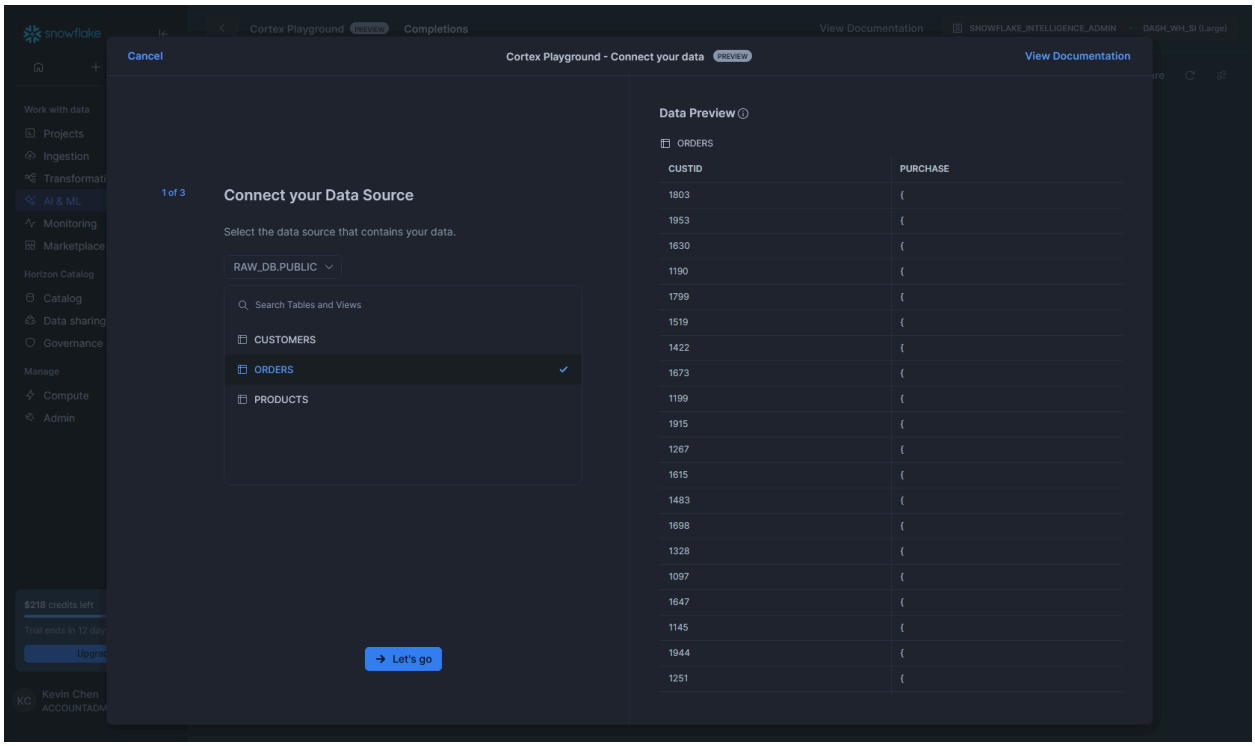


4. Select “Try” for Cortex Playground. Note that we will be testing using the Cortex Playground, which does not directly support Dynamic Tables. To use Dynamic Tables, you would need to leverage Cortex Search and create a custom Agent, which attendees can explore further if they’re interested.

