



Hands-on Lab



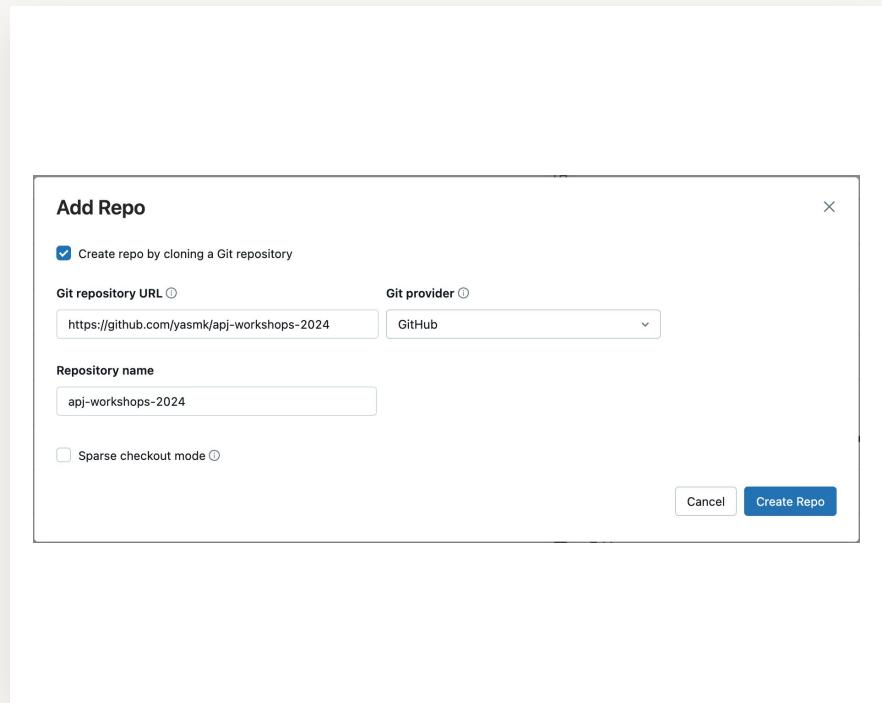
Data Intelligence Day, 2024

Replicate the Code

1. Click on **New** then **Repo**
2. Enter the following **Git URL**

<https://github.com/yasmk/apj-workshops-2024>

3. Click **Create Repo**



Create Datasets

1. Find your cloned repository under **Workspace > Repo**
2. Open the “01 Setup” notebook under the “Lab 01 - SQL” folder
3. Click **Run all**
4. Give Databricks a couple of minutes to create all required datasets

```
01 Setup  main Python  ⭐  File Edit View Run Help Last edit was 13 hours ago New cell UI: ON ▶ Run all 🌟 Marat Levit's → Clu... Schedule Share Workspace Lab 01 - SQL Sort: Name 01 Setup 02 Explore.sql 03 Views.sql 04 Filters and Parameters.sql 05 Alerts.sql 06 JSON.sql 07 UDFs.sql 08 System Tables.sql

import os
reset = True
current_user_id = (
    dbutils.notebook.entry_point.getDbutils().notebook().getContext().userName().get()
)
datasets_location = f"/FileStore/tmp/{current_user_id}/datasets/"
catalog = "workshop"
database_name = current_user_id.split("@")[-1].replace(".", "_")

# if reset:
#     dbutils.fs.rm(datasets_location, True)
#     spark.sql(f"DROP DATABASE IF EXISTS {database_name} CASCADE")
#     spark.sql(f"DROP CATALOG IF EXISTS {catalog} CASCADE")

# Create catalog
spark.sql(f"CREATE CATALOG IF NOT EXISTS {catalog};")
spark.sql(f"GRANT USE CATALOG ON CATALOG {catalog} TO '{current_user_id}'")
spark.sql(f"GRANT CREATE SCHEMA ON CATALOG {catalog} TO '{current_user_id}'")
spark.sql(f"USE CATALOG {catalog};")

# Create database
spark.sql(f"CREATE DATABASE IF NOT EXISTS {database_name};")
spark.sql(f"USE {database_name}")

DataFrame[]

▶ 13 hours ago (7s) 2
working_dir = f"/{os.getcwd().split('/')[-1]}:{0:5}"
git_datasets_location = f"{working_dir}/Datasets/SQL_Lab"

sample_datasets = [
    "dim_customer",
    "dim_locations",
    "dim_products",
    "fact_apj_sale_items",
    "fact_apj_sales",
]
for sample_data in sample_datasets:
    dbutils.fs.rm(f"{datasets_location}/SQL_Lab/{sample_data}.csv.gz")
    dbutils.fs.cp(
        f"file:{git_datasets_location}/{sample_data}.csv.gz",
        f"{datasets_location}/SQL_Lab/{sample_data}.csv.gz"
)
```





Data Exploration



Data Exploration

1. Open “02 Explore.sql”
2. Replace “catalog.database” with “workshop.<username>” and run the command
3. Run each command one at a time by highlighting it and hitting **Ctrl + Shift + Enter**

```
-- Set your catalog and database.
USE catalog_database;

-- Returns all the tables,
SHOW TABLES;

-- Select all the records from the table.
SELECT * FROM fact_api_sales;

-- Return information about schema, partitioning, table size, and so on.
DESCRIBE DETAIL fact_api_sales;

-- Returns provenance information, including the operation,
-- user, and so on, for each write to a table.
DESCRIBE HISTORY fact_api_sales;

-- Display detailed information about the specified columns,
-- including the column statistics collected by the command, and additional metadata information
DESCRIBE EXTENDED fact_api_sales;

-- Insert new records into the table
INSERT INTO
    fact_api_sales (customer_skey, slocation_skey, sale_id, ts, order_source, order_state,
    unique_customer_id, store_id)
VALUES
(
    "3157",
    "7",
    "00000000-0000-0000-0000-000000000000",
    current_timestamp(),
    "IN-STOCK",
    "COMPLETED",
    "SYD01-15",
    "SYD01"
),
(
    "3523",
    "5",
    "00041cc6-30f1-433d-97b5-b92191a92efb",
    current_timestamp(),
    "ONLINE"
)
```





