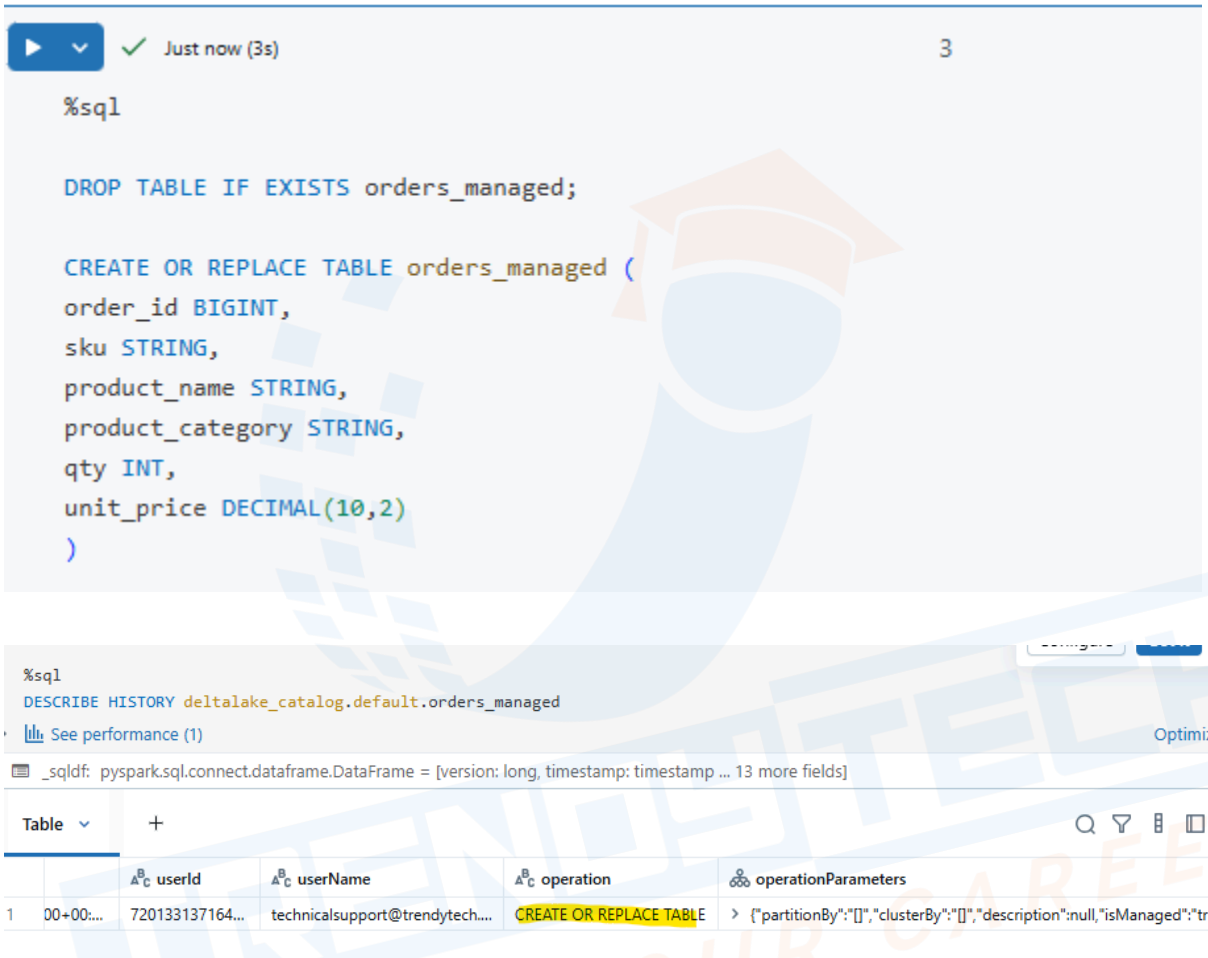
Create a new notebook deltalake-demo-4

Use the deltalake_catalog created in the last session

If you are using a free edition then you can proceed with creating a managed table.
(if you are using azure databricks then you follow the steps shown in the video)