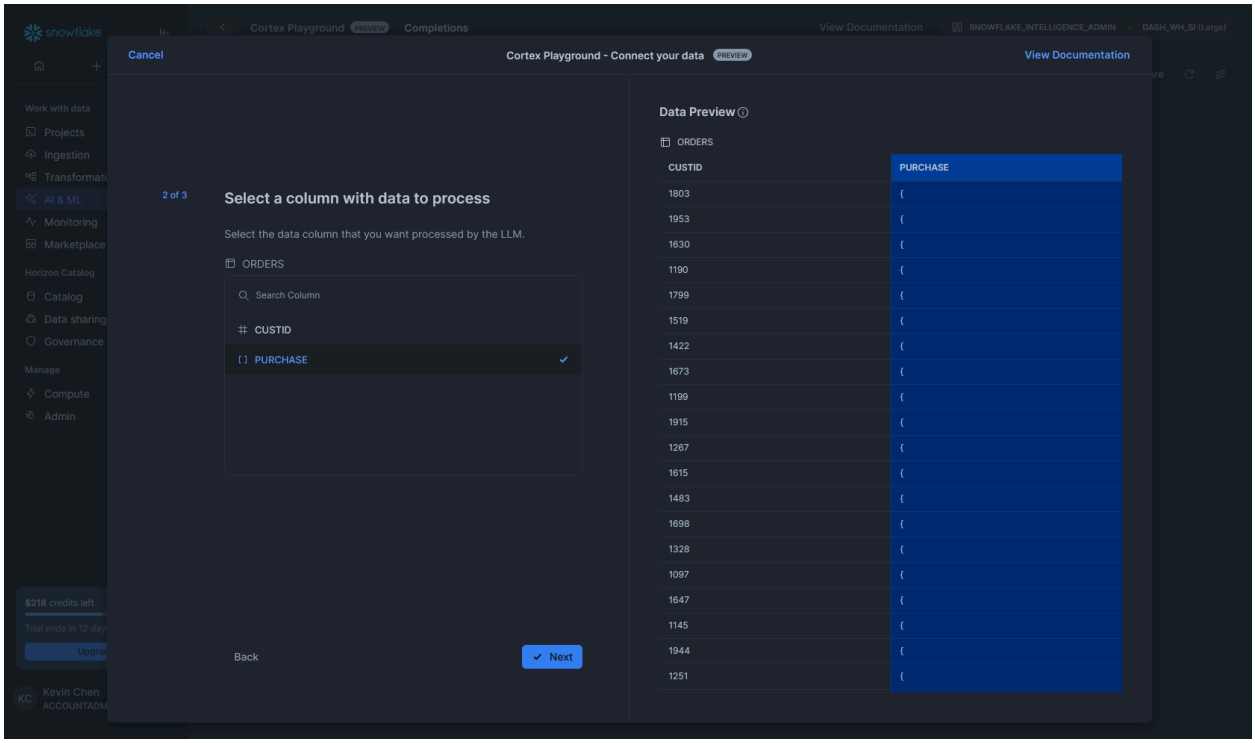


5. Select “Connect your data”

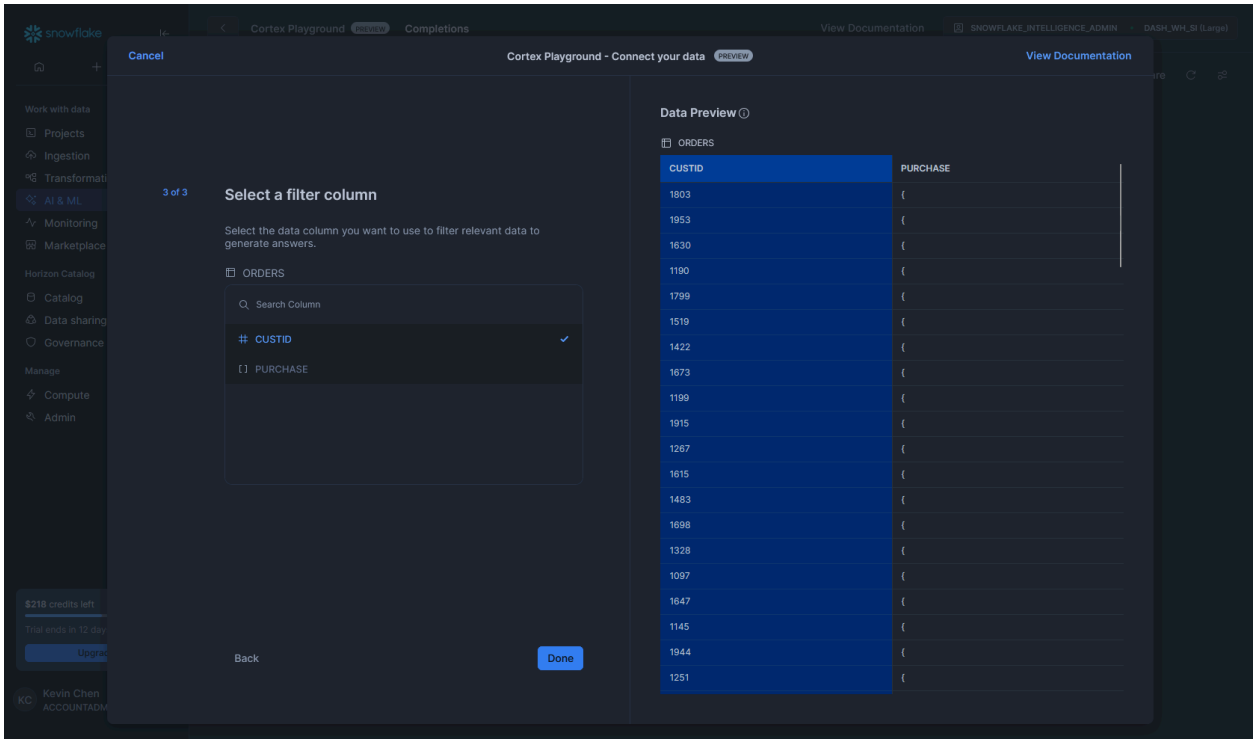
6. Select RAW_DB.PUBLIC.ORDERS for your Data Source, then click “Let’s go”