Views



Views

1. Open “03 Views.sql”
2. Replace “catalog.database” with “workshop.<username>” and run the command
3. Run each command one at a time by highlighting it and hitting **Ctrl + Shift + Enter**

```
-- Set your catalog and database.
USE catalog_database;

-- Select all records from the fact table and join it
-- with the 'items' and 'locations' dimensions
SELECT
    sales.ts::timestamp as date,
    sales.store_id,
    sales.sale_id,
    items.product_cost::double as cost,
    locations.city
FROM
    fact_apj_sales sales
    JOIN fact_apj_sale_items items
        ON items.sale_id = sales.sale_id
    JOIN dim_locations locations
        ON sales.store_id = locations.id;

-- Create a reusable view of the above Select statement
CREATE VIEW IF NOT EXISTS vw_sales_cost_location AS
SELECT
    sales.ts::timestamp as date,
    sales.store_id,
    sales.sale_id,
    items.product_cost::double as cost,
    locations.city
FROM
    fact_apj_sales sales
    JOIN fact_apj_sale_items items
        ON items.sale_id = sales.sale_id
    JOIN dim_locations locations
        ON sales.store_id = locations.id;

-- Select all records from the view
SELECT * FROM vw_sales_cost_location;
```



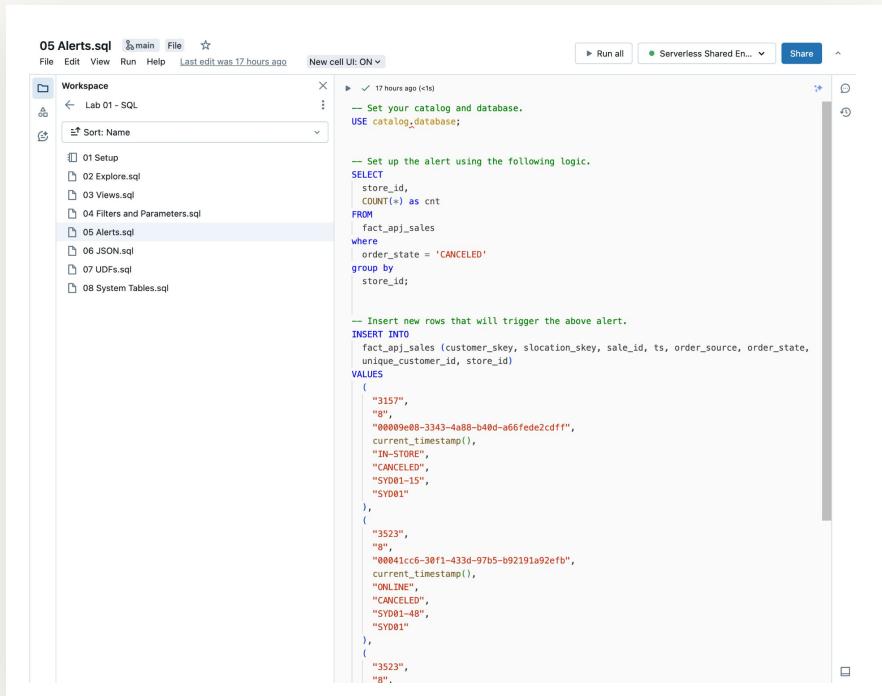


Alerts



Alerts

1. Open “05 Alerts.sql”
2. Replace “catalog.database” with “workshop.<username>” and run the command
3. Copy the “USE workshop...” and the following “SELECT” query to clipboard



The screenshot shows the Databricks SQL workspace interface. The left sidebar displays a list of files under 'Lab 01 - SQL': 01 Setup, 02 Explore.sql, 03 Views.sql, 04 Filters and Parameters.sql, 05 Alerts.sql (which is currently selected), 06 JSON.sql, 07 UDFs.sql, and 08 System Tables.sql. The main pane contains the contents of the '05 Alerts.sql' file:

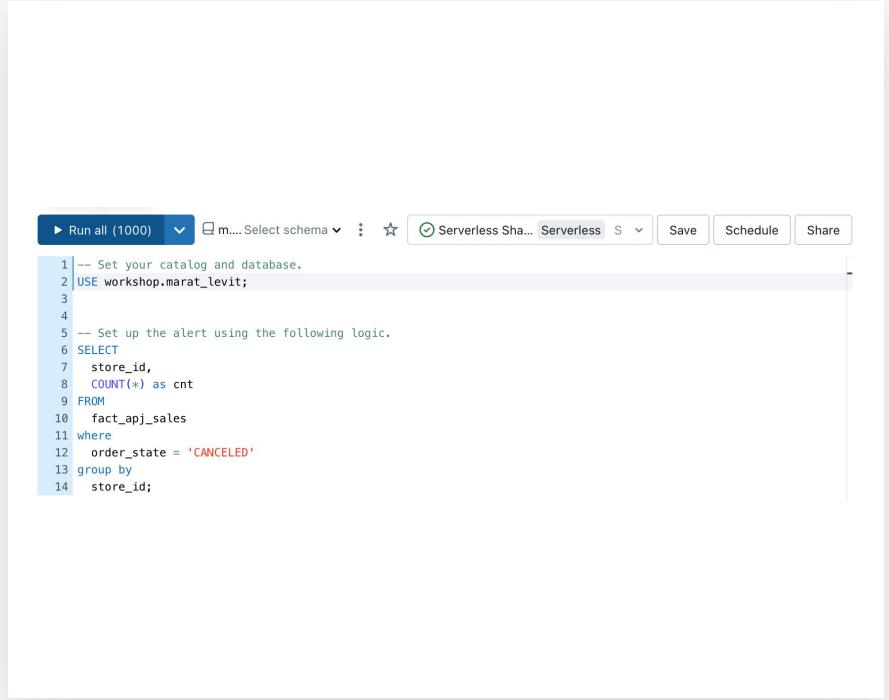
```
-- Set your catalog and database.  
USE catalog_database;  
  
-- Set up the alert using the following logic.  
SELECT  
    store_id,  
    COUNT(*) AS cnt  
FROM  
    fact_apj_sales  
WHERE  
    order_state = 'CANCELED'  
GROUP BY  
    store_id;  
  
-- Insert new rows that will trigger the above alert.  
INSERT INTO  
    fact_apj_sales (customer_skey, slocation_skey, sale_id, ts, order_source, order_state,  
    unique_customer_id, store_id)  
VALUES  
    ('3157',  
     '8',  
     '00009988-3343-4a88-b48d-a66fede2cdf',  
     current_timestamp(),  
     'ON-STORE',  
     'CANCELED',  
     'SY001-15',  
     'SY001'  
,  
    ('3523',  
     '8',  
     '00041cc6-30f1-433d-97b5-b92191a92efb',  
     current_timestamp(),  
     'ONLINE',  
     'CANCELED',  
     'SY01-48',  
     'SY01'  
,  
    ('3523',  
     '8',  
     '00041cc6-30f1-433d-97b5-b92191a92efb',  
     current_timestamp(),  
     'ONLINE',  
     'CANCELED',  
     'SY01-48',  
     'SY01'
```



Alerts

Create Query

1. Open **Queries** from the side panel
2. Click **Create query**. In the **SQL Editor** paste the SELECT query
3. Click **Save** and name the query "canceled_orders"