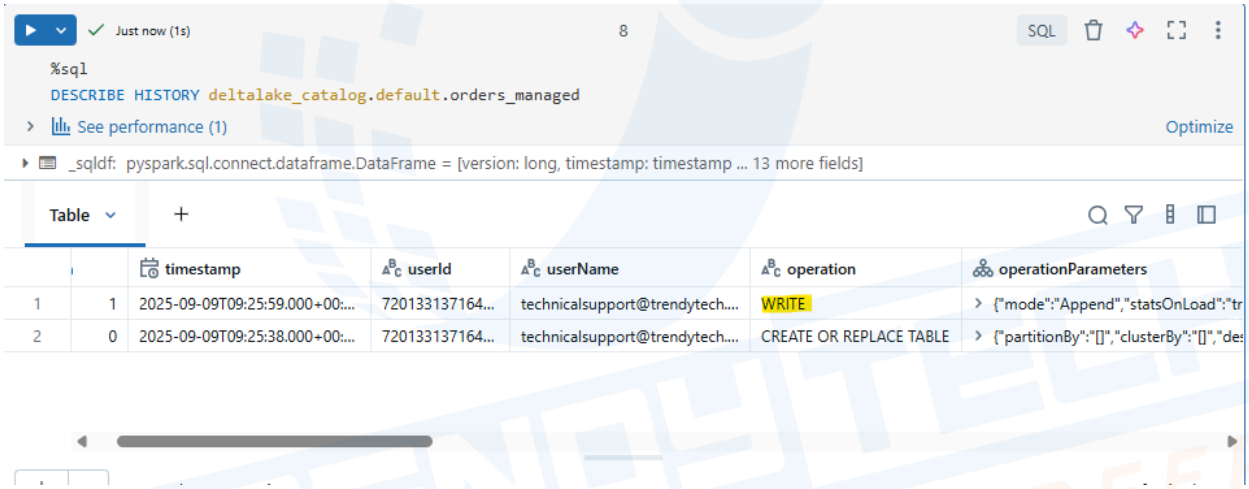
```
%sql
INSERT INTO orders_managed VALUES
  (1, 'SKU-1001', 'Wireless Mouse', 'Electronics', 2, 799.00),
  (2, 'SKU-2001', 'Yoga Mat', 'Fitness', 1, 1199.00),
  (3, 'SKU-3001', 'Notebook A5', 'Stationery', 5, 49.50),
  (4, 'SKU-4001', 'Coffee Mug', 'Kitchen', 3, 299.00),
  (5, 'SKU-5001', 'LED Bulb', 'Electronics', 4, 149.99);
```

See performance (1)

_sqldf: pyspark.sql.connect.dataframe.DataFrame = [num_affected_rows: long, num_inserted_rows: long]

Table +

	num_affected_rows	num_inserted_rows
1	5	5



But unlike Premium/Pro editions, free editions sometimes don't populate the Location field in DESCRIBE EXTENDED.

The table is still a managed Delta table, just that the UI/SQL output doesn’t show the path.

```
%sql
UPDATE deltalake_catalog.default.orders_managed
SET qty = 3
where order_id = 1;
```

See performance (1)

_sqldf: pyspark.sql.connect.dataframe.DataFrame = [num_affected_rows: long]

Table +

	num_affected_rows
1	1

Just now (2s) 12 SQL

```
%sql
DESCRIBE HISTORY deltalake_catalog.default.orders_managed
```

See performance (1) Optimiz

_sqldf: pyspark.sql.connect.dataframe.DataFrame = [version: long, timestamp: timestamp ... 13 more fields]

Table +

	version	timestamp	userId	userName	operation	operationParameter
1	3	2025-09-09T09:41:00.000+00:...	720133137164...	technicalsupport@trendytech....	OPTIMIZE	{"predicate":"[]","auto
2	2	2025-09-09T09:40:58.000+00:...	720133137164...	technicalsupport@trendytech....	UPDATE	{"predicate":"[\"(order
3	1	2025-09-09T09:25:59.000+00:...	720133137164...	technicalsupport@trendytech....	WRITE	{"mode":"Append","st
4	0	2025-09-09T09:25:38.000+00:...	720133137164...	technicalsupport@trendytech....	CREATE OR REPLACE TABLE	{"partitionBy":"[]","clu

