

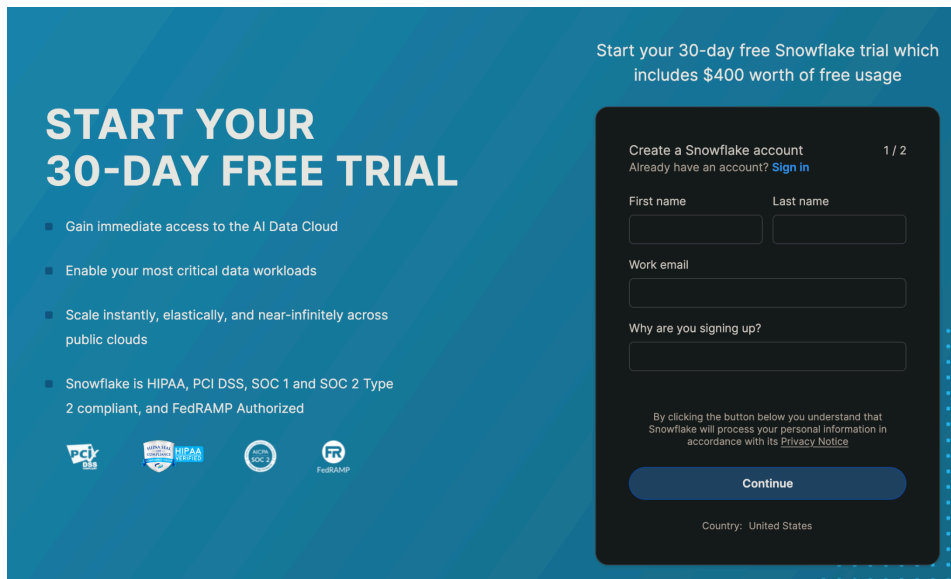
Table of Contents

| | |
|--|-----------|
| Module 1: Setting Up..... | 2 |
| Signing-up for Snowflake Trail Account..... | 2 |
| Setting Up Account..... | 5 |
| Showcase setup.sql..... | 5 |
| Creating and Executing a SQL Files in Snowflake Workspace..... | 6 |
| Module 2: Dynamic Tables..... | 10 |
| Creating Dynamic Tables..... | 10 |
| Module 3: Chaining Dynamic Tables..... | 14 |
| Creating a Fact Table..... | 14 |
| Visualize the Pipeline..... | 16 |
| Module 4: Pipeline Monitoring..... | 17 |
| Showcase Pipeline Monitoring Techniques..... | 17 |
| Pipeline Management in Snowsight..... | 19 |
| Module 5: Building Intelligence..... | 20 |
| Module 6: Snowflake Badge..... | 24 |
| Setup Autograder..... | 24 |
| Run Autograding Scripts..... | 24 |

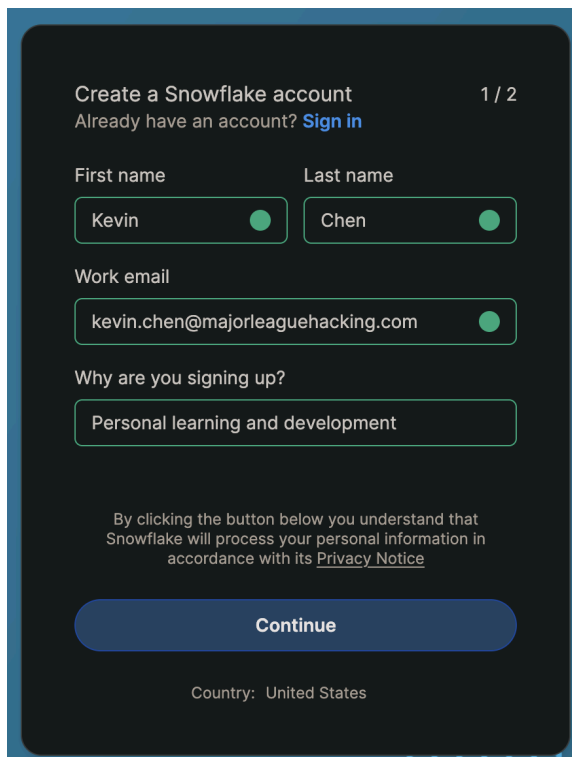
Module 1: Setting Up

Signing-up for Snowflake Trail Account

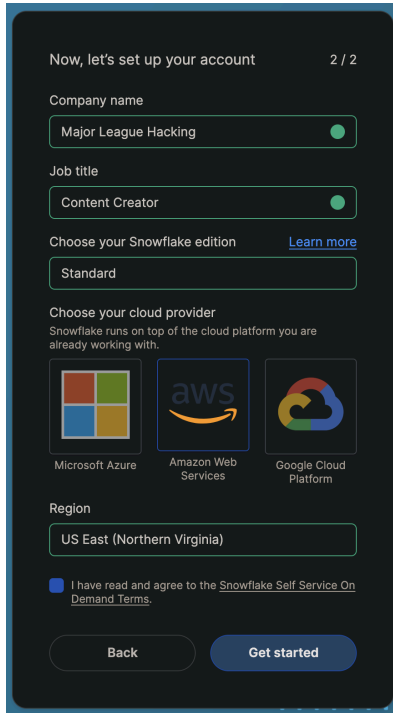
1. Visit mlh.link/snowflake-de-trial

The landing page for the Snowflake 30-day free trial. It features a blue background with white text. On the left, it says "START YOUR 30-DAY FREE TRIAL" and lists benefits: "Gain immediate access to the AI Data Cloud", "Enable your most critical data workloads", "Scale instantly, elastically, and near-ininitely across public clouds", and "Snowflake is HIPAA, PCI DSS, SOC 1 and SOC 2 Type 2 compliant, and FedRAMP Authorized". On the right, it says "Start your 30-day free Snowflake trial which includes \$400 worth of free usage". Below this is a form titled "Create a Snowflake account" with a progress indicator "1 / 2". The form includes fields for "First name", "Last name", "Work email", and "Why are you signing up?". There is a "Continue" button and a "Country" dropdown set to "United States".

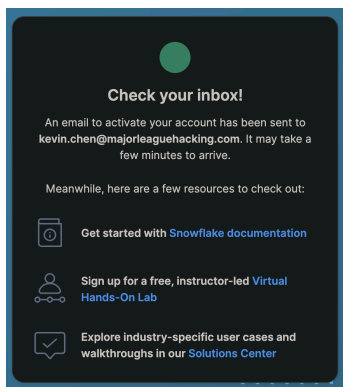
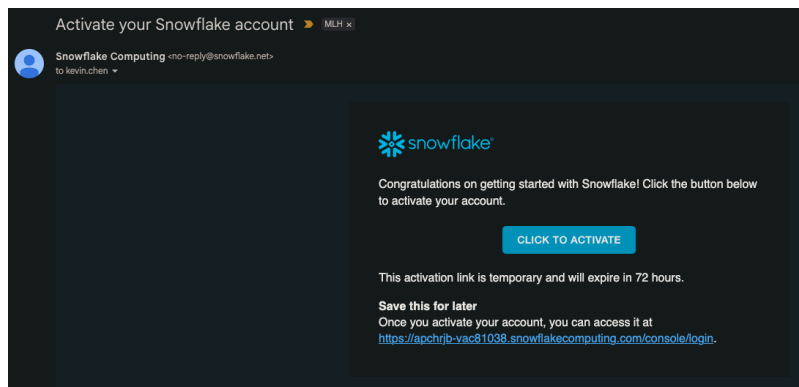
2. Fill out all the information and select "Personal learning and development"

A screenshot of the Snowflake account creation form. The form is titled "Create a Snowflake account" with a progress indicator "1 / 2". It includes a link "Already have an account? Sign in". The form has fields for "First name" (Kevin), "Last name" (Chen), "Work email" (kevin.chen@majorleaguehacking.com), and "Why are you signing up?" (Personal learning and development). There is a "Continue" button and a "Country" dropdown set to "United States".

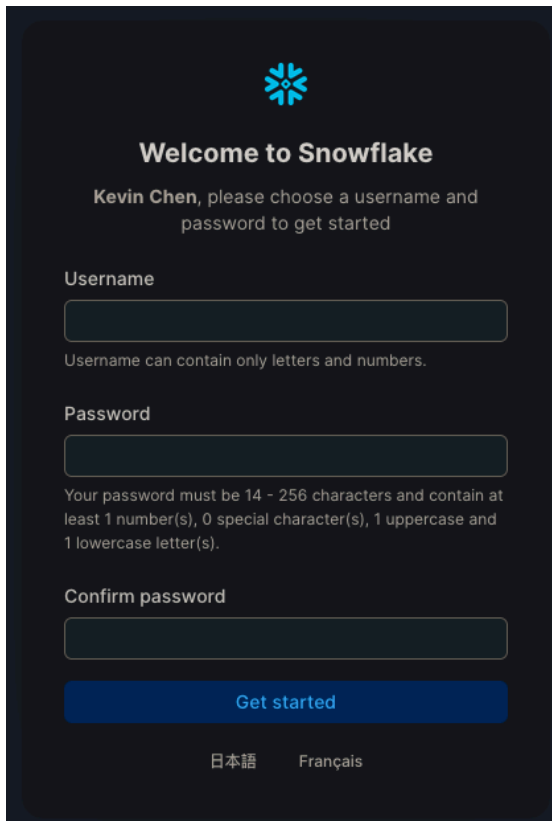
- On the next page, enter your personal details (company name and role can be fictional). Choose the Standard edition, as it includes all AI/ML features at \$2/credit, so your trial credits will stretch further, but any edition will work for this workshop. You should see your cloud provider and region 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.



