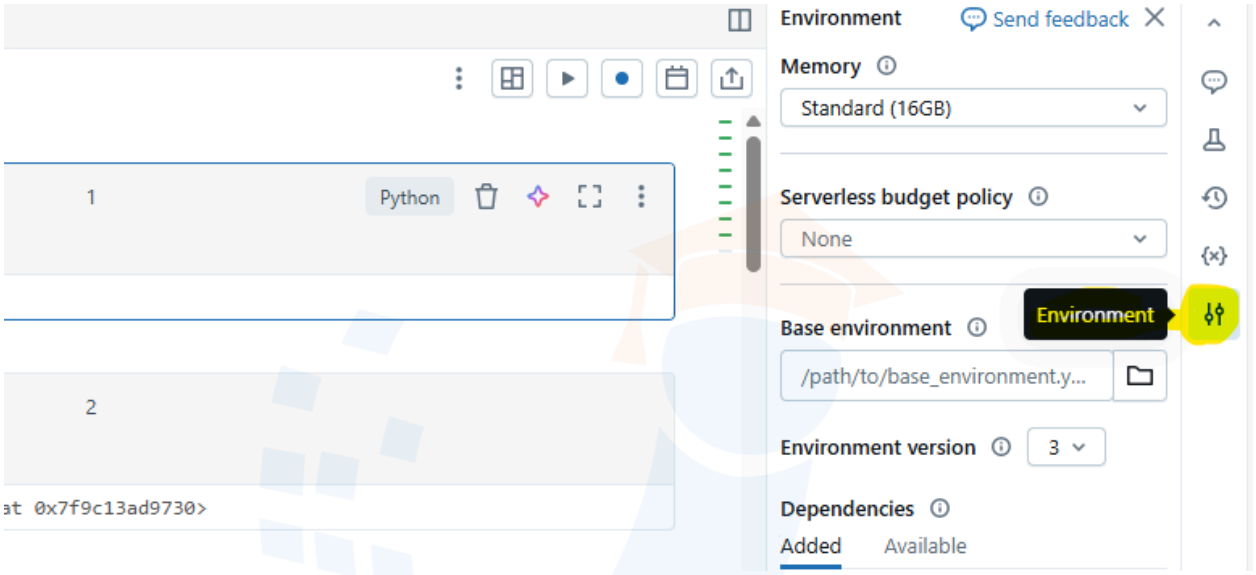
Session 5: Serverless Compute

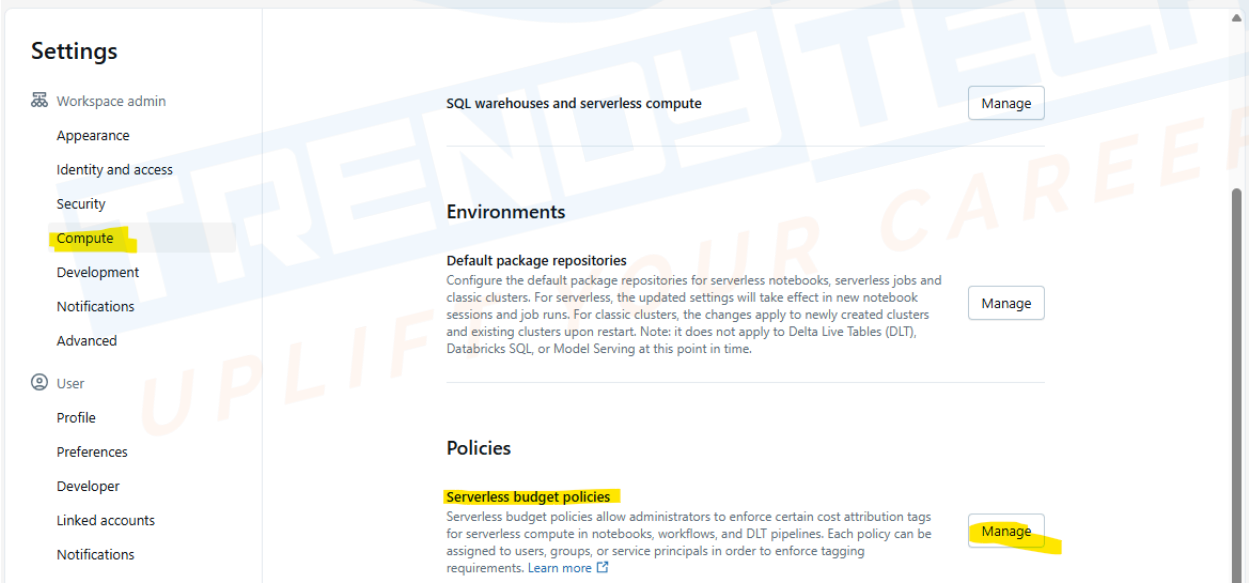
Since there is no Classic Compute in the free edition, running a cell will automatically connect to Serverless Compute

Environment in the right pane is applicable only for serverless



To create budget policy -

User → settings → compute → policies



Workspace settings › Compute › Serverless budget policies ›

Create budget policy

Learn more

General

Name

demo-budget-policy

Workspaces

The workspaces that this budget policy applies to. If empty, the policy applies to all workspaces.

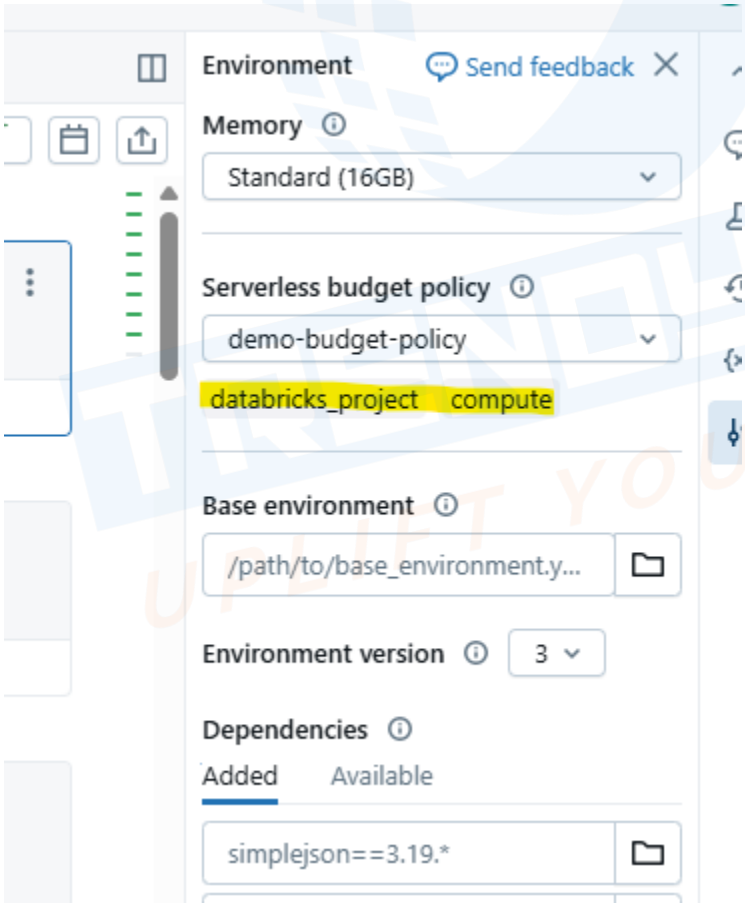
workspace (1985386938000145) ×

Tags

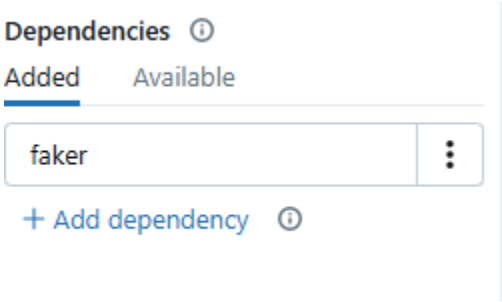
Used for attribution. Serverless budget policies only support fixed tags.