Just now (2s) 14 SQL

```
%sql
INSERT OVERWRITE TABLE orders_managed VALUES
    (1, 'SKU-1001', 'Wireless Mouse', 'Electronics', 3, 799.00),
    (3, 'SKU-3001', 'Notebook A5',    'Stationery',  5, 49.50),
    (5, 'SKU-5001', 'LED Bulb',       'Electronics', 4, 149.99);
```

See performance (1)

_sqldf: pyspark.sql.connect.dataframe.DataFrame = [num_affected_rows: long, num_inserted_rows: long]

Table +

	num_affected_rows	num_inserted_rows
1	3	3

TRENDYTECH UPLIFT YOUR CAREER!

Just now (2s) 16 SQL

```
%sql
SELECT DISTINCT _metadata.file_path
FROM orders_managed;
```

See performance (1)

_sqldf: pyspark.sql.connect.dataframe.DataFrame = [file_path: string]

Table +

	file_path
1	s3://dbstorage-prod-uj0ub/uc/b1f08865-0a64-468d-9e44-5c2b3310926f/5f864d67-9999-41f0-8c1d-da085226bcfc/__unitystorage/ca...

Just now (1s) 17 SQL

```
%sql
describe history orders_managed;
```

See performance (1) Optimize

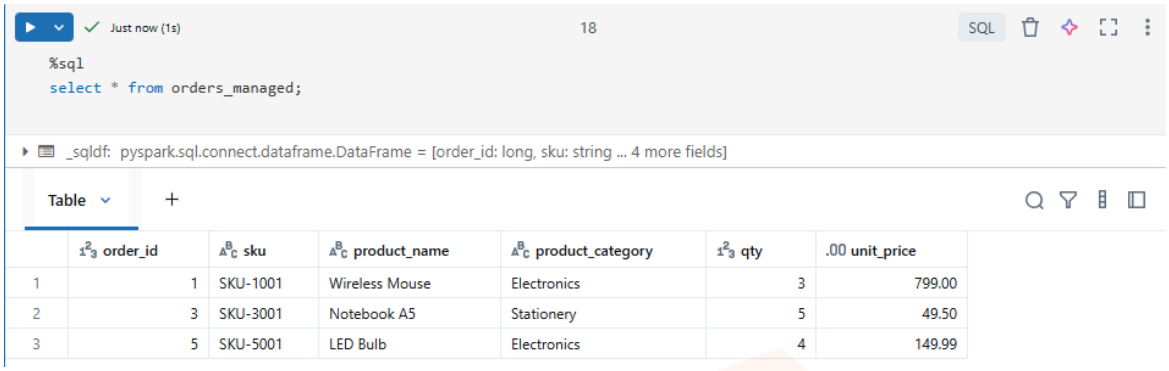
_sqldf: pyspark.sql.connect.dataframe.DataFrame = [version: long, timestamp: timestamp ... 13 more fields]

Table +

	version	timestamp	userId	userName	operation	operationParameter
1	4	2025-09-09T09:49:17.000+00:...	720133137164...	technicalsupport@trendytech....	WRITE	{"mode":"Overwrite",
2	3	2025-09-09T09:41:00.000+00:...	720133137164...	technicalsupport@trendytech....	OPTIMIZE	{"predicate":"[]","autc
3	2	2025-09-09T09:40:58.000+00:...	720133137164...	technicalsupport@trendytech....	UPDATE	{"predicate":"[\"(orde
4	1	2025-09-09T09:25:59.000+00:...	720133137164...	technicalsupport@trendytech....	WRITE	{"mode":"Append","s
5	0	2025-09-09T09:25:38.000+00:	720133137164	technicalsupport@trendytech	CREATE OR REPLACE TABLE	{"partitionBy":"[]","clu

TRENDYTECH UPLIFT YOUR CAREER!