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

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



Welcome to Snowflake

Kevin Chen, please choose a username and password to get started

Username

Username can contain only letters and numbers.

Password

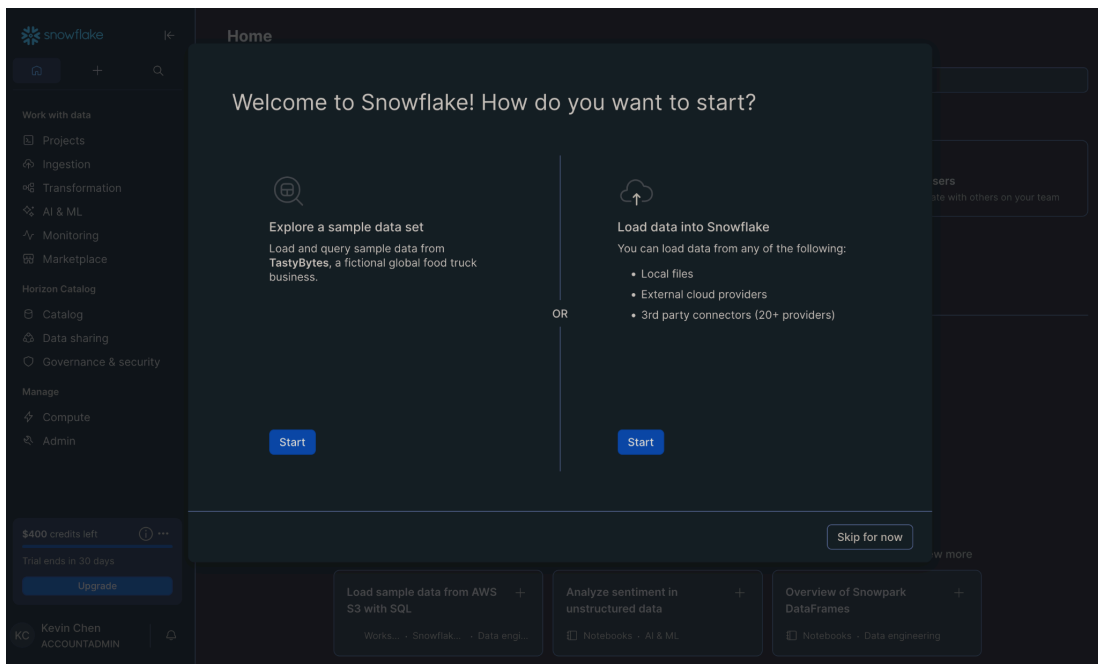
Your password must be 14 - 256 characters and contain at least 1 number(s), 0 special character(s), 1 uppercase and 1 lowercase letter(s).

Confirm password

Get started

日本語 Français

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



Home

Welcome to Snowflake! How do you want to start?

Explore a sample data set
Load and query sample data from TastyBytes, a fictional global food truck business.

Start

OR

Load data into Snowflake
You can load data from any of the following:

- Local files
- External cloud providers
- 3rd party connectors (20+ providers)

Start

Skip for now

Work with data

- Projects
- Ingestion
- Transformation
- AI & ML
- Monitoring
- Marketplace

Horizon Catalog

- Catalog
- Data sharing
- Governance & security

Manage

- Compute
- Admin

\$400 credits left ⓘ ...
Trial ends in 30 days
Upgrade

Kevin Chen
ACCOUNTADMIN

Load sample data from AWS S3 with SQL +
Works... · Snowflake... · Data engi...

Analyze sentiment in unstructured data +
Notebooks · AI & ML

Overview of Snowpark DataFrames +
Notebooks · Data engineering

See more

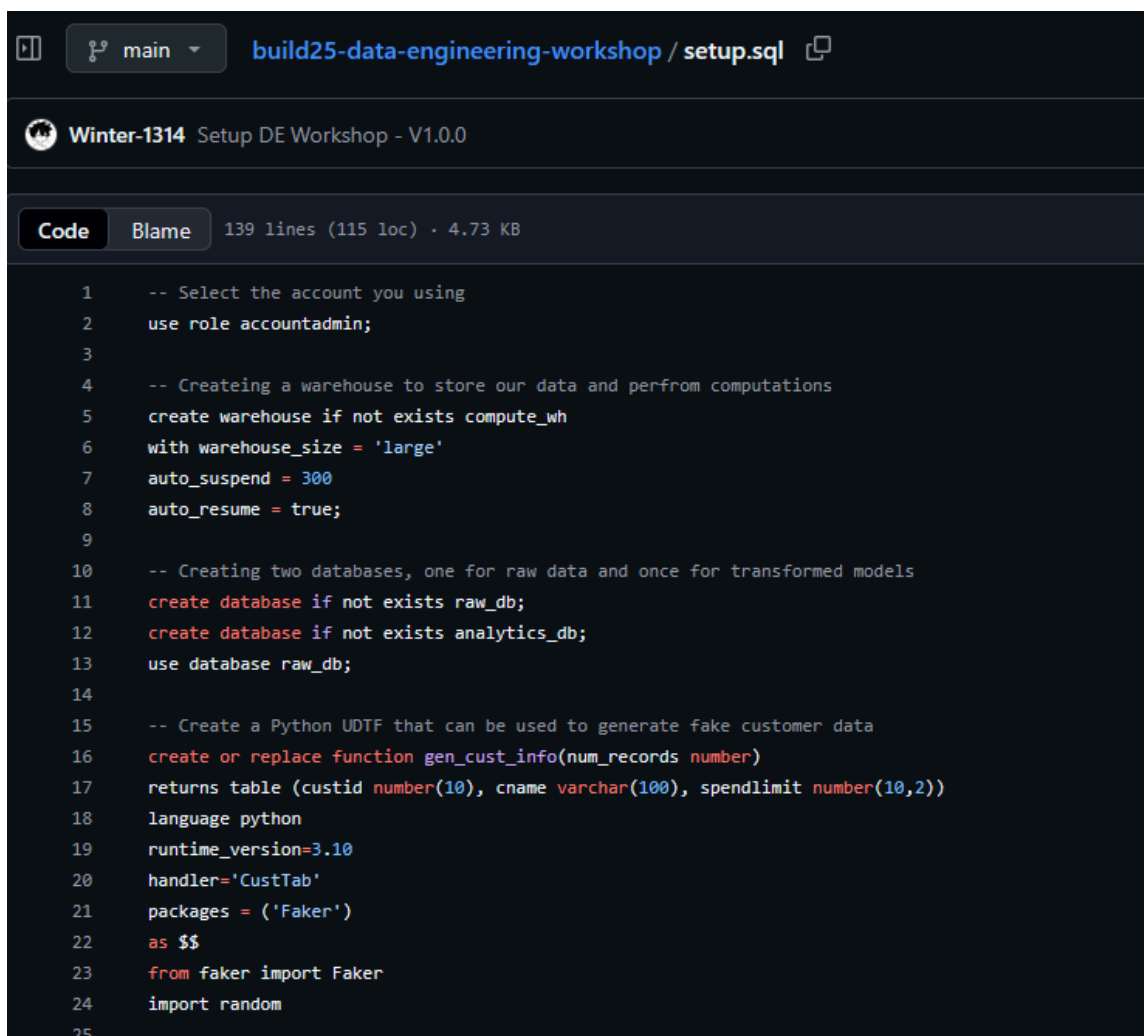
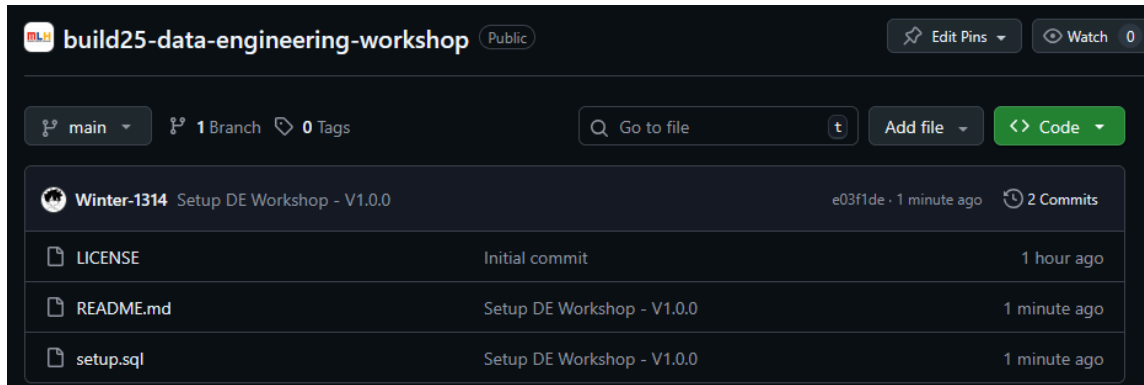
Loading in the Data