The screenshot shows a Databricks SQL editor interface. At the top, there are buttons for 'Run all (1000)', 'Select schema' (set to 'm....'), and various sharing and scheduling options. The code area contains a SELECT statement:

```
1 -- Set your catalog and database.
2 USE workshop.marat_levit;
3
4
5 -- Set up the alert using the following logic.
6 SELECT
7   store_id,
8   COUNT(*) as cnt
9 FROM
10  fact_apj_sales
11 WHERE
12  order_state = 'CANCELED'
13 GROUP BY
14  store_id;
```



Alerts

Create Alert

1. Open **Alerts** from the side panel
2. Click **Create alert**
3. Select the “canceled_orders” query from the dropdown
4. Change the trigger condition to **cnt SUM > 0**
5. Click **Create alert**

New alert

Alert name: canceled_orders: SUM(cnt) > 0

Query: canceled_orders

Trigger condition:

Value column: cnt	Sum	Operator: >	Threshold value: 0
-------------------	-----	-------------	--------------------

Preview alert

Notifications:

- When alert is triggered:
Send notification: Just once
- When alert returns back to normal:
 Send notification

Template: Use default template

Create alert



Alerts

Schedule & Subscribe

1. Click **Add schedule** after creating the Alert
2. Here you can adjust how often the alert query runs
3. Switch to the **Destinations** tab
4. Search for your username
5. Click **Create**

Add schedule

[Settings](#) Destinations

Schedule

Every 1 Hour at 37 minutes past the hour

Show cron syntax

Timezone

(UTC+00:00) UTC

[Settings](#) Destinations

Notify destinations when the alert is triggered

Search to add users or destinations

Destination

✉ marat.levit@databricks.com Delete

Collaborators may have schedules which are not visible to you. Consult with them to avoid creating duplicate schedules.

Cancel Create





JSON



JSON

1. Open “06 JSON.sql”
2. Replace “catalog.database” with “workshop.<username>” and run the command
3. Run each command one at a time by highlighting it and hitting **Ctrl + Shift + Enter**

```
-- Set your catalog and database.
USE catalog_database;

-- Select all data from the table
SELECT * FROM store_data_json;

-- Extract a top-level column
SELECT raw:owner FROM store_data_json;

-- Extract nested fields
SELECT raw:store.bicycle FROM store_data_json;

-- Escape characters
SELECT raw:owner, raw:'fb:TestId', raw:'zip code' FROM store_data_json;

-- Extract values from arrays
SELECT raw:store.fruit[0], raw:store.fruit[1] FROM store_data_json;

-- Extract subfields from arrays
SELECT raw:store.book[*].isbn FROM store_data_json;
```





User-defined Function (UDF)

—



UDF

1. Open “07 UDFs.sql”
2. Replace “catalog.database” with “workshop.<username>” and run the command
3. Run each command one at a time by highlighting it and hitting **Ctrl + Shift + Enter**

```
-- Set your catalog and database.  
USE catalog.database;  
  
-- Create a UDF that returns a literal.  
CREATE OR REPLACE FUNCTION blue()  
RETURNS STRING  
AS  
SELECT 'blue();'  
  
-- Create a UDF that encapsulates other functions.  
CREATE OR REPLACE FUNCTION to_hex(x INT )  
RETURNS STRING  
AS  
SELECT lpad(hex(least(greatest(0, x), 255)), 2, 0);  
  
-- Create a lookup table.  
CREATE OR REPLACE TABLE colors(rgb STRING NOT NULL, name STRING NOT NULL);  
INSERT INTO colors VALUES  
('FF00FF', 'magenta'),  
('FF0080', 'rose'),  
('BFFF00', 'lime'),  
('70D9FF', 'electric blue');  
  
-- Create function to translate an RGB colour to its name.  
CREATE OR REPLACE FUNCTION from_rgb_scalar(rgb STRING)  
RETURNS STRING  
AS  
SELECT FIRST(name) FROM colors WHERE colors.rgb = from_rgb_scalar.rgb;  
  
-- Select the names of the colours.  
SELECT from_rgb_scalar(rgb) FROM values ('70D9FF'), ('BFFF00') AS codes(rgb);
```





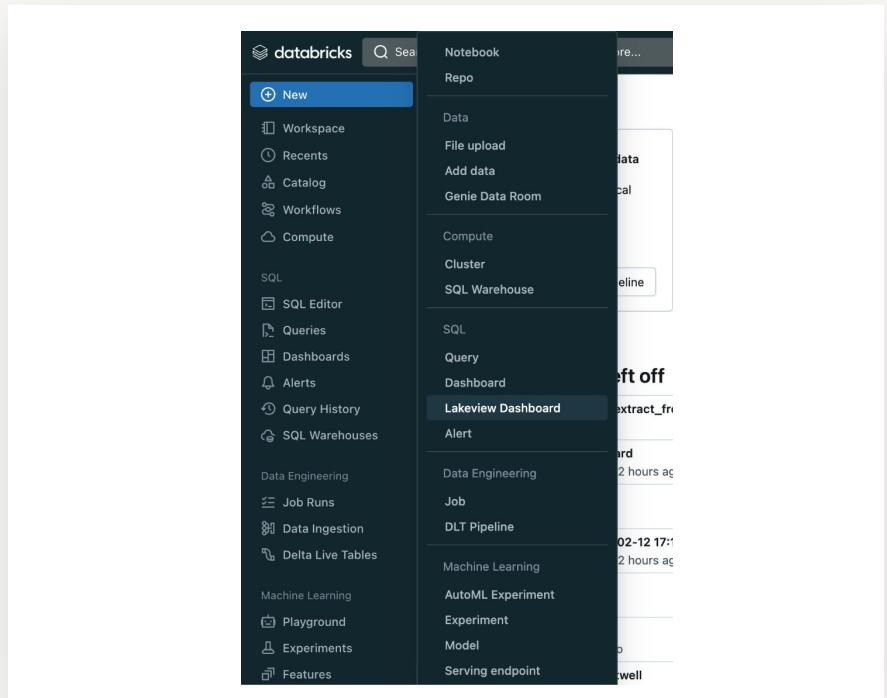
Dashboards

Presented by Junchi



Create a Lakeview Dashboard

1. Click on **New**
2. Select **Lakeview Dashboard**



Select the DIM Customer Dataset

1. Switch over to the **Data** tab and click **Select a table**
2. In **Catalog** enter “workshop”
3. In **Database** enter your username
4. Select the “dim_customer” table

The screenshot shows the 'Select table' interface in the Databricks Unity Catalog. At the top, there is a search bar labeled 'Search tables' and two filter dropdowns: 'workshop X' and 'marat_levit X'. Below these filters, a list of tables is displayed:

- store_data_json
- fact_api_sale_items
- dim_products
- fact_api_sales
- vw_sales_cost_location
- dim_customer
- dim_locations

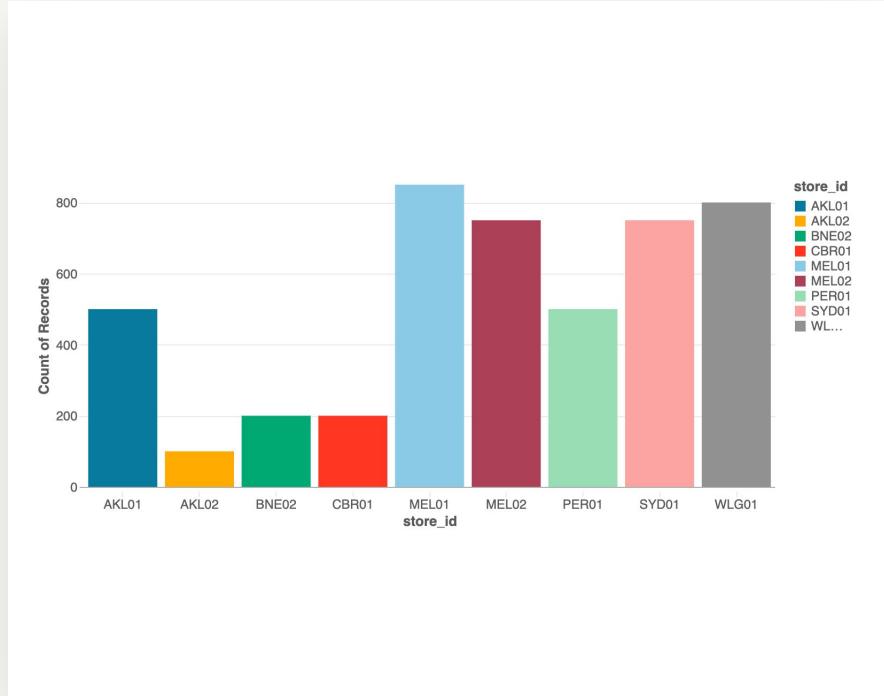
Each table entry includes the schema (e.g., workshop.marat_levit) and the owner (Marat Levit). There is also a 'Clear filters' button at the top right of the filter section.



Create Customer by Store

How many customers does each store have?

1. Switch over to the **Canvas** tab and click the **Create a visualization** button
2. Place the visualization onto the canvas
3. For **X Axis** select “store_id”
4. For **Y Axis** select “COUNT(*)”
5. For **Color/Group by** select “store_id”



Create a Sales by Week Dataset

1. Switch over to the **Data** tab and click the **Create from SQL** button
2. Enter the query on the right and run it
3. Rename the dataset to “Sales by Week”

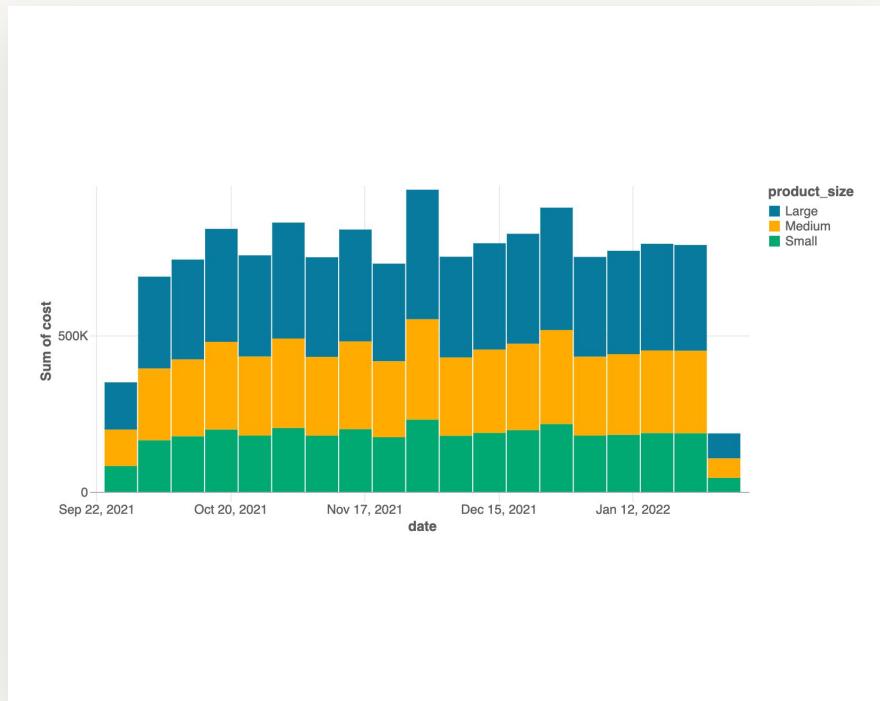
```
select
    sales.ts::timestamp as date,
    items.product_cost::double as cost,
    items.product_id as product,
    items.product_size,
    sales.sale_id,
    sales.store_id
from
    workshop.database.fact_apj_sales sales
    join workshop.database.fact_apj_sale_items items
        on items.sale_id = sales.sale_id
```



Create Sales by Week

Sales by week by product size purchased

1. Switch over to the **Canvas** tab and click the **Create a visualization** button
2. For **Dataset** select “Sales by Week”
3. For **X Axis** select “date”
4. For **Y Axis** select “cost”
5. For **Color/Group by** select “product_size”



Create a Sales by Region Dataset

1. Switch over to the **Data** tab and click the **Create from SQL** button
2. Enter the query on the right and run it
3. Rename the dataset to “Sales by Region”