Delta lake session 2:

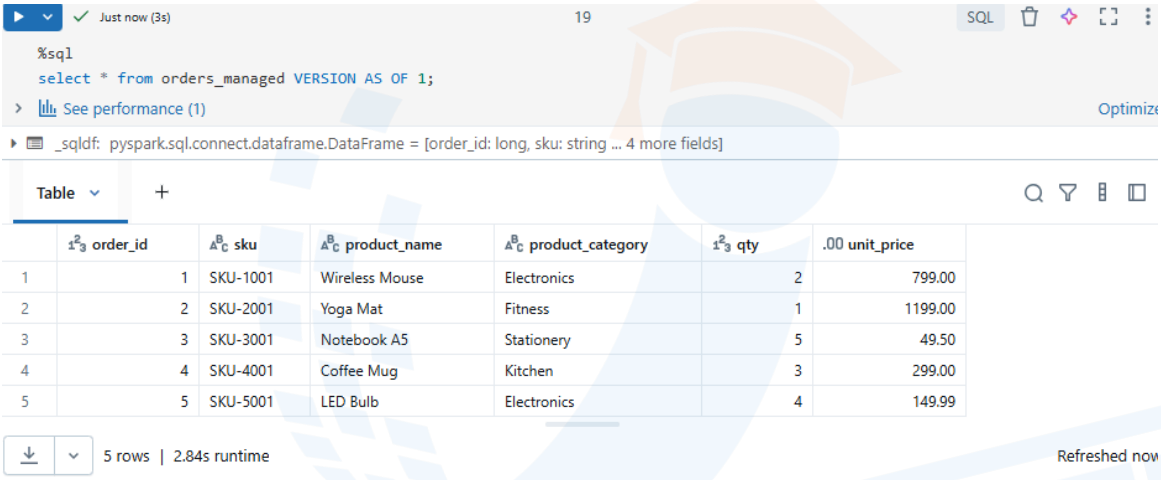


```
%sql
select * from orders_managed;
```

_sqldf: 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	1	SKU-1001	Wireless Mouse	Electronics	3	799.00
2	3	SKU-3001	Notebook A5	Stationery	5	49.50
3	5	SKU-5001	LED Bulb	Electronics	4	149.99



```
%sql
select * from orders_managed VERSION AS OF 1;
```

See performance (1)

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

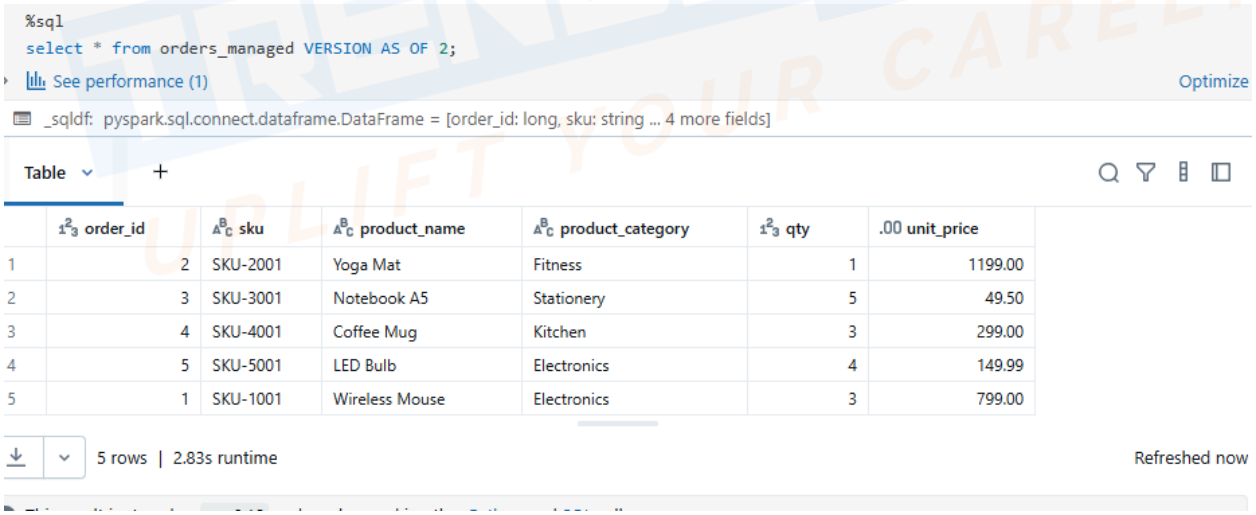
Table

	order_id	sku	product_name	product_category	qty	unit_price
1	1	SKU-1001	Wireless Mouse	Electronics	2	799.00
2	2	SKU-2001	Yoga Mat	Fitness	1	1199.00
3	3	SKU-3001	Notebook A5	Stationery	5	49.50
4	4	SKU-4001	Coffee Mug	Kitchen	3	299.00
5	5	SKU-5001	LED Bulb	Electronics	4	149.99