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

Setting Up Account

Showcase `setup.sql`

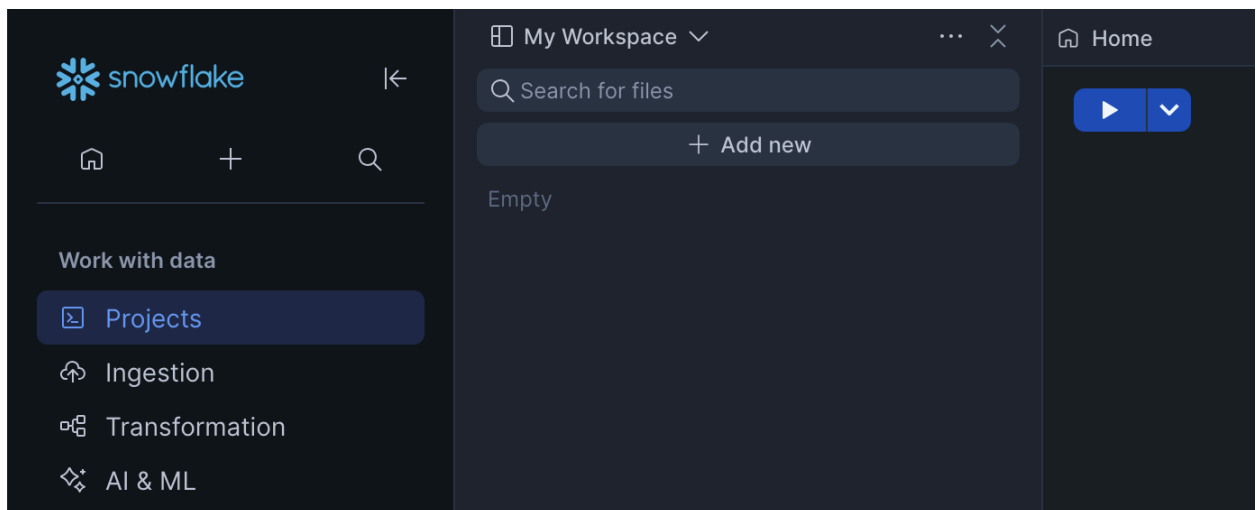
1. Show the attendees the `setup.sql` script



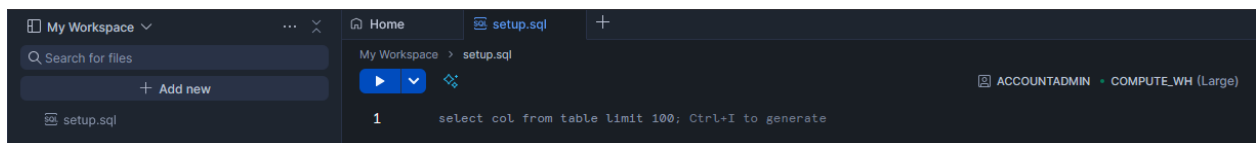
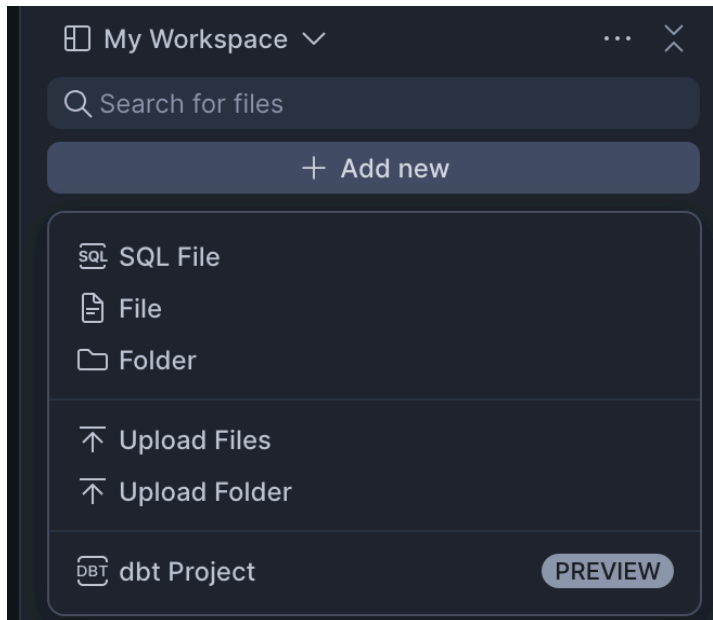
2. Scroll through and explain what the script is doing.
 - a. 1: Switch to the ACCOUNTADMIN role
 - b. 5-8: Create the Warehouse that will be used for compute/storage
 - c. 10-13: Create a database called RAW_DB for raw data and one called ANALYTICS_DB for transformed models
 - d. 16-122: Create three 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

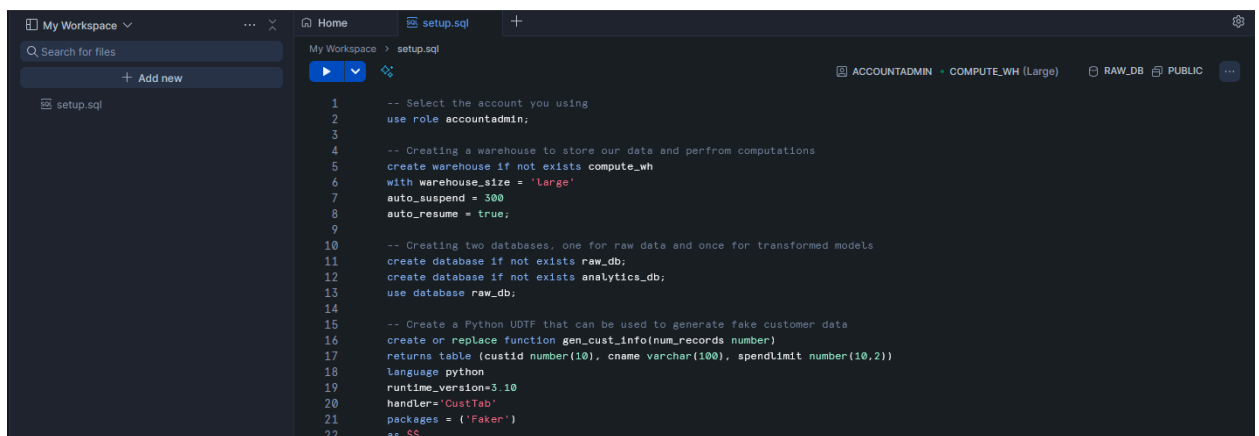
1. Go to "Projects" which should take you to "My Workspace"



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

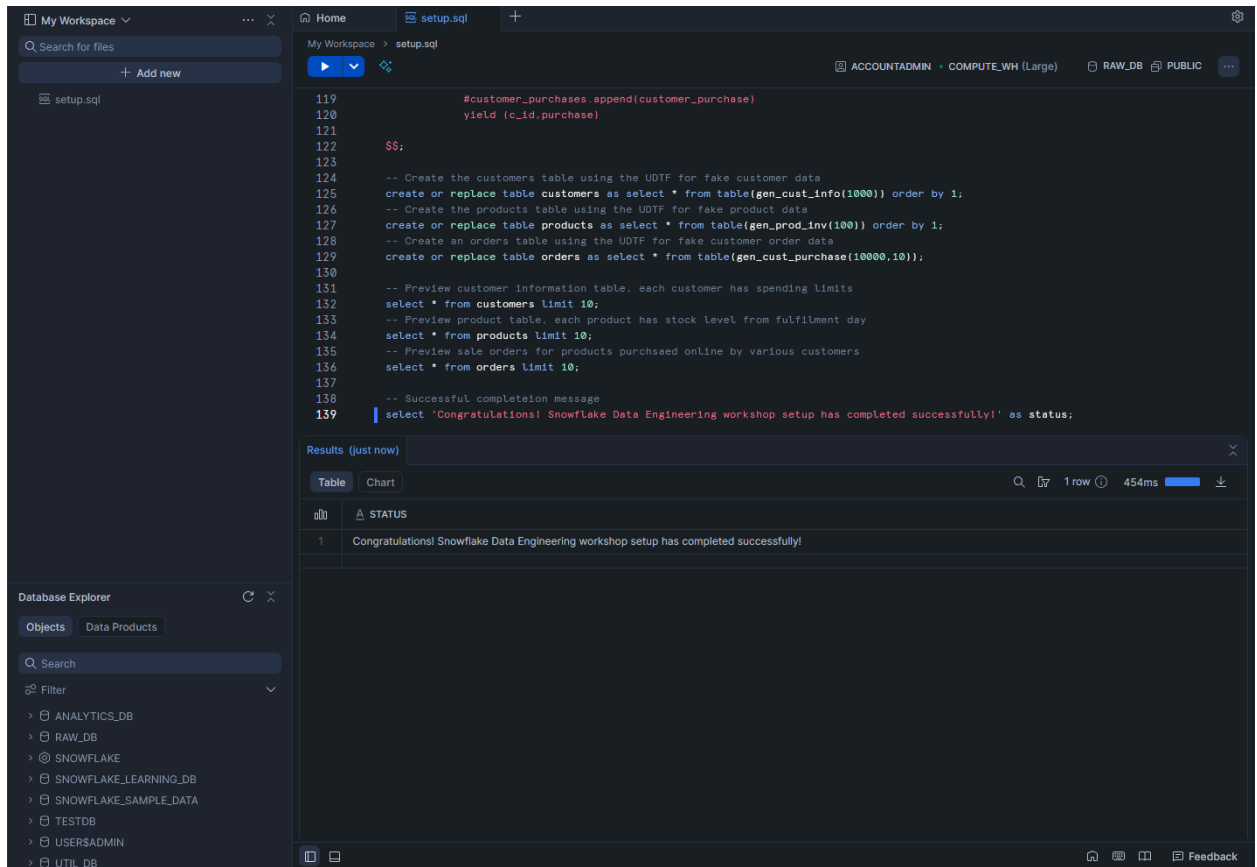


- Navigate back to the GitHub repo, open `setup.sql`, copy its contents, and paste them into the SQL file you created in your workspace.



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

setup has completed successfully!"