Tag	Value
project	databricks_project
topic	compute

+ Add tag

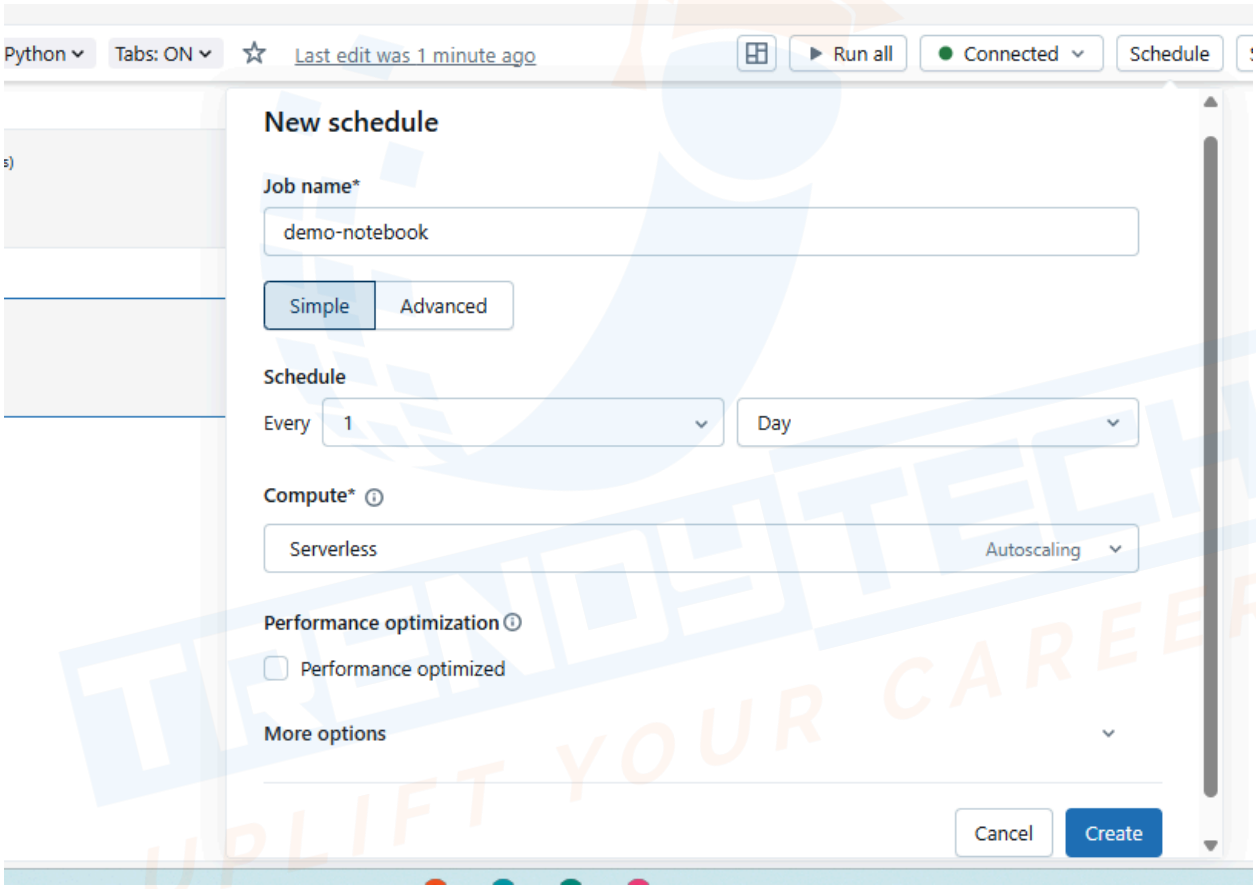


Installing dependencies:



Click ‘enter’ after entering the name and then click on apply

To Schedule the notebook:



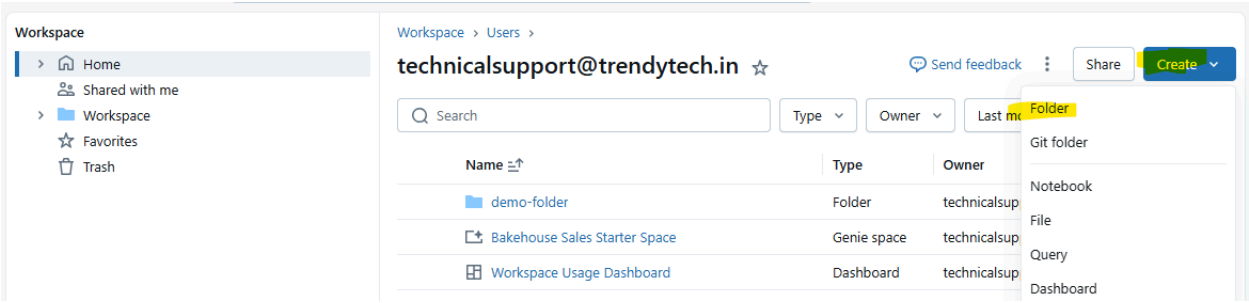
SQL warehouses:

In Free Edition:

- You cannot edit SQL warehouse permissions.
- There’s only one user (your account) in the workspace.
- Since no additional users or groups exist, the Permissions tab / edit option is hidden.
- Effectively, you are the only owner and user of the warehouse.

Delta Lake practical session 1:

Go to workspace create a folder



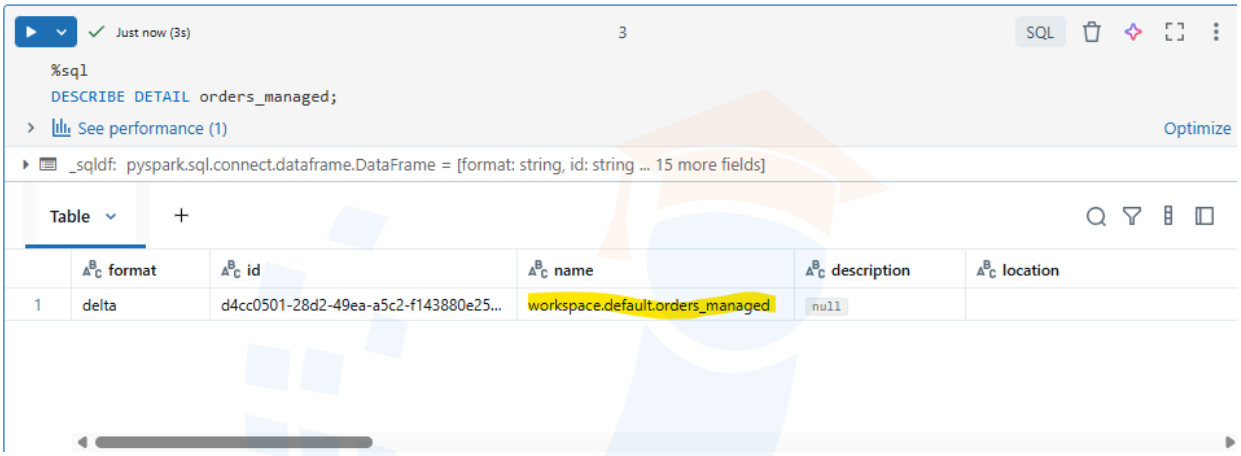
Create a notebook inside this folder

Pyspark:

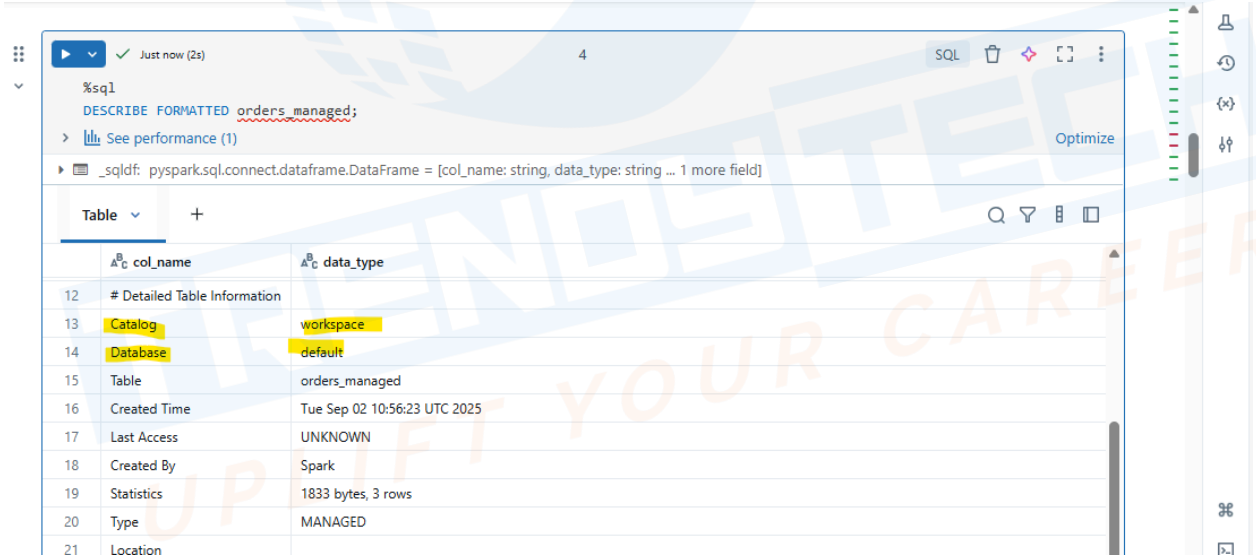
```
spark.sql("""
CREATE OR REPLACE TABLE orders_managed (
    order_id BIGINT,
    sku STRING,
    product_name STRING,
    product_category STRING,
    qty INT,
    unit_price DECIMAL(10,2)
) USING DELTA
""")
```

```
spark.sql("""
INSERT INTO orders_managed VALUES
  (1,  'A101', 'iPhone Charger',       'Electronics', 2, 19.99),
  (2,  'A102', 'Bluetooth Headphones', 'Electronics', 1, 49.50),
  (3,  'A103', 'HDMI Cable',           'Electronics', 3, 5.99)
""")
```

It will be saved in the default unity catalog present



SQL code:

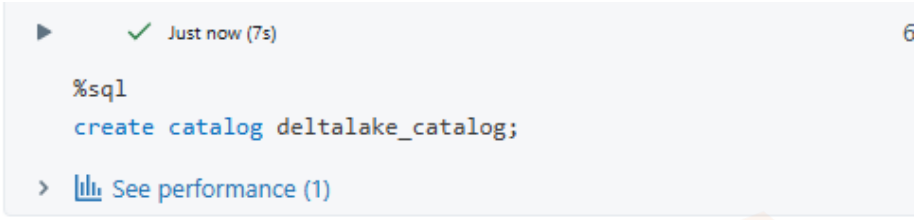


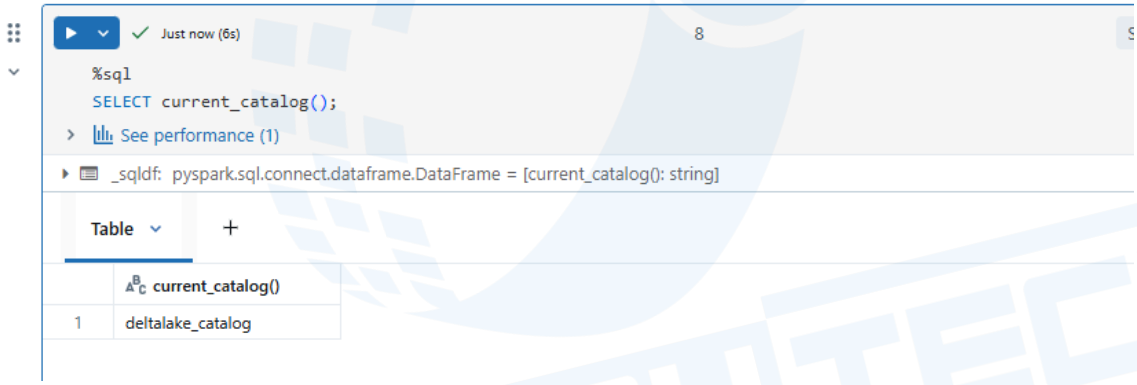
Pyspark:

```
df = spark.sql("DESCRIBE DETAIL orders_managed")
df.show(truncate=False)

df = spark.sql("DESCRIBE FORMATTED orders_managed")
df.show(truncate=False)
```

Create catalog

One of the most significant differences is the absence of a fully configurable Unity Catalog. Features like MANAGED LOCATION and EXTERNAL LOCATION, which link your Databricks environment to your own cloud storage (e.g., Azure Data Lake Storage or AWS S3), are not available.