5 rows | 2.84s runtime

Refreshed now

Python equivalent code for above sql code:
orders_df = spark.sql("SELECT * FROM orders_managed VERSION AS OF 2")
orders_df.show()



```
%sql
select * from orders_managed VERSION AS OF 2;
```

See performance (1)

Optimize

_sqldf: 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	2	SKU-2001	Yoga Mat	Fitness	1	1199.00
2	3	SKU-3001	Notebook A5	Stationery	5	49.50
3	4	SKU-4001	Coffee Mug	Kitchen	3	299.00
4	5	SKU-5001	LED Bulb	Electronics	4	149.99
5	1	SKU-1001	Wireless Mouse	Electronics	3	799.00

5 rows | 2.83s runtime

Refreshed now

```
%sql
select * from orders_managed@v2;
```

See performance (1)

_sqldf: 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	2	SKU-2001	Yoga Mat	Fitness	1	1199.00
2	3	SKU-3001	Notebook A5	Stationery	5	49.50
3	4	SKU-4001	Coffee Mug	Kitchen	3	299.00
4	5	SKU-5001	LED Bulb	Electronics	4	149.99
5	1	SKU-1001	Wireless Mouse	Electronics	3	799.00

5 rows | 2.55s runtime

```
orders_df = spark.sql("SELECT * FROM orders_managed VERSION AS OF 2")
orders_df.show()
```

```
%sql
select * from orders_managed timestamp as of '2025-09-09T9:35:55.000+00:00';
```

See performance (1)

_sqldf: 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	1	SKU-1001	Wireless Mouse	Electronics	2	799.00
2	2	SKU-2001	Yoga Mat	Fitness	1	1199.00
3	3	SKU-3001	Notebook A5	Stationery	5	49.50
4	4	SKU-4001	Coffee Mug	Kitchen	3	299.00
5	5	SKU-5001	LED Bulb	Electronics	4	149.99

5 rows | 2.66s runtime

Refresh

```
orders_df = spark.sql("""
  SELECT * FROM orders_managed TIMESTAMP AS OF '2025-09-09T09:35:55.000+00:00'
""")
orders_df.show()
```

UPLIFT YOUR CAREER!

Just now (1s) 22

```
df_latest = spark.read.table("orders_managed")
df_latest.show()
```

See performance (1)

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

```
+--------+--------+--------------+----------------+---+----------+
|order_id|     sku|  product_name|product_category|qty|unit_price|
+--------+--------+--------------+----------------+---+----------+
|       1|SKU-1001|Wireless Mouse|     Electronics|  3|    799.00|
|       3|SKU-3001|   Notebook A5|      Stationery|  5|     49.50|
|       5|SKU-5001|      LED Bulb|     Electronics|  4|    149.99|
+--------+--------+--------------+----------------+---+----------+
```

Just now (1s) 23

```
df_v2 = spark.read.option("versionAsOf",2).table("orders_managed")
df_v2.show()
```

df_v2: pyspark.sql.connect.dataframe.DataFrame = [order_id: long, sku: string ... 4 more fields]

```
+--------+--------+--------------+----------------+---+----------+
|order_id|     sku|  product_name|product_category|qty|unit_price|
+--------+--------+--------------+----------------+---+----------+
|       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|
|       1|SKU-1001|Wireless Mouse|     Electronics|  3|    799.00|
+--------+--------+--------------+----------------+---+----------+
```