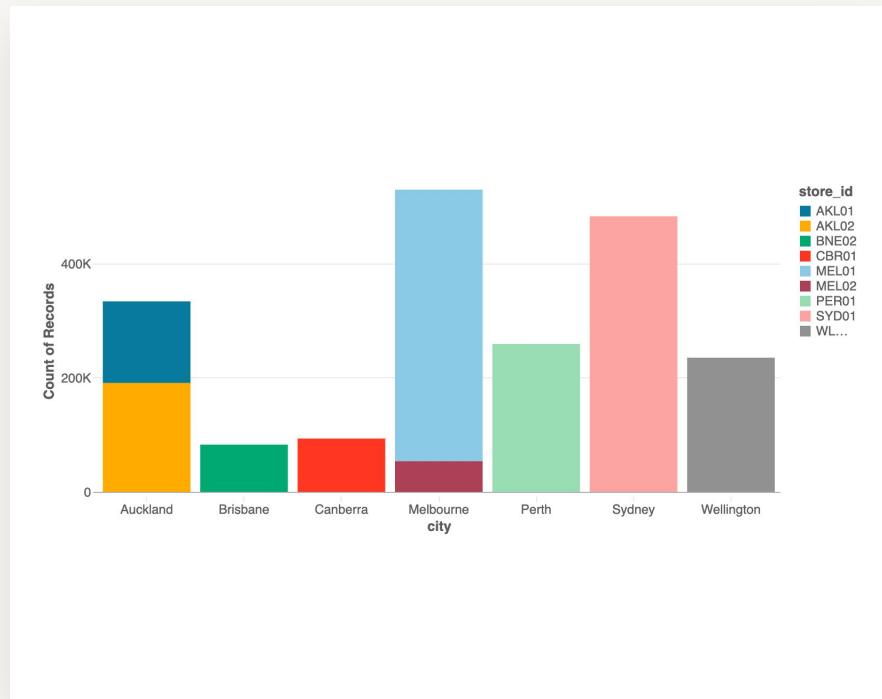
```
select
    sales.ts::timestamp as date,
    sales.store_id,
    sales.sale_id,
    items.product_cost::double as cost,
    locations.city
from
    workshop.database.fact_apj_sales sales
    join workshop.database.fact_apj_sale_items items
        on items.sale_id = sales.sale_id
    join workshop.database.dim_locations locations
        on sales.store_id = locations.id
```



Create Sales by Region

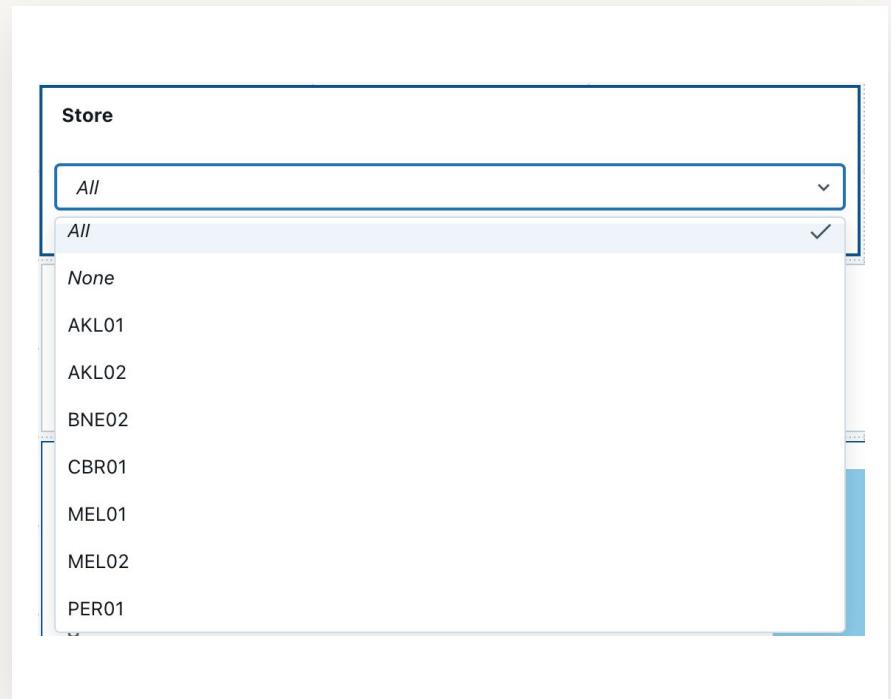
Which stores are performing well or poorly?

1. Switch over to the **Canvas** tab and click the **Create a visualization** button
2. For **Dataset** select “Sales by Region”
3. For **X Axis** select “city”
4. For **Y Axis** select “COUNT(*)”
5. For **Color/Group by** select “store_id”



Create Store Filter

1. On the **Canvas** tab click the **Add a filter** button
2. Add all three datasets to the **Filter on** section and select “store_id” as the filter column for each dataset