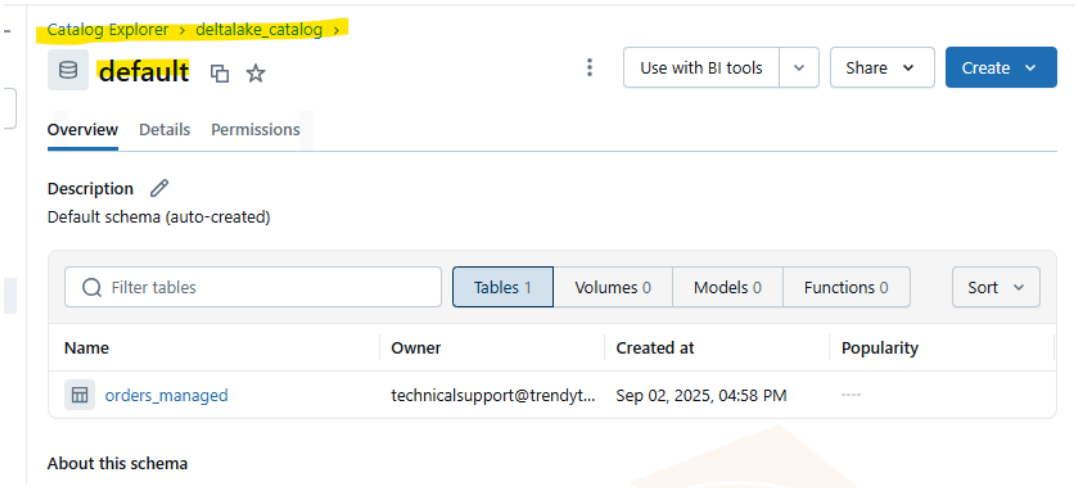


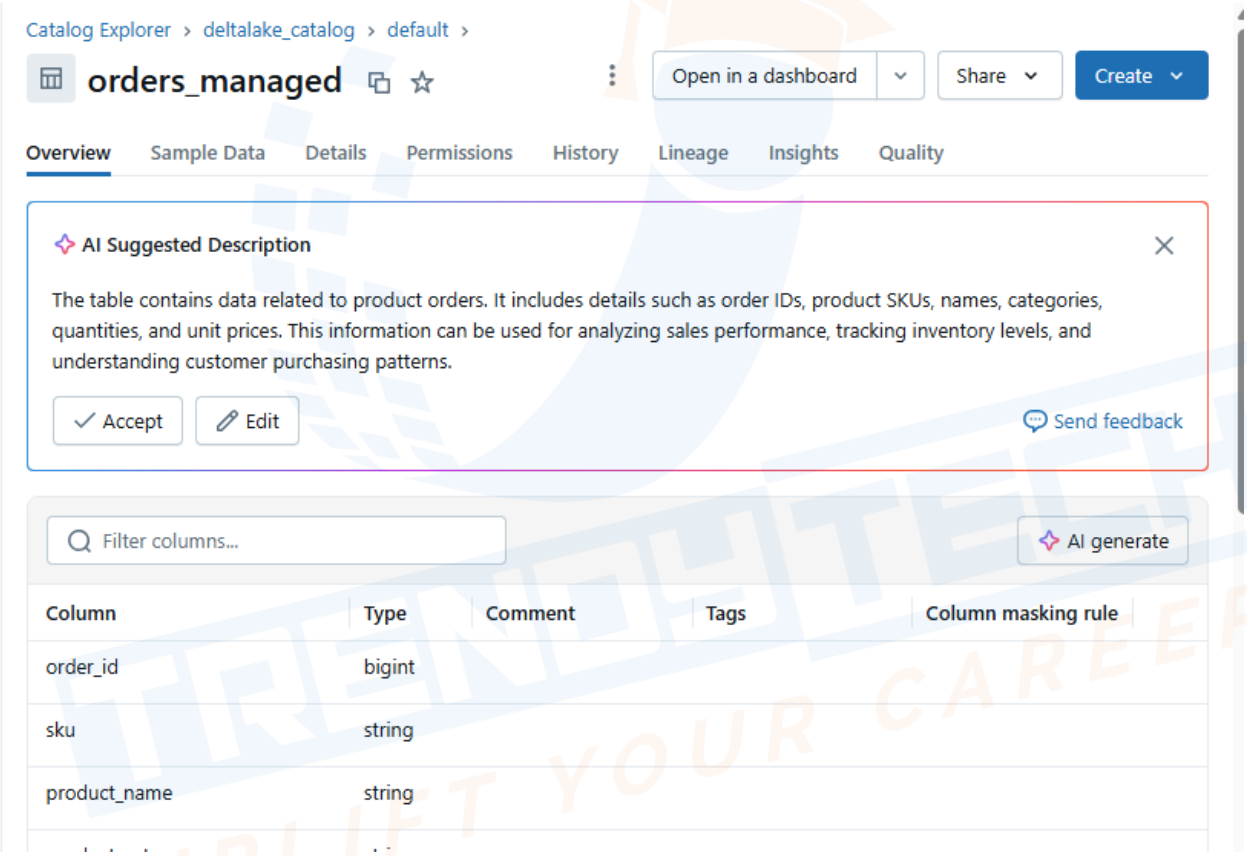
Pyspark:

df = spark.sql("SELECT current_catalog()")

df.show(truncate=False)

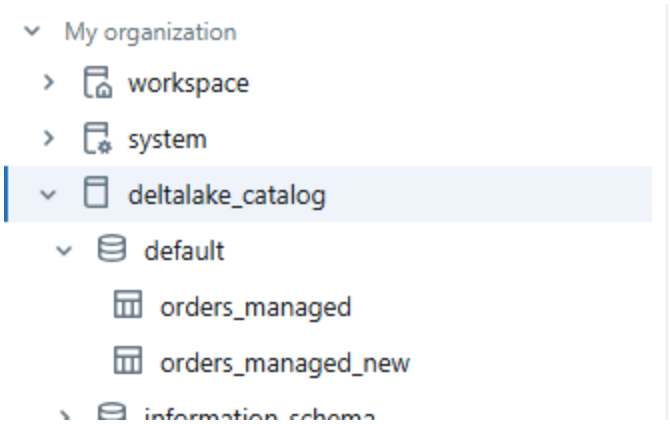
Table will be created in below location catalog → default





Delta builder APIs


```
from pyspark.sql.types import StructType, StructField, LongType, StringType, IntegerType, DecimalType

# Define schema
schema = StructType([
    StructField("order_id", LongType(), True),
    StructField("sku", StringType(), True),
    StructField("product_name", StringType(), True),
    StructField("product_category", StringType(), True),  # comment not supported in PySpark schema
    StructField("qty", IntegerType(), True),
    StructField("unit_price", DecimalType(10,2), True)
])

# Create empty DataFrame with schema
df = spark.createDataFrame([], schema)

# Save as managed Delta table
df.write.format("delta").saveAsTable("orders_managed_new")
```



Delta lake practical session 2:

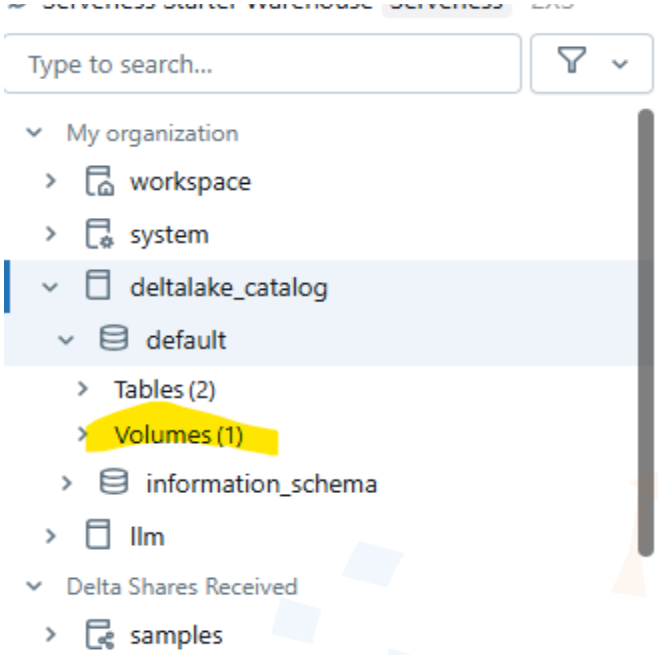
Create a new notebook - ‘deltalake-demo-2’

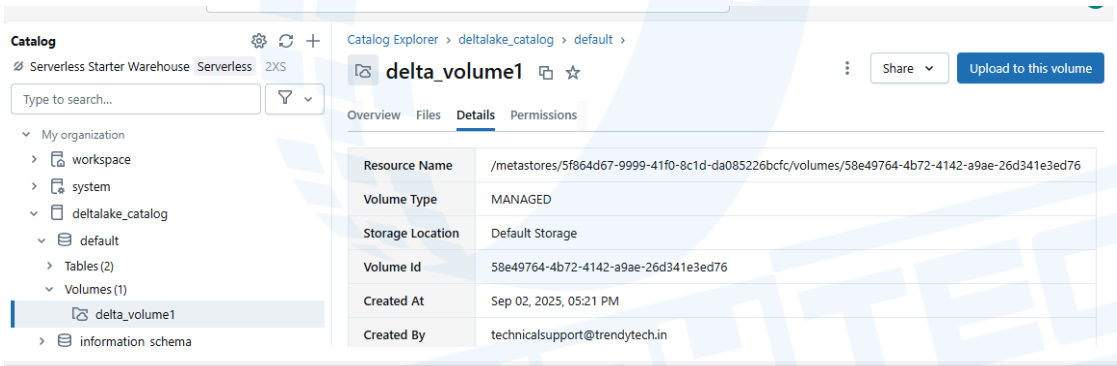
Use the same catalog created in the previous session

Create volume and refresh (2 tables in the screenshot because we had created order_managed_new under the catalog)

```
%sql
create volume if not exists deltalake_catalog.default.delta_volume1
```

Pyspark equivalent:
spark.sql("CREATE VOLUME IF NOT EXISTS deltalake_catalog.default.delta_volume1")






```
dbutils.fs.mkdirs("/Volumes/deltalake_catalog/default/delta_volume1/ordersdata1")
```