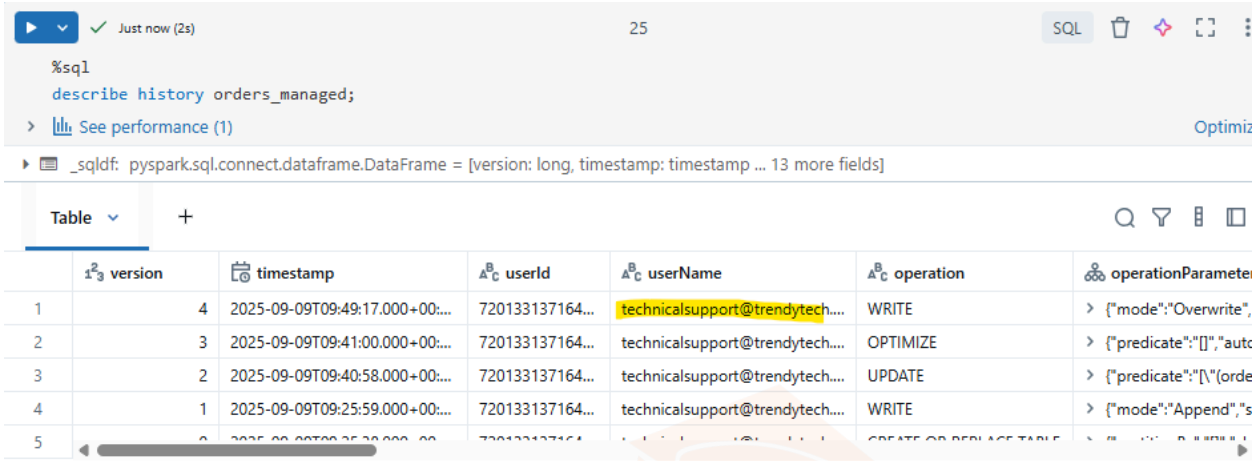
Just now (1s) 24

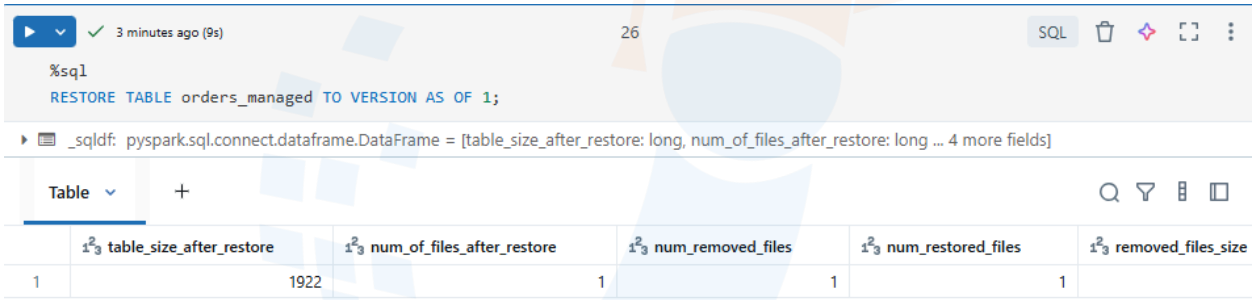
```
df_v2 = spark.read.format("delta") \
.option("timestampAsOf",'2025-09-09T9:35:55.000+00:00') \
.table("orders_managed")

df_v2.show()
```

df_v2: pyspark.sql.connect.dataframe.DataFrame = [order_id: long, sku: string ... 4 more fields]