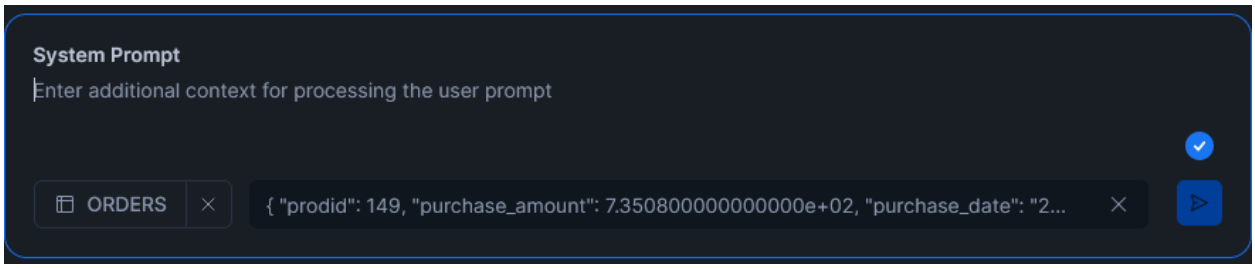
7. Select “PURCHASE” then click “Next”



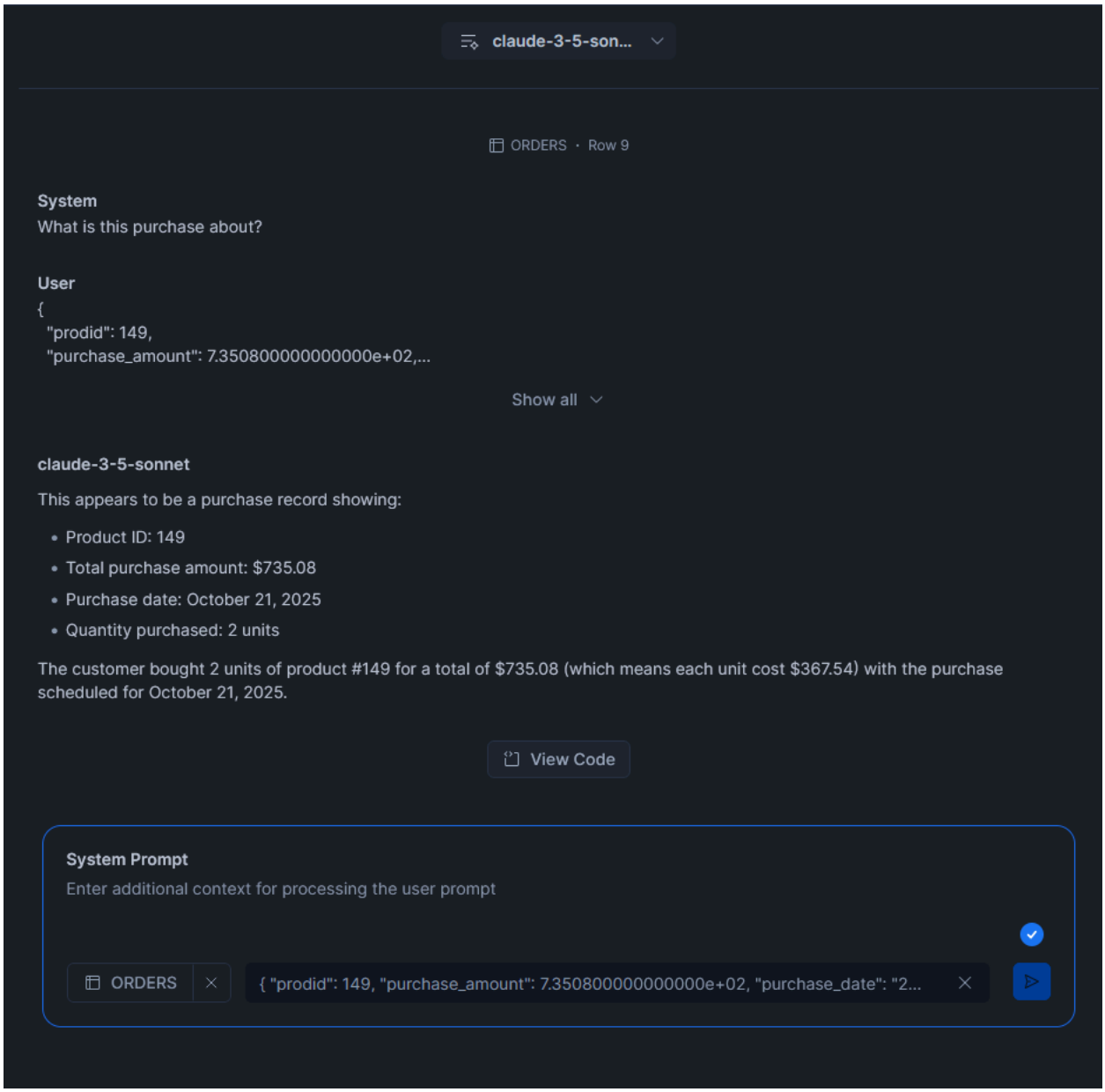
8. Select “CUSTID” then click “Done”



9. Now select which Custom's Order do you want the AI to have access to and be able to answer questions about. Pick any "CUSTID"



10. Ask it a question like "What is this purchase about?"



Module 6: Snowflake Badge

Setup Autograder

1. Follow the instructions in the Autograder setup repo at https://mlh.link/snowflake-autograder

Run Autograding Scripts

1. Run the `snowflake-intelligence.sql` script at https://mlh.link/snowflake-autograding-scripts step by step after you set up the Autograder