See performance (1) Optimize

Just now (1s) 4

```
%fs mkdirs /Volumes/deltalake_catalog/default/delta_volume1/ordersdata2
```

See performance (1)

Just now (5s) 5 Python

```
%fs ls /Volumes/deltalake_catalog/default/delta_volume1
```

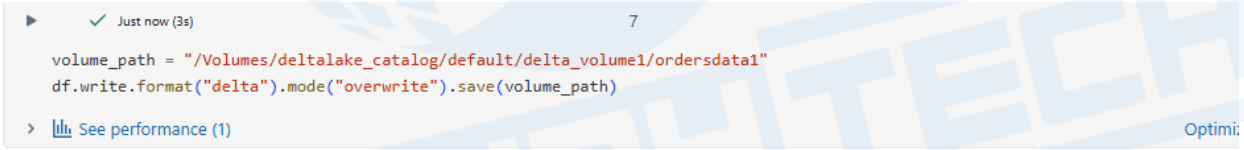
See performance (2)

Table +

	path	name	size	modificationTime
1	dbfs:/Volumes/deltalake_catalog/default/delta_volume1/ordersdata1/	ordersdata1/	0	1756814031295
2	dbfs:/Volumes/deltalake_catalog/default/delta_volume1/ordersdata2/	ordersdata2/	0	1756814031295

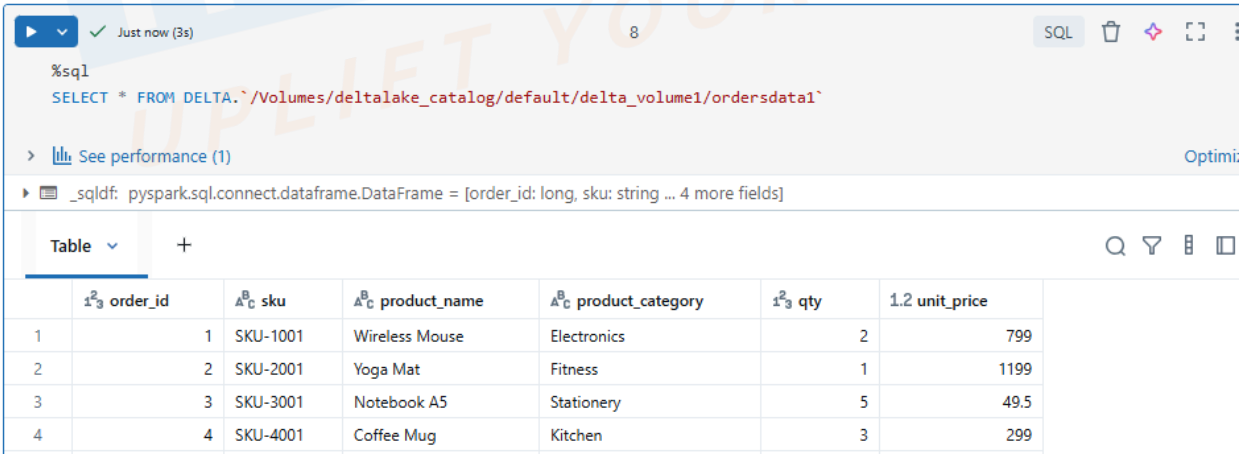
Above is already a valid python code.

```
data = [
    (1, "SKU-1001", "Wireless Mouse", "Electronics", 2, 799.00),
    (2, "SKU-2001", "Yoga Mat", "Fitness", 1, 1199.00),
    (3, "SKU-3001", "Notebook A5", "Stationery", 5, 49.50),
    (4, "SKU-4001", "Coffee Mug", "Kitchen", 3, 299.00),
    (5, "SKU-5001", "LED Bulb", "Electronics", 4, 149.99)
]

# Define schema (column names)
columns = ["order_id", "sku", "product_name", "product_category", "qty",
"unit_price"]

# Create DataFrame
df = spark.createDataFrame(data, columns)

# Display DataFrame
display(df)
```



Above is already a valid python code.

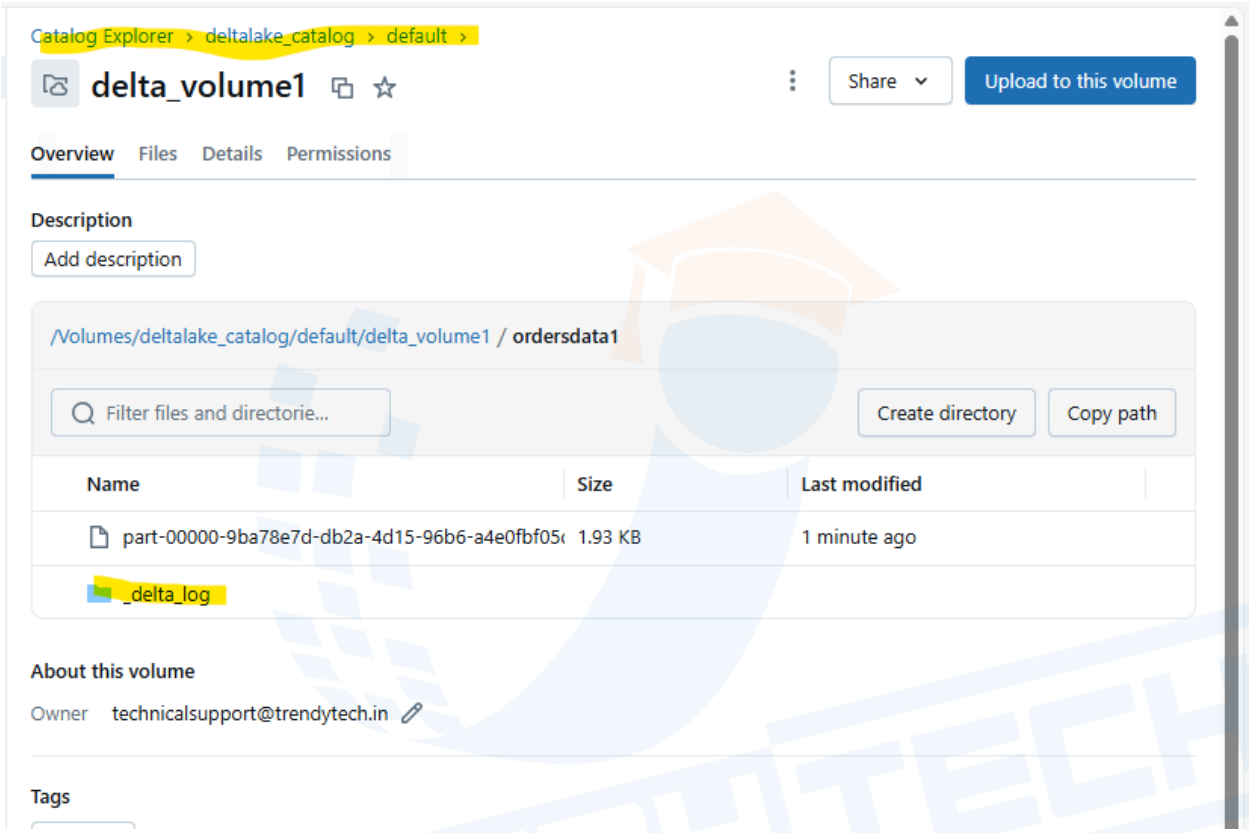
SQl code:

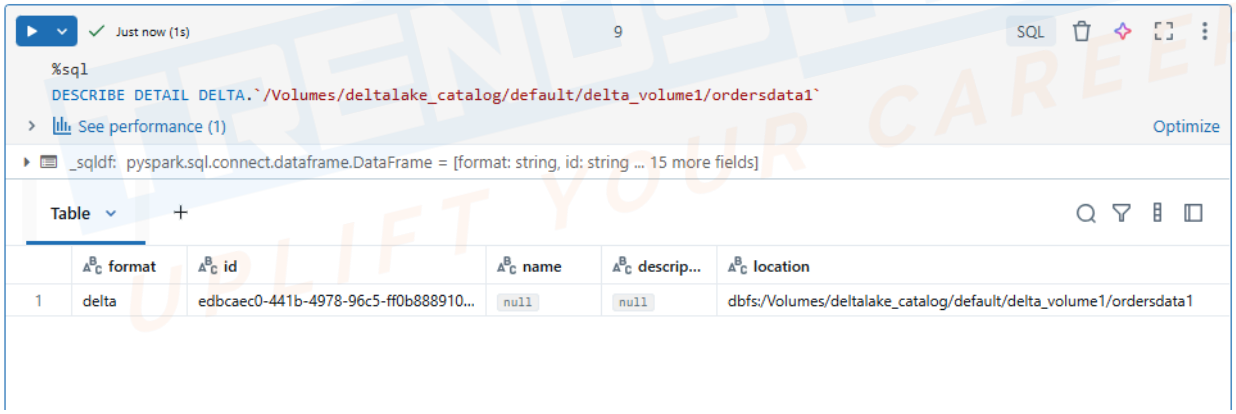


	order_id	sku	product_name	product_category	qty	unit_price
1	1	SKU-1001	Wireless Mouse	Electronics	2	799
2	2	SKU-2001	Yoga Mat	Fitness	1	1199
3	3	SKU-3001	Notebook A5	Stationery	5	49.5
4	4	SKU-4001	Coffee Mug	Kitchen	3	299

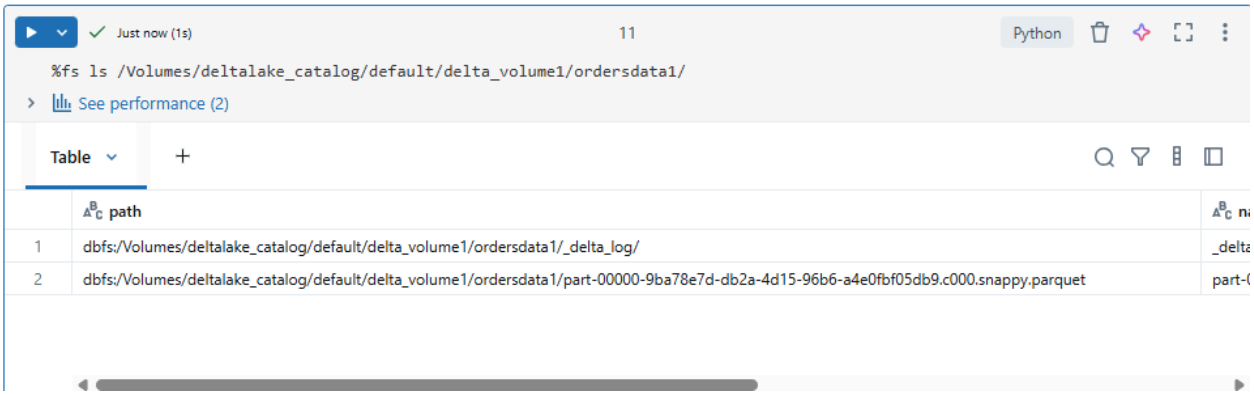
Pyspark equivalent:

```
df = spark.read.format("delta") \
    .load("/Volumes/deltalake_catalog/default/delta_volume1/ordersdata1")

df.show(truncate=False)
```

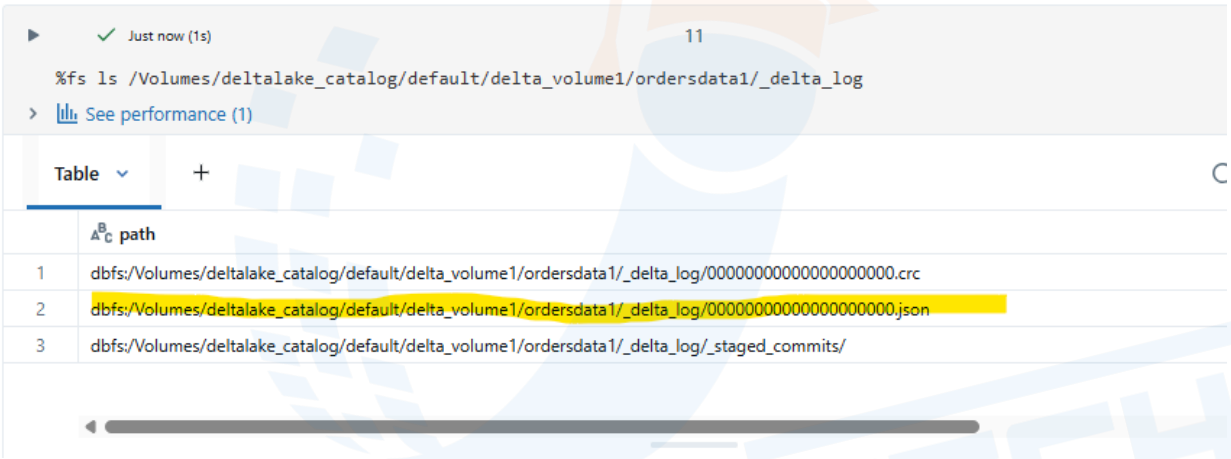




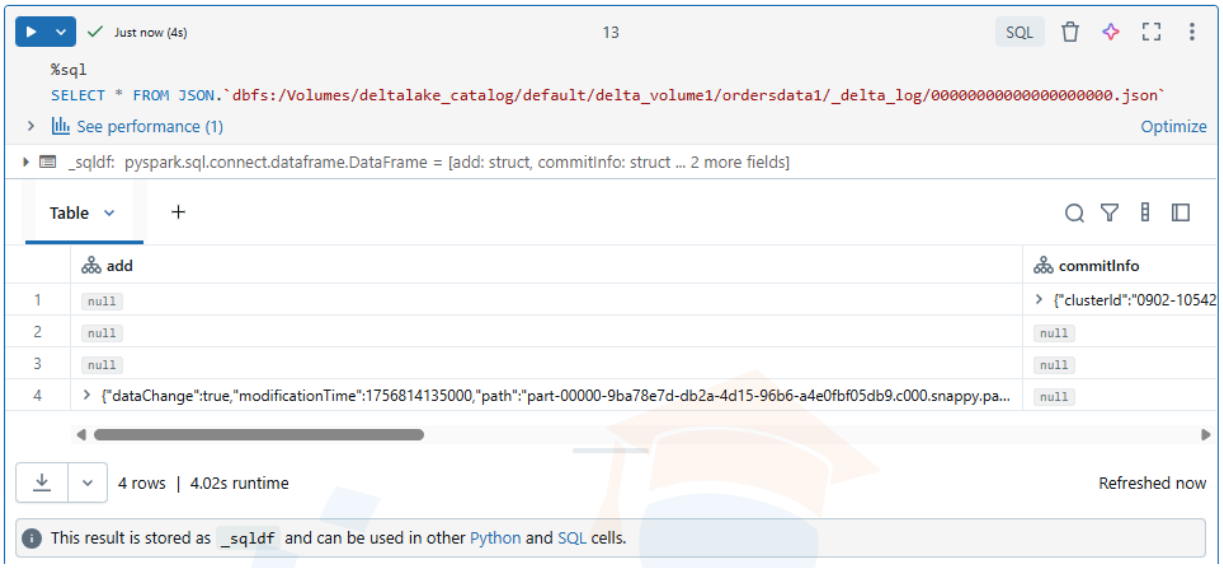

```
df = spark.sql("""
DESCRIBE DETAIL DELTA.`/Volumes/deltalake_catalog/default/delta_volume1/ordersdata1`
""")
df.show(truncate=False)
```