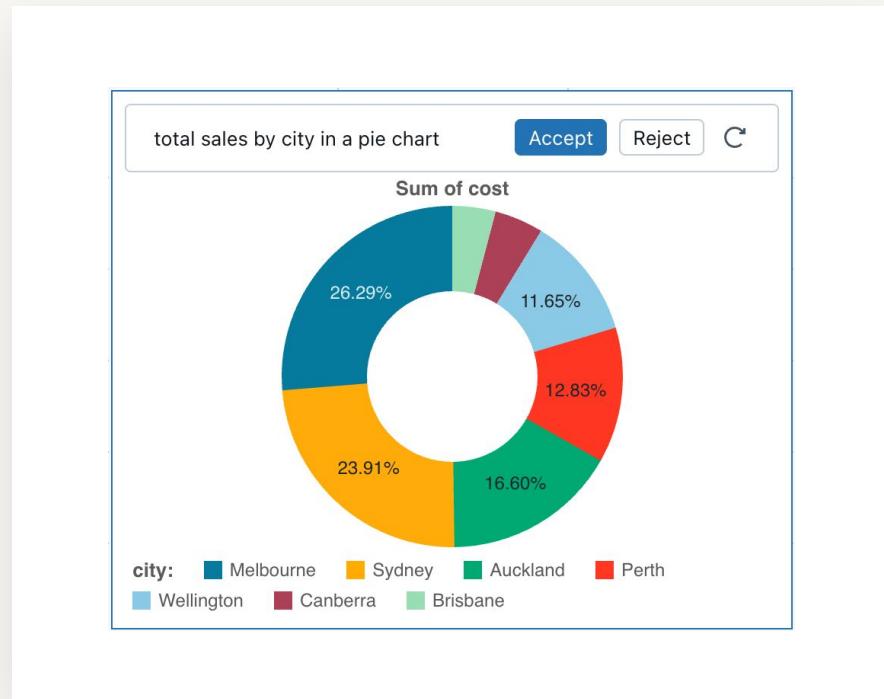


Natural Language Visualisations



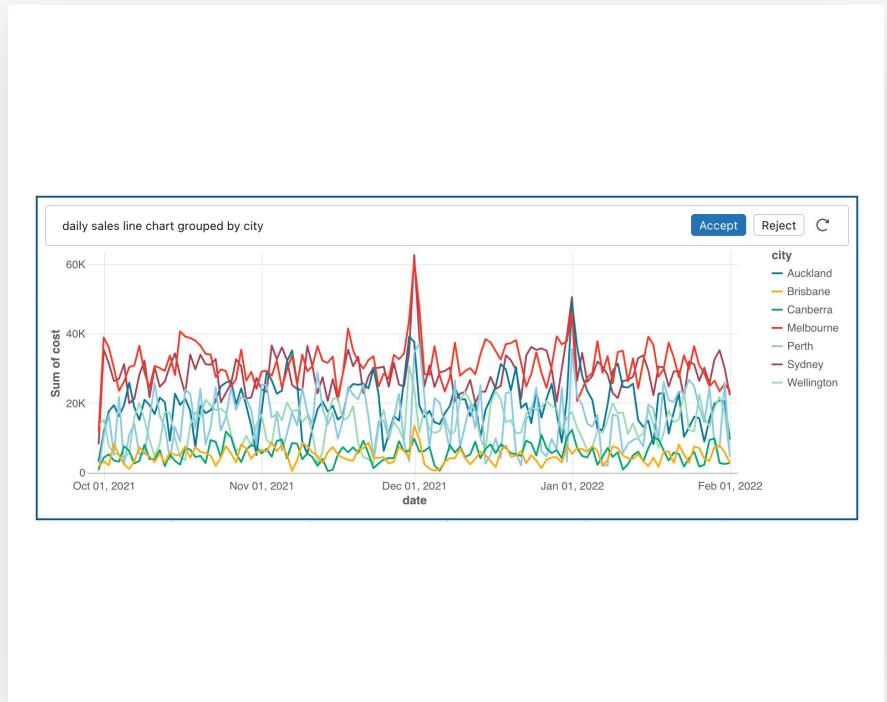
Create Sales by City

1. On the **Canvas** tab and click the **Create a visualization** button
2. In the **Describe a chart...** section enter “total sales by city in a pie chart”
3. Hit **Enter**



Create Daily Sales by City

1. On the **Canvas** tab and click the **Create a visualization** button
2. In the **Describe a chart...** section enter “daily sales line chart grouped by city”
3. Hit **Enter**





Governance



View Tables in Unity Catalog

1. From the side panel, click on **Catalog**
2. Navigate to your table by drilling down into the **workshop** catalog, followed by your **username** database, and select “dim_customer”

The screenshot shows the Databricks Catalog Explorer interface. On the left, a sidebar navigation includes 'New', 'Workspace', 'Recents', 'Catalog' (which is selected and highlighted in red), 'Workflows', 'Compute', 'SQL', 'SQL Editor', 'Queries', 'Dashboards', 'Alerts', 'Query History', and 'SQL Warehouses'. Below these are sections for 'Data Engineering', 'Machine Learning', and 'Serving'. The main content area is titled 'Catalog Explorer' and shows the path 'Catalogs > workshop > marat_levit > dim_customer'. The table 'dim_customer' is listed under the 'marat_levit' database. To the right of the table listing are buttons for '+ Add', 'Browse DBFS', 'Serverless Shared...', 'Serverless', and 'Create'. Below the table listing is a 'Tags' section with an 'Add tags' button. A callout box provides an 'AI Suggested Comment' about the table's purpose. The table details view shows columns: unique_id, pk, id, store_id, name, and email, each with a string type and a comment field. There are also 'Tags' and 'AI generate' buttons.



View Column and Sample Data

1. Click through the **Columns** and **Sample Data** tabs

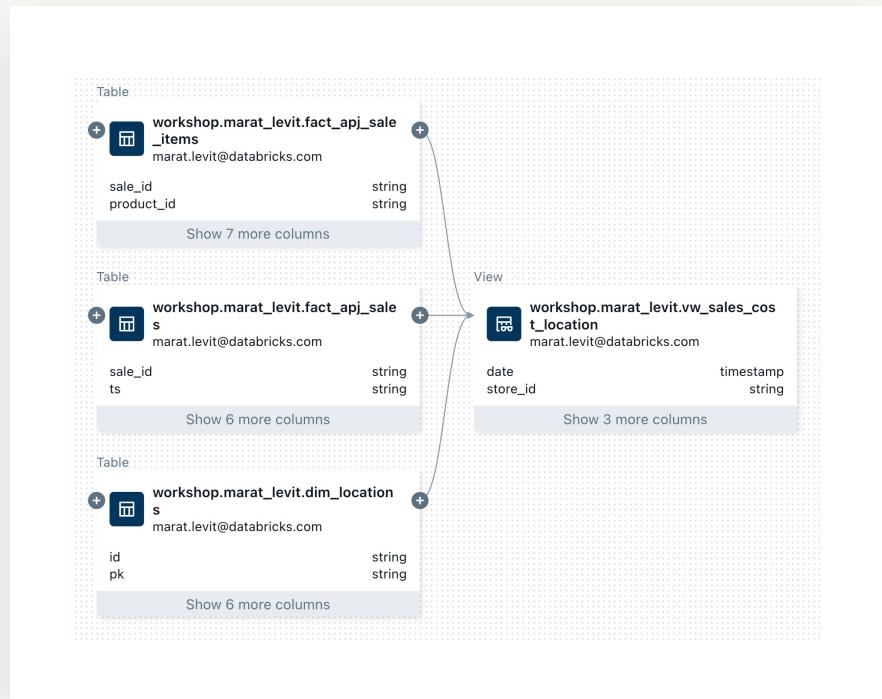
Columns		Sample Data		Details		Permissions		History		Lineage		Insights		Quality															
<input type="text"/> Filter columns...																													
Ai generate																													
Column	Type	Comment	Tags																										
unique_id	string	?	?																										
pk	string	?	?																										
id	string	?	?																										
store_id	string	?	?																										
name	string	?	?																										
email	string	?	?																										

Columns		Sample Data		Details		Permissions		History		Lineage		Insights		Quality	
? unique_id															
1	BNE02-1	1	1	BNE02	Stephanie Brown	howardalejandra@example.net									
2	PER01-1	2	1	PER01	Cindy Lester	walkerbrittany@example.net									
3	CBR01-1	3	1	CBR01	Steven White	tammybyrd@example.com									
4	MEL01-1	4	1	MEL01	Brian Chandler	davidgrant@example.net									
5	MEL02-1	5	1	MEL02	Cynthia Moody	fparsons@example.org									
6	AKL01-1	6	1	AKL01	Jessica Williams	david17@example.com									
7	AKL02-1	7	1	AKL02	Matthew Clark	nwalker@example.net									
8	WLG01-1	8	1	WLG01	Kimberly Gray	qvega@example.com									
9	SYD01-1	9	1	SYD01	Peter Carr	paulacurry@example.net									
10	BNE02-2	10	2	BNE02	Christopher Cooper	campbelljohn@example.net									
11	PER01-2	11	2	PER01	Brian McMahon Jr.	hbarnes@example.org									
12	CBR01-2	12	2	CBR01	Summer Wilson	kfowler@example.org									