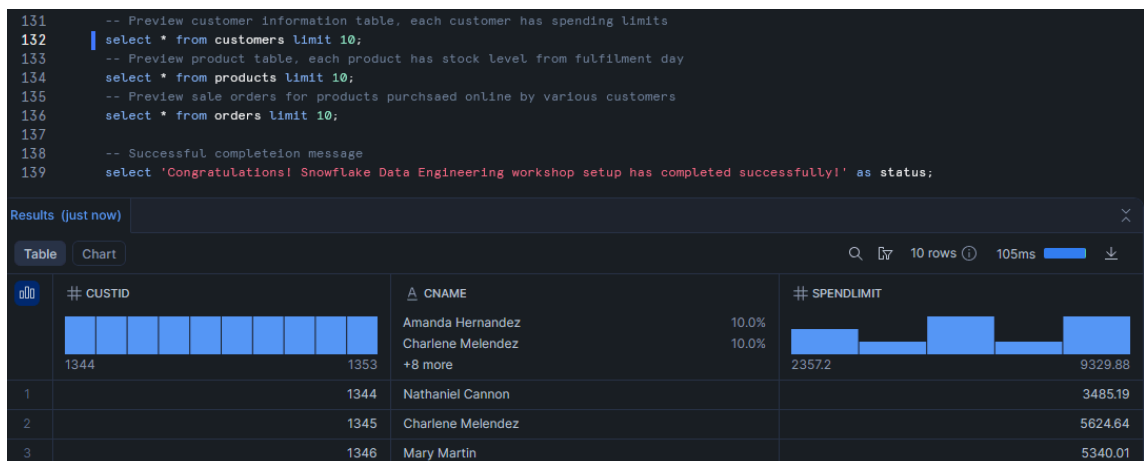


```

119      #customer_purchases.append(customer_purchase)
120      yield (c_id,purchase)
121
122  $$;
123
124  -- Create the customers table using the UDTF for fake customer data
125  create or replace table customers as select * from table(gen_cust_info(1000)) order by 1;
126  -- Create the products table using the UDTF for fake product data
127  create or replace table products as select * from table(gen_prod_inv(100)) order by 1;
128  -- Create an orders table using the UDTF for fake customer order data
129  create or replace table orders as select * from table(gen_cust_purchase(10000,10));
130
131  -- Preview customer information table, each customer has spending limits
132  select * from customers limit 10;
133  -- Preview product table, each product has stock level from fulfilment day
134  select * from products limit 10;
135  -- Preview sale orders for products purchased online by various customers
136  select * from orders limit 10;
137
138  -- Successful completion message
139  select 'Congratulations! Snowflake Data Engineering workshop setup has completed successfully!' as status;
  
```

| RESULTS (just now) | |
|--------------------|--|
| Table | Chart |
| 1 row 454ms | |
| STATUS | |
| 1 | Congratulations! Snowflake Data Engineering workshop setup has completed successfully! |

- 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



```

131  -- Preview customer information table, each customer has spending limits
132  select * from customers limit 10;
133  -- Preview product table, each product has stock level from fulfilment day
134  select * from products limit 10;
135  -- Preview sale orders for products purchased online by various customers
136  select * from orders limit 10;
137
138  -- Successful completion message
139  select 'Congratulations! Snowflake Data Engineering workshop setup has completed successfully!' as status;
  
```

| RESULTS (just now) | | | |
|--------------------|------------------------|--------------|---------|
| Table | Chart | | |
| 10 rows 105ms | | | |
| # CUSTID | A CNAME | # SPENDLIMIT | |
| 1344 | Amanda Hernandez | 10.0% | 2357.2 |
| 1353 | Charlene Melendez | 10.0% | 9329.88 |
| | +8 more | | |
| 1 | 1344 Nathaniel Cannon | | 3485.19 |
| 2 | 1345 Charlene Melendez | | 5624.64 |
| 3 | 1346 Mary Martin | | 5340.01 |

```

131 -- Preview customer information table, each customer has spending limits
132 select * from customers limit 10;
133 -- Preview product table, each product has stock level from fulfilment day
134 select * from products limit 10;
135 -- Preview sale orders for products purchased online by various customers
136 select * from orders limit 10;
137
138 -- Successful completion message
139 select 'Congratulations! Snowflake Data Engineering workshop setup has completed successfully!' as status;

```

Results (just now)

Table Chart 10 rows 68ms

| # | PID | PNAME | # STOCK | STOCKDATE |
|---|-----|---|---------|------------|
| | 114 | Customizable actuating framework Extended fault-tolerant encoding +8 more | 502 | 8/14/2025 |
| 1 | 119 | Profit-focused radical encoding | 770.00 | 2025-09-16 |
| 2 | 124 | Reactive radical forecast | 660.00 | 2025-10-02 |
| 3 | 125 | Re-contextualized contextually-based budgetary manager | 887.00 | 2025-10-22 |

```

131 -- Preview customer information table, each customer has spending limits
132 select * from customers limit 10;
133 -- Preview product table, each product has stock level from fulfilment day
134 select * from products limit 10;
135 -- Preview sale orders for products purchased online by various customers
136 select * from orders limit 10;
137
138 -- Successful completion message
139 select 'Congratulations! Snowflake Data Engineering workshop setup has completed successfully!' as status;

```

Results (just now)

Table Chart 10 rows 51ms

| # | CUSTID | PURCHASE |
|---|--------|--|
| | 1190 | 100% filled |
| 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} |

- 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.

snowflake Database Explorer HORIZON CATALOG

Work with data: Projects, Ingestion, Transformation, AI & ML, Monitoring, Marketplace, Horizon Catalog, Catalog, Data sharing, Governance & security, Manage, Compute, Admin

Database Explorer: Search, Databases, ANALYTICS_DB, RAW_DB, INFORMATION_SCHEMA, PUBLIC, Tables, CUSTOMERS, ORDERS, PRODUCTS, Functions, GEN_CUST_INFO(NUMBER), GEN_CUST_PURCHASE(NUMBER, N...), GEN_PROD_INV(NUMBER), SNOWFLAKE, SNOWFLAKE_LEARNING_DB, SNOWFLAKE_SAMPLE_DATA, TESTDB, USERADMIN, UTIL_DB

RAW_DB / PUBLIC / CUSTOMERS

Table Details Columns Data Preview Copy History

DASH_WH_SI 100 of 1K Rows • Updated just now

| | CUSTID | CNAME | SPENDLIMIT |
|----|--------|-------------------|------------|
| 2 | 1313 | Anthony Webb | 5375.62 |
| 3 | 1314 | Karen Bowman | 5859.37 |
| 4 | 1315 | Michelle Evans | 6588.51 |
| 5 | 1316 | Kevin Cain | 3467.97 |
| 6 | 1317 | Jennifer Jones | 8023.87 |
| 7 | 1318 | Louis Warren | 1789.20 |
| 8 | 1319 | Stephen Jordan | 8805.75 |
| 9 | 1320 | James Carroll | 8878.74 |
| 10 | 1321 | Brian Castro | 4940.93 |
| 11 | 1322 | Amber George | 9199.26 |
| 12 | 1323 | David Jones | 4030.52 |
| 13 | 1324 | Ricky Watson | 3563.99 |
| 14 | 1325 | Gary Aguilar | 1848.84 |
| 15 | 1326 | Katelyn Hood | 1967.68 |
| 16 | 1327 | Alexander Swanson | 9632.73 |
| 17 | 1328 | Marcus Glenn | 7380.01 |
| 18 | 1329 | Sarah Daniels | 3332.44 |
| 19 | 1330 | Jacob Willis | 3253.47 |
| 20 | 1331 | Patrick Glenn | 2479.21 |
| 21 | 1332 | David Torres | 5284.96 |
| 22 | 1333 | Joseph Gilbert | 3343.08 |
| 23 | 1334 | Nicole King | 9479.56 |
| 24 | 1335 | Maureen Watson | 7633.52 |
| 25 | 1336 | Belinda Nelson | 9206.19 |

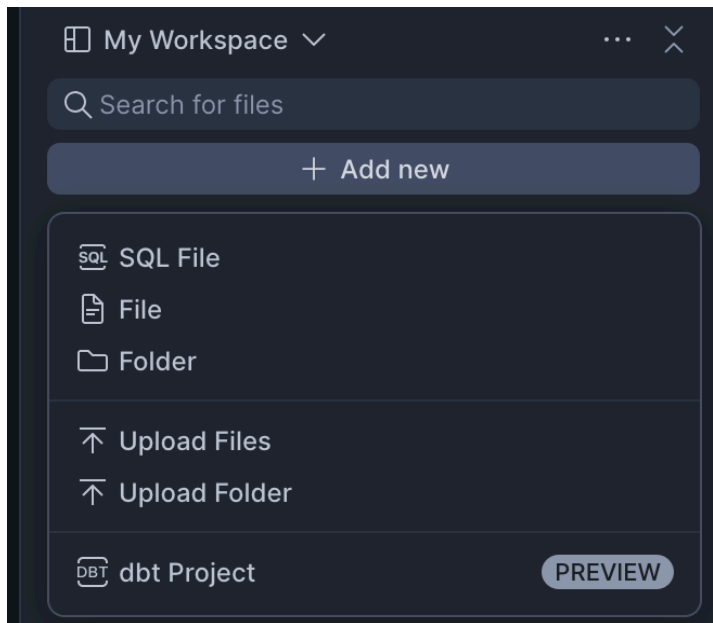
\$277 credits left Trial ends in 16 days Upgrade

Kevin Chen SNOWFLAKE_INTE...