Copy the path of the json file



Sql way:



Pyspark code:
df = spark.read.format("json") \

.load("dbfs:/Volumes/deltalake_catalog/default/delta_volume1/ordersdata1/_delta_log/00000000000000000000.json")

df.show(truncate=False)

Parquet path:
SQL code:

Pyspark code:
df = spark.read.format("parquet") \

.load("dbfs:/Volumes/deltalake_catalog/default/delta_volume1/ordersdata1/part-00000-9ba78e7d-db2a-4d15-96b6-a4e0fbf05db9.c000.snappy.parquet")

df.show(truncate=False)

Adding two more records:

```
# Orders sample data
data = [
    (6, "SKU-6001", "Running Shoes", "Fitness", 1, 2599.00),
    (7, "SKU-7001", "Desk Chair", "Furniture", 1, 5499.00)
]

# Define schema (column names)
columns = ["order_id", "sku", "product_name", "product_category", "qty", "unit_price"]
# Create DataFrame
new_df = spark.createDataFrame(data, columns)

# Display DataFrame
display(new_df)
```

See performance (1) Optimiz

new_df: pyspark.sql.connect.dataframe.DataFrame = [order_id: long, sku: string ... 4 more fields]

Table +

	order_id	sku	product_name	product_category	qty	unit_price
1	6	SKU-6001	Running Shoes	Fitness	1	2599
2	7	SKU-7001	Desk Chair	Furniture	1	5499

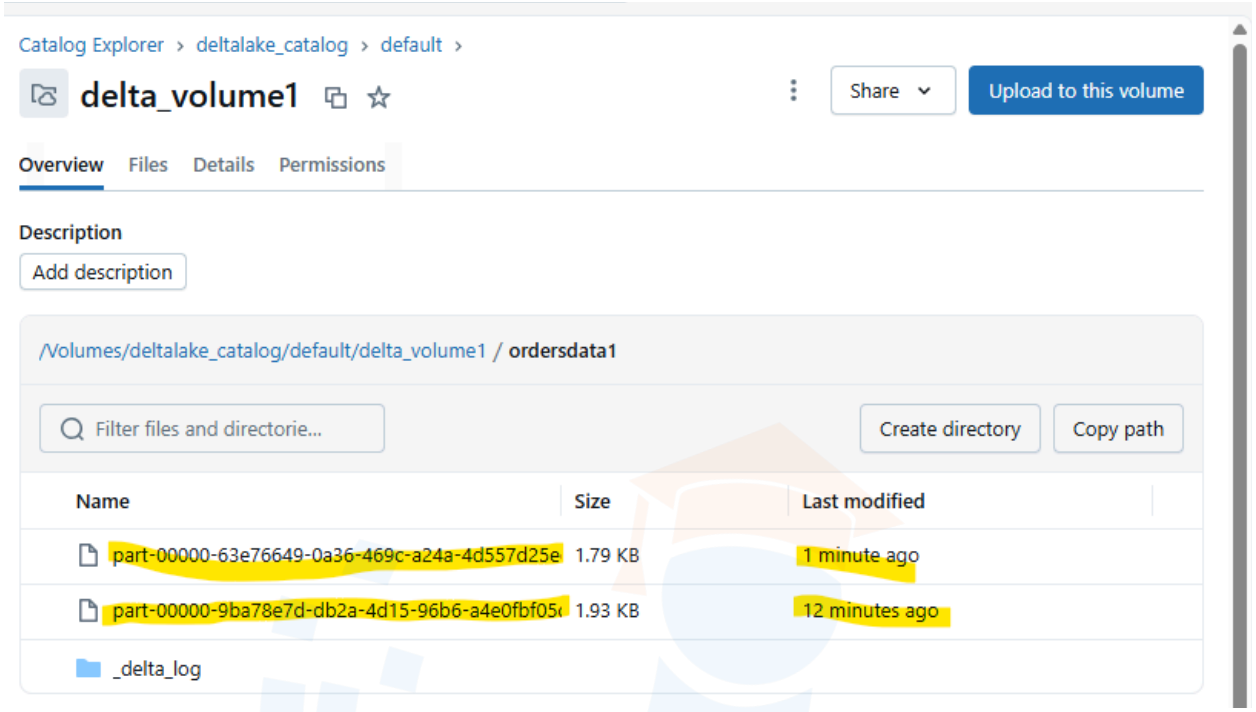
After executing below command

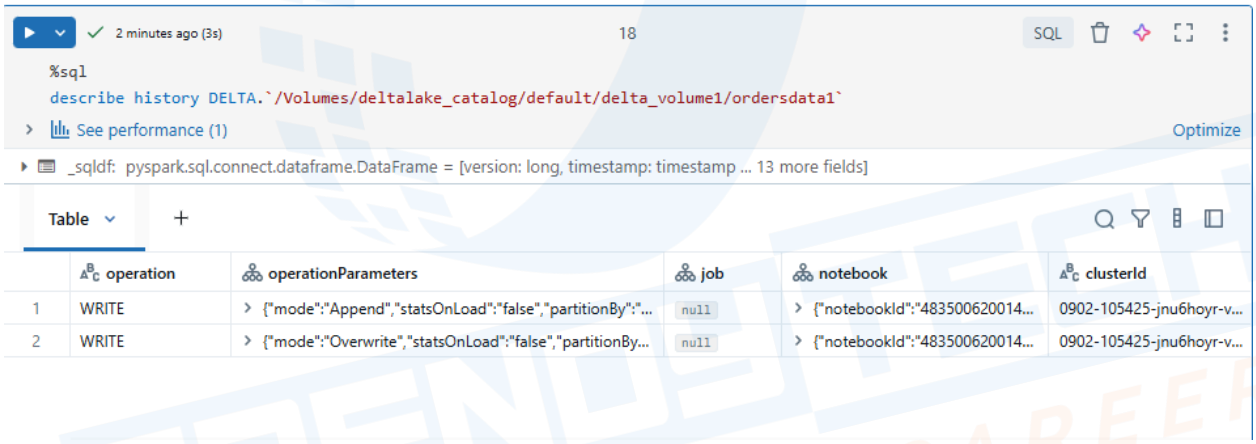
Just now (1s) 16

```
volume_path = "/Volumes/deltalake_catalog/default/delta_volume1/ordersdata1"
new_df.write.format("delta").mode("append").save(volume_path)
```

See performance (1)

Check the volume





df = spark.sql("""
DESCRIBE HISTORY DELTA.`/Volumes/deltalake_catalog/default/delta_volume1/ordersdata1`
""")
df.show(truncate=False)

Practical Session 3:(need azure)

SQL code:

```
%sql
CREATE OR REPLACE TABLE orders_ext_01 (
order_id BIGINT,
sku STRING,
```

```
product_name STRING,
product_category STRING,
qty INT,
unit_price DECIMAL(10,2)
)
using delta
LOCATION '/Volumes/deltalake_catalog/orders/volume2/';
```

Pyspark:

```
spark.sql("""
CREATE OR REPLACE TABLE orders_ext_01 (
   order_id BIGINT,
   sku STRING,
   product_name STRING,
   product_category STRING,
   qty INT,
   unit_price DECIMAL(10,2)
)
USING DELTA
LOCATION '/Volumes/deltalake_catalog/orders/volume2/'
""")
```

```
data = [
    (1, 'SKU-1001', 'Wireless Mouse', 'Electronics', 2, 799.00),
    (2, 'SKU-2001', 'Yoga Mat', 'Fitness', 1, 1199.00),
    (3, 'SKU-3001', 'Notebook A5', 'Stationery', 5, 49.50),
    (4, 'SKU-4001', 'Coffee Mug', 'Kitchen', 3, 299.00),
    (5, 'SKU-5001', 'LED Bulb', 'Electronics', 4, 149.99)
]
columns = ["order_id", "sku", "product_name", "product_category", "qty",
"unit_price"]

df = spark.createDataFrame(data, columns)

df.write.format("delta").mode("overwrite").save("/Volumes/deltalake_catalog/ord
ers/volume2/")
```