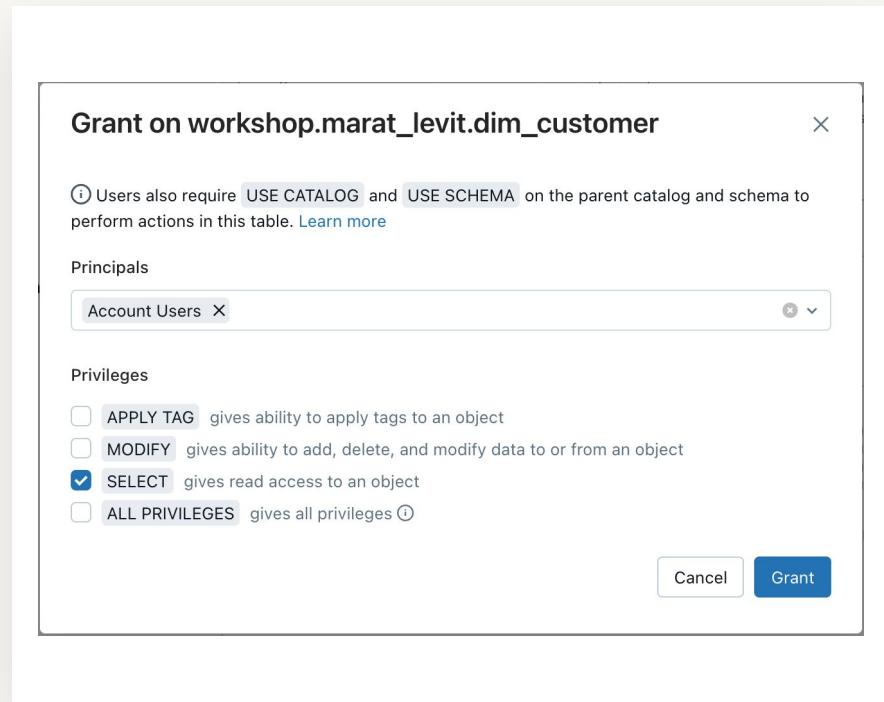
Table and Column Lineage

1. Select the "vw_sales_cost_location" view from the **Catalog Explorer**
2. Click through to the **Lineage** tab
3. Click the **See lineage graph** button



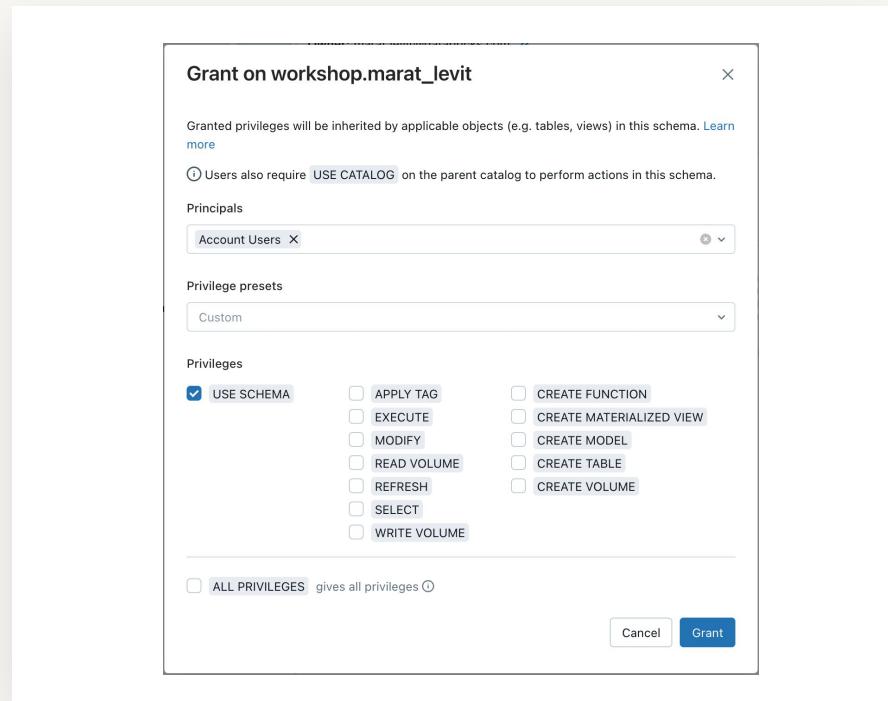
Manage Table Permissions

1. Navigate to the **Permissions** tab
2. Click the **Grant** button
3. Enter “Account Users” into the **Principals** field
4. Tick “SELECT” under **Privileges**
5. Click the **Grant** button to grant those privileges



Manage Catalog Permissions

1. Navigate to the “workshop” catalog in the **Catalog Explorer** panel
2. Navigate to the **Permissions** tab
3. Click the **Grant** button
4. Enter “Account Users” into the **Principals** field
5. Tick “USE SCHEMA” under **Privileges**
6. Click the **Grant** button to grant those privileges





Management



Billing Usage

Analyze and optimize DBU consumption

1. Navigate to the “system” catalog in the **Catalog Explorer** panel
2. Open the “billing” database and click on the “usage” table
3. With system tables, your account’s billable usage data is centralized and routed to all regions, so you can view your account’s global usage from whichever region your workspace is in.
4. Usage is aggregated hourly, so each record in **system.billing.usage** represents an hour of billable usage.

Catalog Explorer unity-catalog-demo ⚙️ ⓘ Send feedback

Catalogs > system > billing > **system.billing.usage**

Owner: System user ⓘ Popularity: ⓘ

Tags: Add tags

Columns Sample Data Details Permissions

Filter columns...

Column	Type
account_id	string
workspace_id	string
record_id	string
sku_name	string
cloud	string
usage_start_time	timestamp
usage_end_time	timestamp
usage_date	date
custom_tags	map
usage_unit	string
usage_quantity	decimal
usage_metadata	struct