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;
```

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 |

c.



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;
```

Results (just now)

Table Chart

2 rows 39ms

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

C.

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;
```

Results (just now)

Table Chart

| | status |
|---|---|
| 1 | Dynamic table STG_ORDERS_DT successfully created. |

k.

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;
```

Results (just now)

Table Chart 1,000 rows 490ms

| | # CUSTOMER_ID | CUSTOMER_NAME | # SPEND_LIMIT |
|---|---------------|--|---------------|
| | 1001 | Jessica Scott Luis Wilson +98 more | 0.2% 0.2% |
| 1 | 1985 | Samuel Alvarez | 3553.61 |
| 2 | 1986 | Russell Collins | 8900.24 |
| 3 | 1987 | Craig Rush | 1569.95 |

c.

Results (just now)

Table Chart 10,000 rows 657ms

| | # CUSTOMER_ID | # PRODUCT_ID | # ORDER_PRICE | # QUANTITY | ORDER_DATE |
|---|---------------|--------------|---------------|------------|------------|
| 1 | 1803 | 143 | 833.46 | 2 | 2025-10-23 |
| 2 | 1953 | 154 | 121.73 | 5 | 2025-10-17 |
| 3 | 1630 | 168 | 951.73 | 3 | 2025-10-25 |

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.

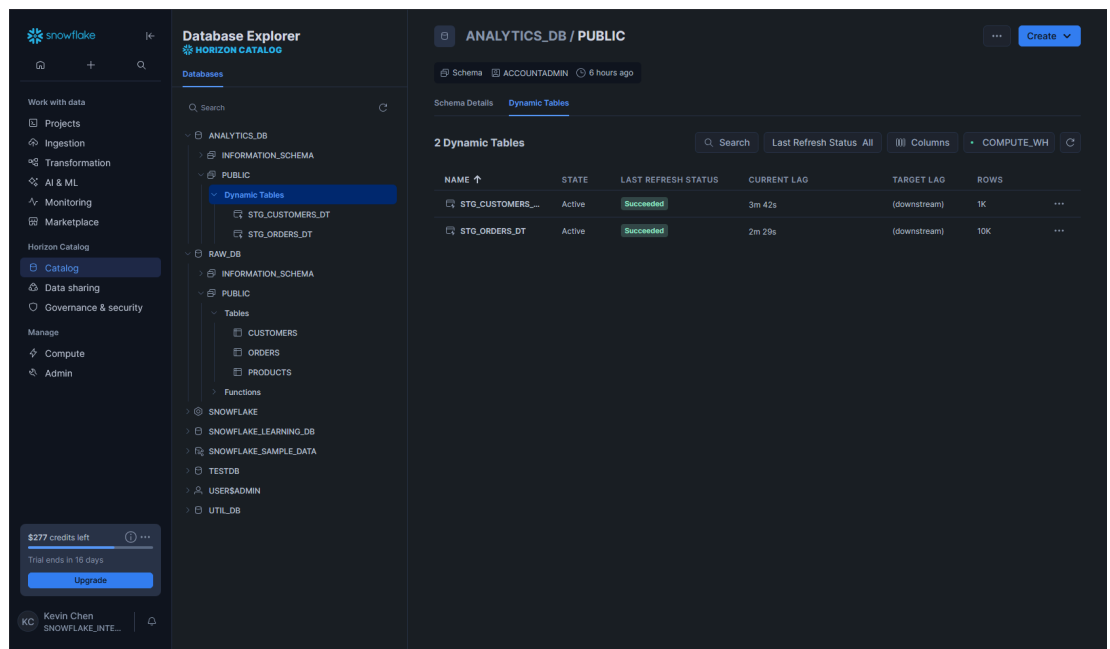
a. `show dynamic tables;`

Results (just now)

Table Chart 2 rows 101ms

| | created_on | name | database_name | schema_name | cluster_by | # rows | # bytes | owner |
|---|-------------------------------|------------------|---------------|-------------|------------|--------|---------|--------------|
| 1 | 2025-10-27 04:08:07.727 -0700 | STG_CUSTOMERS_DT | ANALYTICS_DB | PUBLIC | | 1000 | 72704 | ACCOUNTADMIN |
| 2 | 2025-10-27 04:09:21.225 -0700 | STG_ORDERS_DT | ANALYTICS_DB | PUBLIC | | 10000 | 410624 | ACCOUNTADMIN |

b.



Database Explorer

ANALYTICS_DB / PUBLIC

Schema Details: **Dynamic Tables**

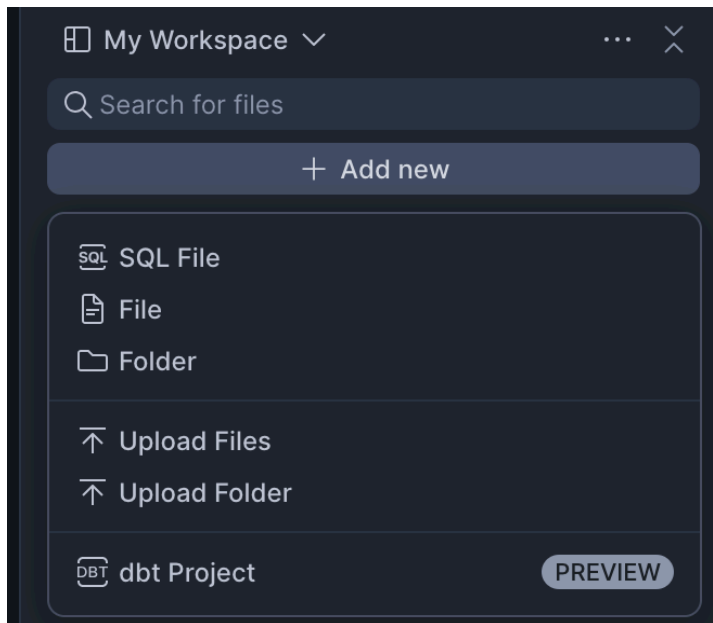
2 Dynamic Tables

| NAME | STATE | LAST REFRESH STATUS | CURRENT LAG | TARGET LAG | ROWS |
|------------------|--------|---------------------|-------------|--------------|------|
| STG_CUSTOMERS_DT | Active | Succeeded | 3m 42s | (downstream) | 1K |
| STG_ORDERS_DT | Active | Succeeded | 2m 29s | (downstream) | 10K |