Above code is already in pyspark

SQL code:

```
%sql
INSERT INTO deltalake_catalog.default.orders_ext_01
(order_id, sku, product_name, product_category, qty, unit_price)
VALUES
(1, 'SKU-1001', 'Wireless Mouse', 'Electronics', 2, 799.00),
(2, 'SKU-2001', 'Yoga Mat', 'Fitness', 1, 1199.00),
(3, 'SKU-3001', 'Notebook A5', 'Stationery', 5, 49.50),
(4, 'SKU-4001', 'Coffee Mug', 'Kitchen', 3, 299.00),
(5, 'SKU-5001', 'LED Bulb', 'Electronics', 4, 149.99);
```

Pyspark:

```
spark.sql("""
INSERT INTO deltalake_catalog.default.orders_ext_01
(order_id, sku, product_name, product_category, qty, unit_price)
VALUES
(1, 'SKU-1001', 'Wireless Mouse', 'Electronics', 2, 799.00),
(2, 'SKU-2001', 'Yoga Mat', 'Fitness', 1, 1199.00),
(3, 'SKU-3001', 'Notebook A5', 'Stationery', 5, 49.50),
(4, 'SKU-4001', 'Coffee Mug', 'Kitchen', 3, 299.00),
(5, 'SKU-5001', 'LED Bulb', 'Electronics', 4, 149.99)
""")
```

```
%sql
DESCRIBE DETAIL deltalake_catalog.default.orders_ext_01
```

Pyspark:

```
df = spark.sql("DESCRIBE DETAIL deltalake_catalog.default.orders_ext_01")
df.show(truncate=False)
```

```
display(dbutils.fs.ls('abfss://externaldata@ttmystorageaccount001.dfs.core.windows.net/orders'))
```

```
display(dbutils.fs.ls('abfss://externaldata@ttmystorageaccount001.dfs.core.windows.net/orders/_delta_log'))
```

```
%sql
SELECT * FROM
JSON.`abfss://externaldata@ttmystorageaccount001.dfs.core.windows.net/orders/_delta_log/00000000000000000005.json`
```

Pyspark:

```
df = spark.read.format("json") \

.load("abfss://externaldata@ttmystorageaccount001.dfs.core.windows.net/orders/_delta_log/00000000000000000005.json")

df.show(truncate=False)
```

```
%sql
DESCRIBE HISTORY deltalake_catalog.default.orders_ext_01;
```

Pyspark:

```
df = spark.sql("DESCRIBE HISTORY deltalake_catalog.default.orders_ext_01")
df.show(truncate=False)
```

```
%sql
DESCRIBE FORMATTED deltalake_catalog.default.orders_ext_01;
```

Pyspark:

```
df = spark.sql("DESCRIBE FORMATTED deltalake_catalog.default.orders_ext_01")
df.show(truncate=False)
```

```
%sql
INSERT OVERWRITE orders_ext_01 (order_id, sku, product_name, product_category, qty, unit_price)
VALUES (8, 'SKU-8001', 'Water Bottle', 'Fitness', 2, 399.00);
```

Pyspark:

```
spark.sql("""
INSERT OVERWRITE TABLE orders_ext_01 (order_id, sku, product_name, product_category, qty, unit_price)
VALUES (8, 'SKU-8001', 'Water Bottle', 'Fitness', 2, 399.00)
""")
```

```
%sql
select * from orders_ext_01
```

Pyspark:

df = spark.sql("SELECT * FROM orders_ext_01")
df.show(truncate=False)

```
%sql
DESCRIBE HISTORY deltalake_catalog.default.orders_ext_01;
%sql
SELECT * FROM
JSON.`abfss://externaldata@ttmystorageaccount001.dfs.core.windows.net/orders/_delta_log/00000000000000000002.json`
```

Pysaprk:

```
# Get Delta table transaction history
history_df = spark.sql("DESCRIBE HISTORY deltalake_catalog.default.orders_ext_01")
history_df.show(truncate=False)

# Read a specific Delta log JSON file from ADLS
delta_log_df = spark.read.format("json") \

.load("abfss://externaldata@ttmystorageaccount001.dfs.core.windows.net/orders/_delta_log/00000000000000000002.json")

delta_log_df.show(truncate=False)
```

```
%sql
drop table deltalake_catalog.default.orders_ext_01;
```

Pyspark:

```
spark.sql("DROP TABLE deltalake_catalog.default.orders_ext_01")
```

```
%sql
INSERT INTO orders_ext_01
(order_id, sku, product_name, product_category, qty, unit_price)
VALUES
(1, 'SKU-1001', 'Wireless Mouse', 'Electronics', 2, 799.00),
(2, 'SKU-2001', 'Yoga Mat', 'Fitness', 1, 1199.00),
(3, 'SKU-3001', 'Notebook A5', 'Stationery', 5, 49.50);
```

Pyspark:

```
spark.sql("""
INSERT INTO orders_ext_01
(order_id, sku, product_name, product_category, qty, unit_price)
VALUES
(1, 'SKU-1001', 'Wireless Mouse', 'Electronics', 2, 799.00),
(2, 'SKU-2001', 'Yoga Mat', 'Fitness', 1, 1199.00),
```

(3, 'SKU-3001', 'Notebook A5', 'Stationery', 5, 49.50)
""")

```
%sql
INSERT INTO orders_ext_01
(order_id, sku, product_name, product_category, qty, unit_price)
VALUES
(4, 'SKU-4001', 'Coffee Mug', 'Kitchen', 3, 299.00),
(5, 'SKU-5001', 'LED Bulb', 'Electronics', 4, 149.99);
```

Pyspark:
spark.sql("""
INSERT INTO orders_ext_01
(order_id, sku, product_name, product_category, qty, unit_price)
VALUES
(4, 'SKU-4001', 'Coffee Mug', 'Kitchen', 3, 299.00),
(5, 'SKU-5001', 'LED Bulb', 'Electronics', 4, 149.99)
""")

```
%sql
DESCRIBE HISTORY orders_ext_01;
```

Pyspark:
history_df = spark.sql("DESCRIBE HISTORY orders_ext_01")
history_df.show(truncate=False)

```
%sql
DELETE FROM orders_ext_01 WHERE order_id = 3;
```

Pyspark:
spark.sql("DELETE FROM orders_ext_01 WHERE order_id = 3")

```
%sql
DESCRIBE HISTORY orders_ext_01;
```

history_df = spark.sql("DESCRIBE HISTORY orders_ext_01")
history_df.show(truncate=False)

```
%sql
select * from orders_ext_01;
```

df = spark.sql("SELECT * FROM orders_ext_01")
df.show(truncate=False)

```sql
%sql
DROP TABLE orders_ext_01;
```

Pyspark:
spark.sql("DROP TABLE IF EXISTS orders_ext_01")

```sql
%sql
DELETE FROM orders_ext_01 WHERE order_id IN (1,5);
```

Pyspark:
spark.sql("DELETE FROM orders_ext_01 WHERE order_id IN (1,5)")

```sql
%sql
UPDATE orders_ext_01
SET unit_price = 52
WHERE order_id = 3;
```

Pyspark:
spark.sql("UPDATE orders_ext_01 SET unit_price = 52 WHERE order_id = 3")