Delta Sharing

Clean Rooms

External Data

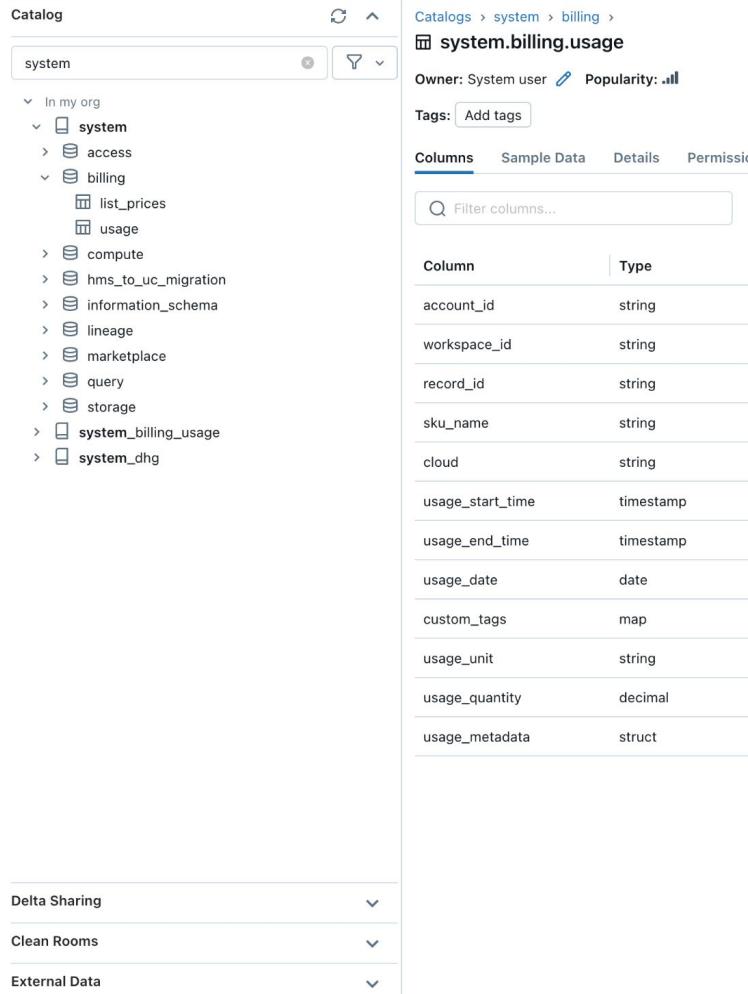


Table and Column Lineage

Programmatically query lineage data

1. Navigate to the “system” catalog in the **Catalog Explorer** panel
2. Open the “lineage” database and click on the “column_lineage” or “table_lineage” tables
3. The **table lineage table** includes a record for each read or write event on a Unity Catalog table or path.
4. The **column lineage table** does not include events that do not have a source. For example, if you insert into a column using explicit values, it is not captured.

Catalog Explorer unity-catalog-demo ⚙️ 💬 Send feedback

Catalogs > system > lineage > system.lineage.table_lineage

Owner: System user Popularity: 🔍

Tags: Add tags

Columns Sample Data Details Permissions

Filter columns...

Column	Type
account_id	string
metastore_id	string
workspace_id	long
entity_type	string
entity_id	string
entity_run_id	string
source_table_full_name	string
source_table_catalog	string
source_table_schema	string
source_table_name	string
source_path	string
source_type	string
target_table_full_name	string
target_table_catalog	string
target_table_schema	string
target_table_name	string
target_path	string
target_type	string

Delta Sharing

Clean Rooms

External Data

Warehouse Management

1. Navigate to the **SQL Warehouses** the side panel
2. Click on your SQL Warehouse name and select the **Monitoring** tab. Here you'll find:
 - a. **Live statistics** show the currently running and queued queries, active SQL sessions, the warehouse status, and the current cluster count.
 - b. The **peak query count chart** shows the maximum number of concurrent queries, either running or queued, on the warehouse during the selected time frame.
 - c. The **running clusters chart** shows the number of clusters allocated to the warehouse during the selected time frame.
 - d. The **query history table** shows all of the queries active during the selected time frame

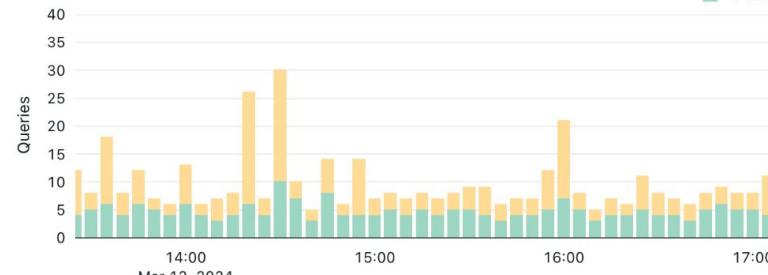
Running queries ⓘ

0

Queued queries ⓘ

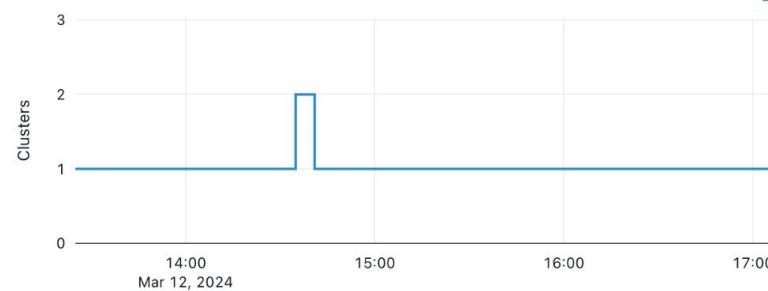
0

Peak query count



⌚ just now

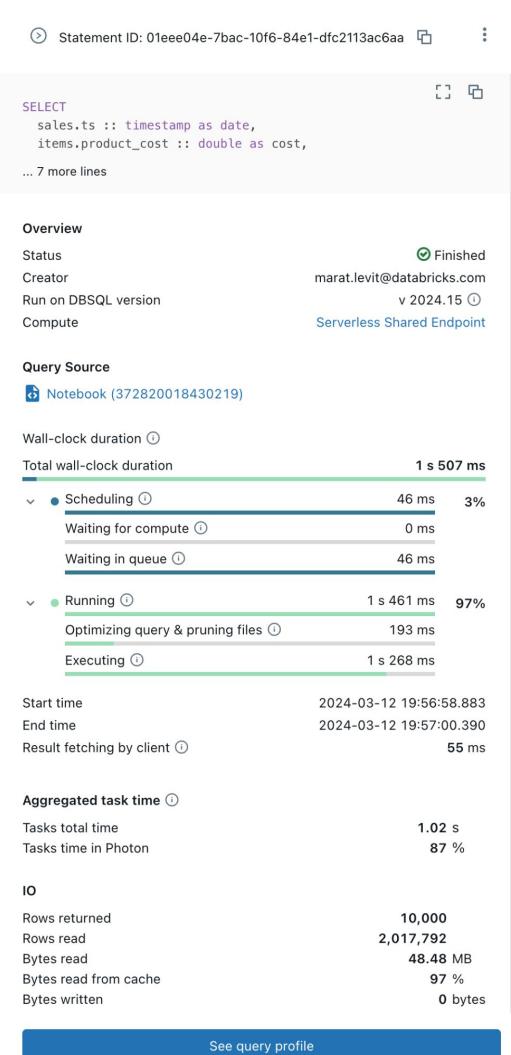
Running clusters



⌚ just now

Query Inspection

- Click on any of the successful queries in the **query history table**





databricks