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 so 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;
```

Results (just now) 10,001 rows 679ms

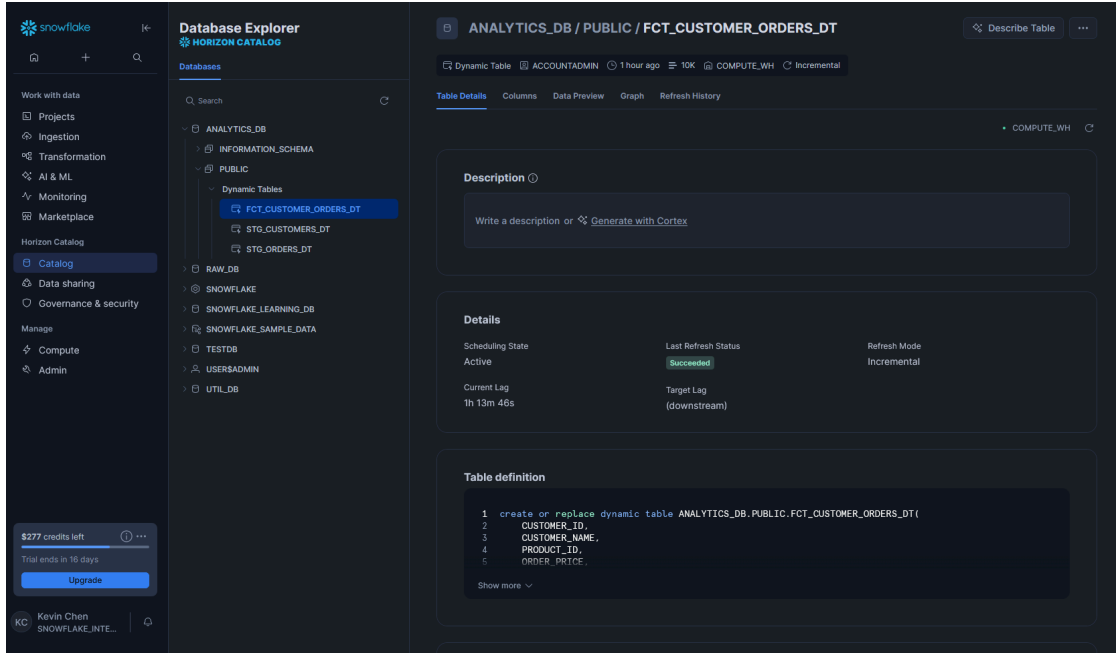
Table Chart

| | # CUSTOMER_ID | A 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 |

b.

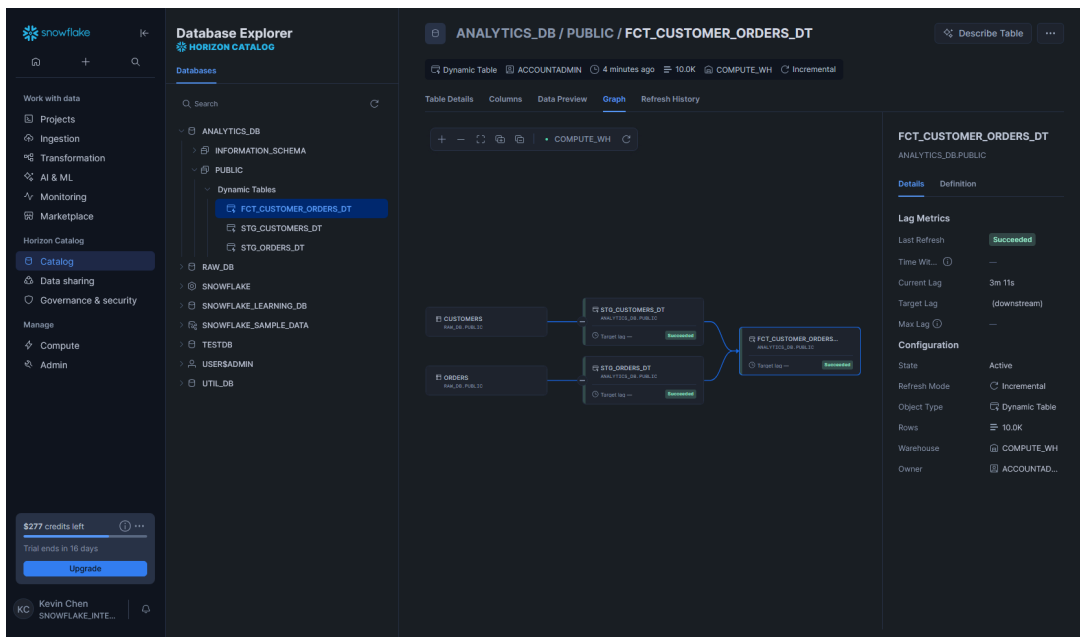
Visualize the Pipeline

1. Navigate to "Catalog" and under "Dynamic Table" select FCT_CUSTOMER_ORDER_DT.



The screenshot shows the Snowflake Database Explorer interface. On the left, the 'Catalog' tab is selected in the sidebar. The main pane displays the 'Dynamic Tables' section under the 'PUBLIC' schema. The table 'FCT_CUSTOMER_ORDERS_DT' is highlighted. The right pane shows the 'Table Details' for 'ANALYTICS_DB / PUBLIC / FCT_CUSTOMER_ORDERS_DT'. It includes a description field, a 'Details' section with scheduling state (Active), last refresh status (Succeeded), and refresh mode (Incremental). The 'Table definition' section shows the SQL code for creating the dynamic table.

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.

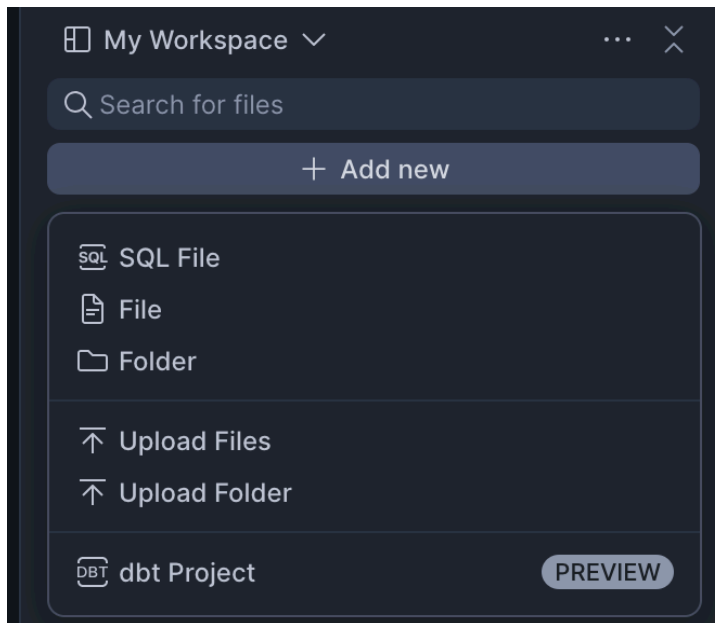


The screenshot shows the Snowflake Database Explorer interface with the 'Graph' tab selected. The main pane displays a visual DAG (Directed Acyclic Graph) of the pipeline. The graph shows the flow from raw tables (CUSTOMERS, ORDERS) to staging dynamic tables (STG_CUSTOMERS_DT, STG_ORDERS_DT) and finally to the fact dynamic table (FCT_CUSTOMER_ORDERS_DT). The right pane shows the 'FCT_CUSTOMER_ORDERS_DT' details, including lag metrics and configuration.

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.

```
a. 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.

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

Results (just now)

TableChart

75 rows361ms

| | NAME | SCHEMA_NAME | DATABASE_NAME | STATE | STATE_CODE | STATE_MESSAGE | QUERY_ID |
|---|------------------------|---------------|--------------------|-----------------|---------------|---------------------|--------------------------|
| | STG_ORDERS_DT 68.0% | | ANALYTICS... 98.7% | | | All values are null | 01bffc15-0000-4214-00... |
| | STG_CUSTOMERS_DT 20.0% | | RAW_DB 1.3% | | | | 01bffc23-0000-4214-00... |
| | +1 more | PUBLIC 100.0% | | SUCCE... 100.0% | SUCCESS100.0% | | +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.

```
a. 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.

```
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
```

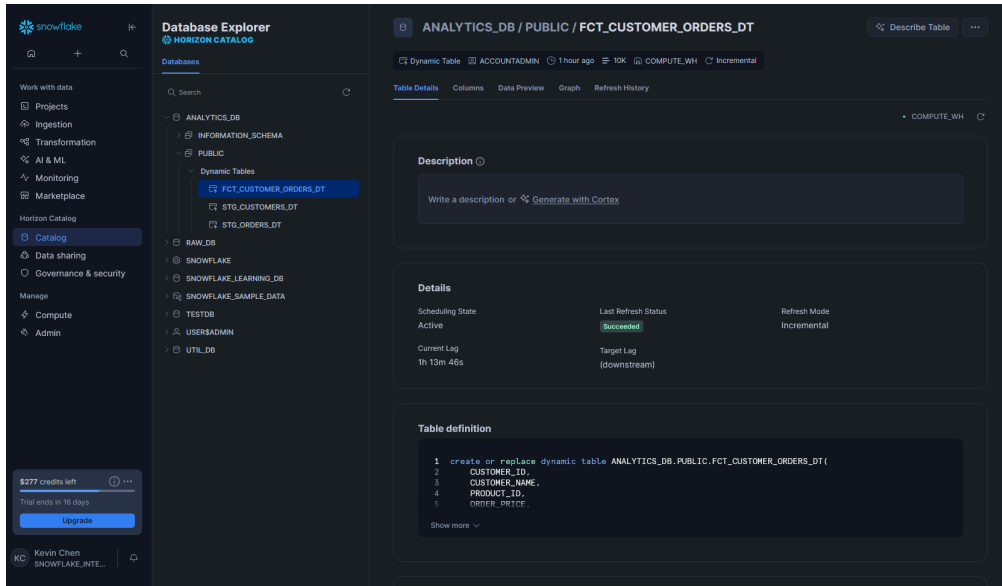
```
n.      where o.product_id is not null;
```

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

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

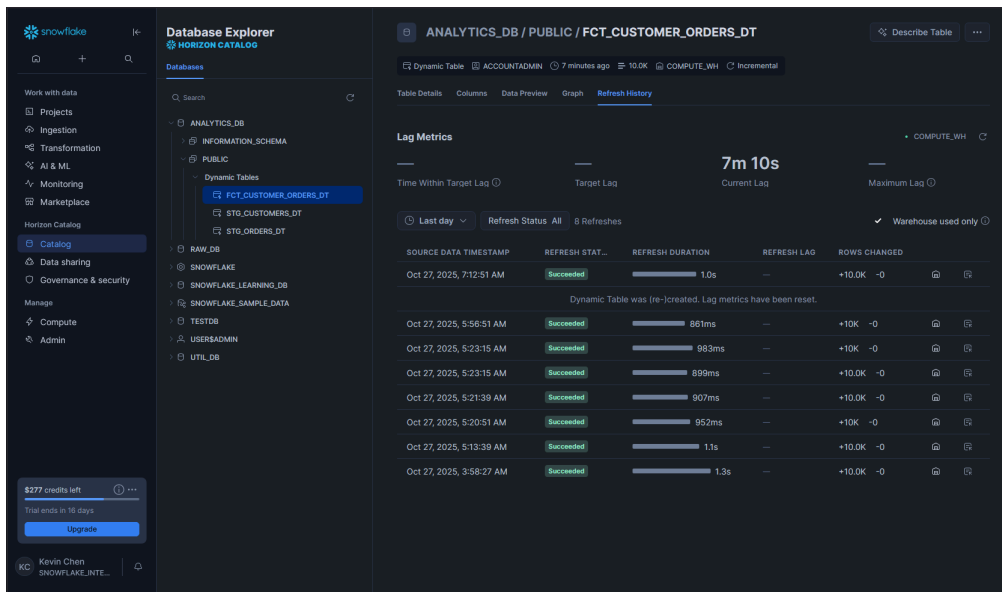
Pipeline Management in Snowsight

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



The screenshot shows the Snowflake Database Explorer interface. On the left, the 'Catalog' tab is selected in the sidebar. In the main pane, the 'Dynamic Tables' section is expanded, and 'FCT_CUSTOMER_ORDERS_DT' is selected. The right pane displays the table's details, including its description, scheduling state (Active), last refresh status (Succeeded), and the table definition SQL code.

- 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.

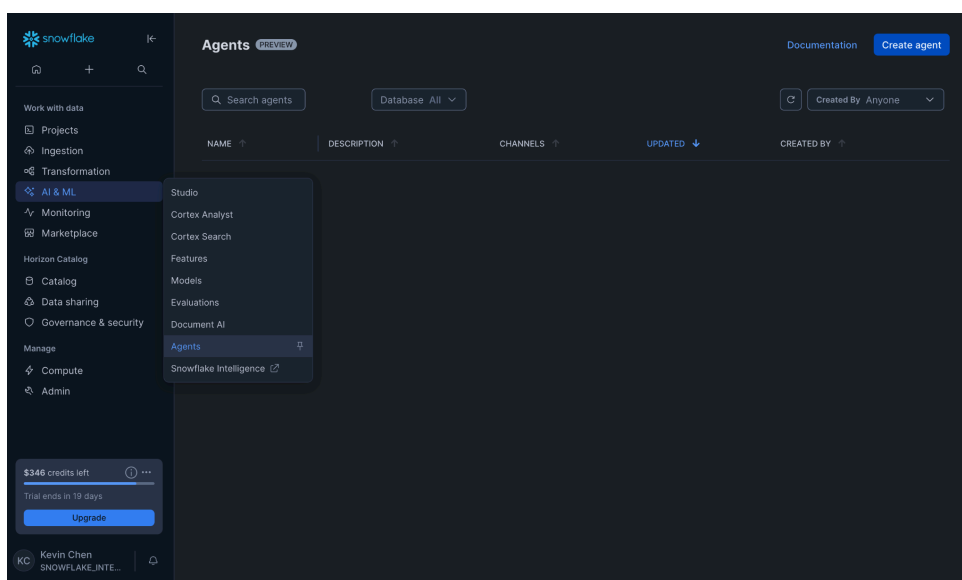


The screenshot shows the 'Refresh History' tab selected for the dynamic table 'FCT_CUSTOMER_ORDERS_DT'. The 'Lag Metrics' section displays a current lag of 7m 10s. Below, a table lists the refresh history with columns for Source Data Timestamp, Refresh Status, Refresh Duration, Refresh Lag, and Rows Changed.

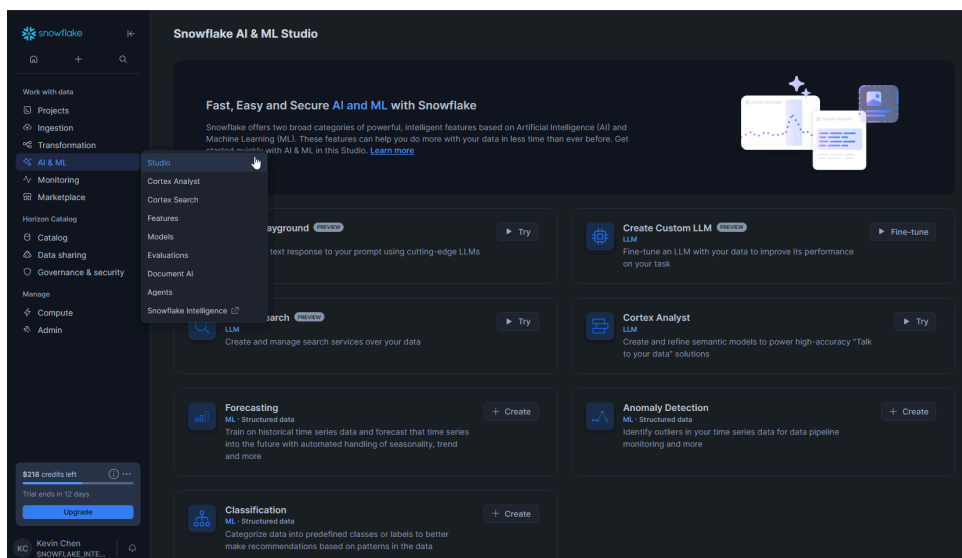
| SOURCE DATA TIMESTAMP | REFRESH STAT... | REFRESH DURATION | REFRESH LAG | ROWS CHANGED |
|--|-----------------|------------------|-------------|--------------|
| Oct 27, 2025, 7:12:51 AM | Succeeded | 1.0s | — | +10.0K -0 |
| Dynamic Table was (re-)created. Lag metrics have been reset. | | | | |
| Oct 27, 2025, 5:56:51 AM | Succeeded | 861ms | — | +10K -0 |
| Oct 27, 2025, 5:23:15 AM | Succeeded | 983ms | — | +10K -0 |
| Oct 27, 2025, 5:23:15 AM | Succeeded | 899ms | — | +10.0K -0 |
| Oct 27, 2025, 5:21:39 AM | Succeeded | 907ms | — | +10.0K -0 |
| Oct 27, 2025, 5:20:51 AM | Succeeded | 952ms | — | +10K -0 |
| Oct 27, 2025, 5:13:39 AM | Succeeded | 1.1s | — | +10.0K -0 |
| Oct 27, 2025, 3:58:27 AM | Succeeded | 1.3s | — | +10.0K -0 |

Module 5: Snowflake Intelligence

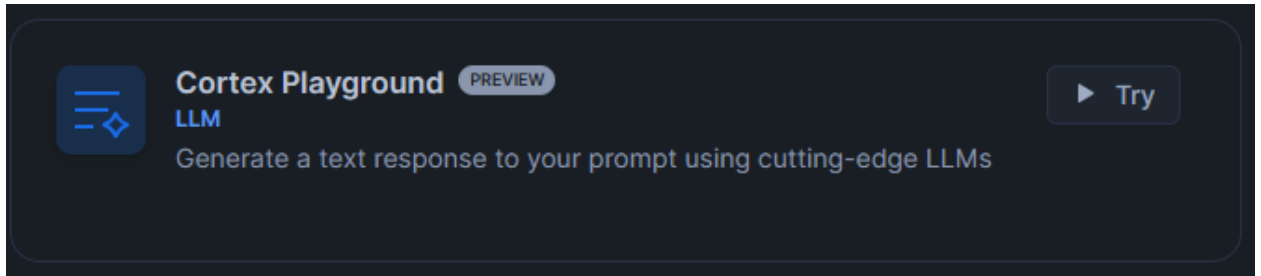
1. Resources to explore Snowflake Intelligence
 - a. [Snowflake Intelligence Overview](#)
 - b. [Snowflake Cortex Agents](#)
 - c. [Snowflake Cortex Analyst](#)
 - d. [Snowflake Cortex Search](#)
 - e. [Understanding Snowflake Cortex](#)
 - f. [Tutorial: Getting Started with Snowflake Intelligence](#)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



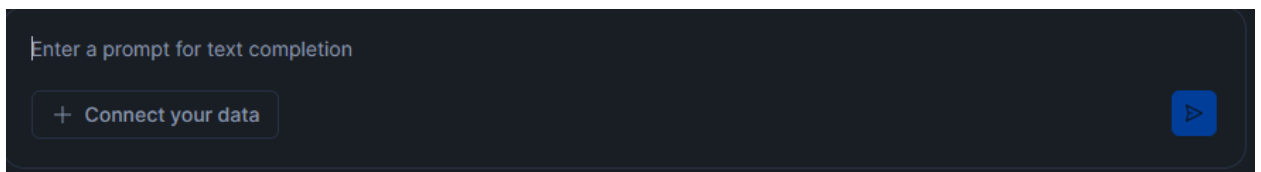
3. Navigate to “Studio” where you can see the “Snowflake AI & ML Studio”



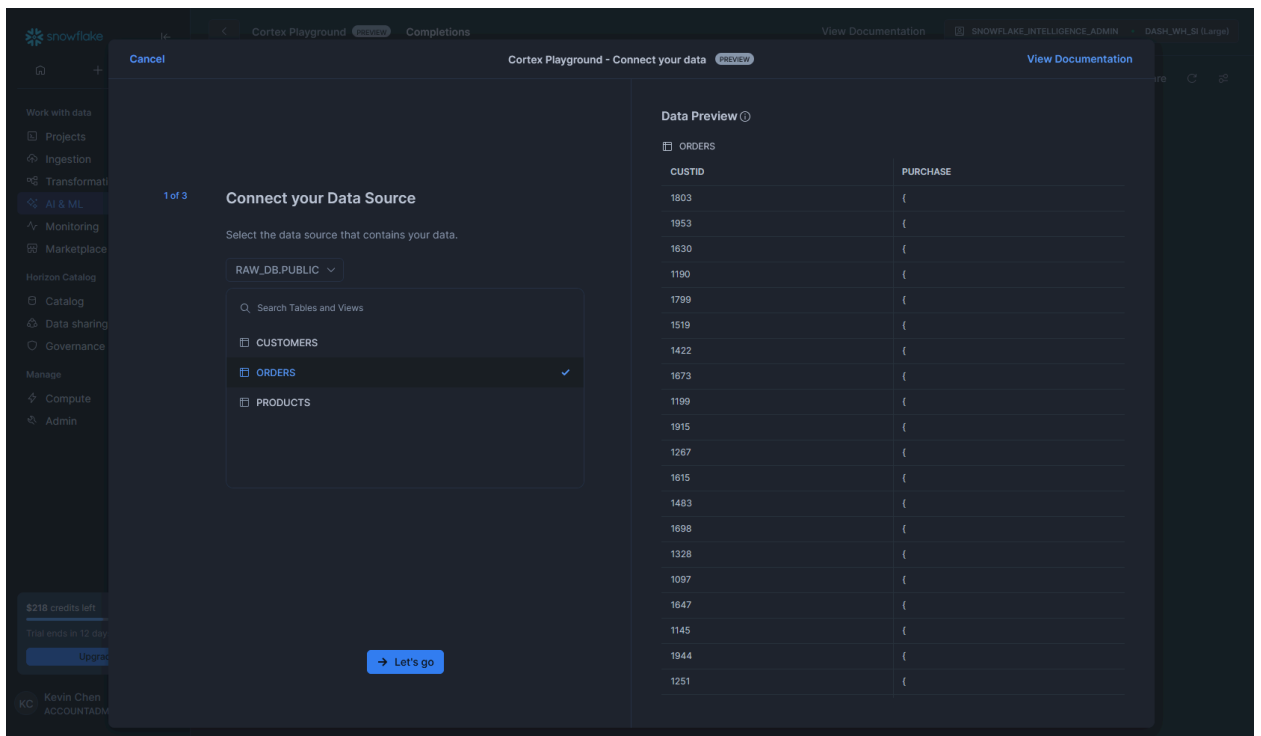
4. Select “Try” for Cortex Playground. Note that we will be testing using the Cortex Playground, which does not directly support Dynamic Tables. To use Dynamic Tables, you would need to leverage Cortex Search and create a custom Agent, which attendees can explore further if they’re interested.



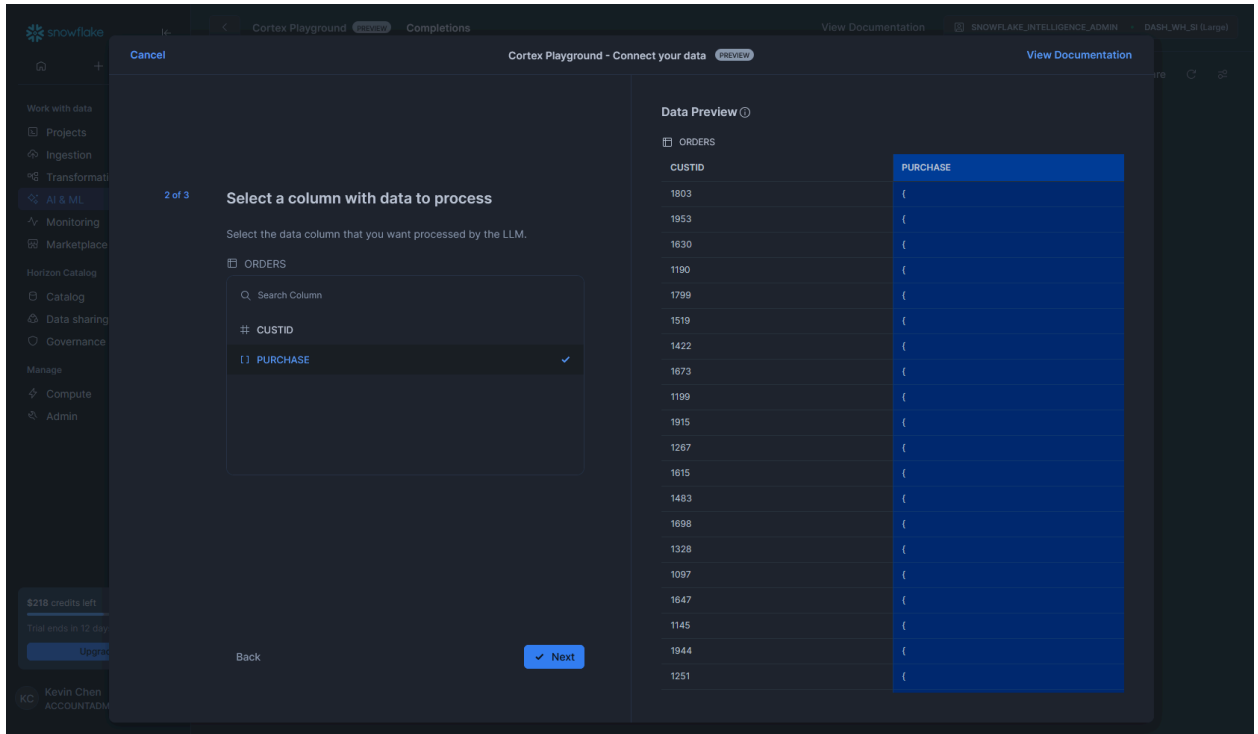
5. Select “Connect your data”



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



7. Select "PURCHASE" then click "Next"



2 of 3 Select a column with data to process

Select the data column that you want processed by the LLM.

ORDERS

Search Column

CUSTID

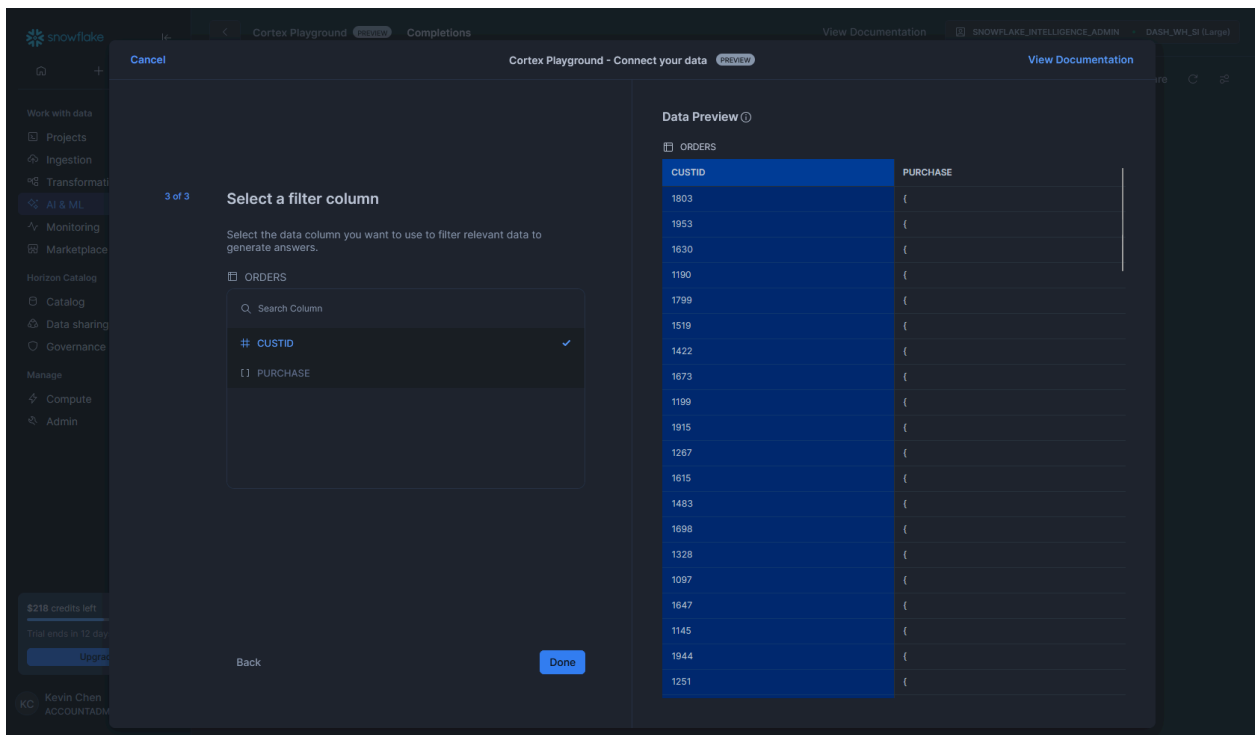
[X] PURCHASE ✓

Back Next

Data Preview

| ORDERS | PURCHASE |
|--------|----------|
| 1803 | { |
| 1953 | { |
| 1630 | { |
| 1190 | { |
| 1799 | { |
| 1519 | { |
| 1422 | { |
| 1673 | { |
| 1199 | { |
| 1915 | { |
| 1267 | { |
| 1615 | { |
| 1483 | { |
| 1698 | { |
| 1328 | { |
| 1097 | { |
| 1647 | { |
| 1145 | { |
| 1944 | { |
| 1251 | { |

8. Select "CUSTID" then click "Done"



3 of 3 Select a filter column

Select the data column you want to use to filter relevant data to generate answers.

ORDERS

Search Column

CUSTID ✓

[X] PURCHASE

Back Done

Data Preview

| ORDERS | PURCHASE |
|--------|----------|
| 1803 | { |
| 1953 | { |
| 1630 | { |
| 1190 | { |
| 1799 | { |
| 1519 | { |
| 1422 | { |
| 1673 | { |
| 1199 | { |
| 1915 | { |
| 1267 | { |
| 1615 | { |
| 1483 | { |
| 1698 | { |
| 1328 | { |
| 1097 | { |
| 1647 | { |
| 1145 | { |
| 1944 | { |
| 1251 | { |

- System Prompt

Enter additional context for processing the user prompt

ORDERS

{ "prodid": 149, "purchase_amount": 7.350800000000000e+02, "purchase_date": "2...

- ⌵

claude-3-5-sonnet

⌵

ORDERS · Row 9

System

What is this purchase about?

User

{
 "prodid": 149,
 "purchase_amount": 7.350800000000000e+02,...

Show all

⌵

claude-3-5-sonnet

This appears to be a purchase record showing:
 - Product ID: 149
 - Total purchase amount: \$735.08
 - Purchase date: October 21, 2025
 - Quantity purchased: 2 unitsThe customer bought 2 units of product #149 for a total of \$735.08 (which means each unit cost \$367.54) with the purchase scheduled for October 21, 2025.

📄

View Code

System Prompt

Enter additional context for processing the user prompt

ORDERS

×

{ "prodid": 149, "purchase_amount": 7.350800000000000e+02, "purchase_date": "2025-10-21" }

×

✓

▶

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