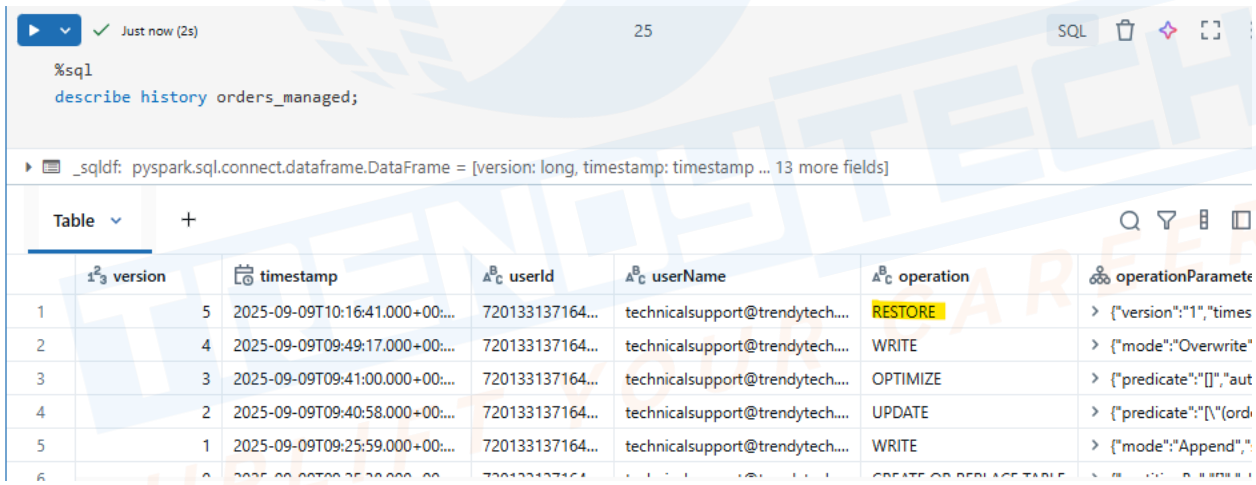
```
+--------+--------+--------------+----------------+---+----------+
|order_id|     sku|  product_name|product_category|qty|unit_price|
+--------+--------+--------------+----------------+---+----------+
|       1|SKU-1001|Wireless Mouse|     Electronics|  2|    799.00|
|       2|SKU-2001|      Yoga 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|
+--------+--------+--------------+----------------+---+----------+
```

UPLIFT YOUR CAREER!

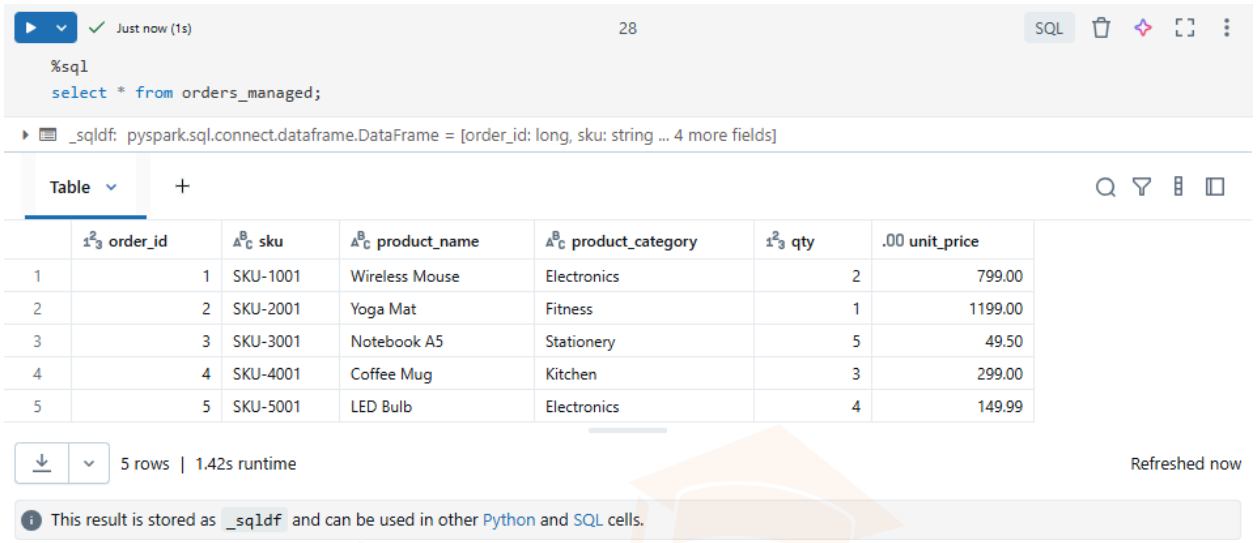




spark.sql("RESTORE TABLE orders_managed TO VERSION AS OF 1")



Since we have restored to old version, we should see 5 records -



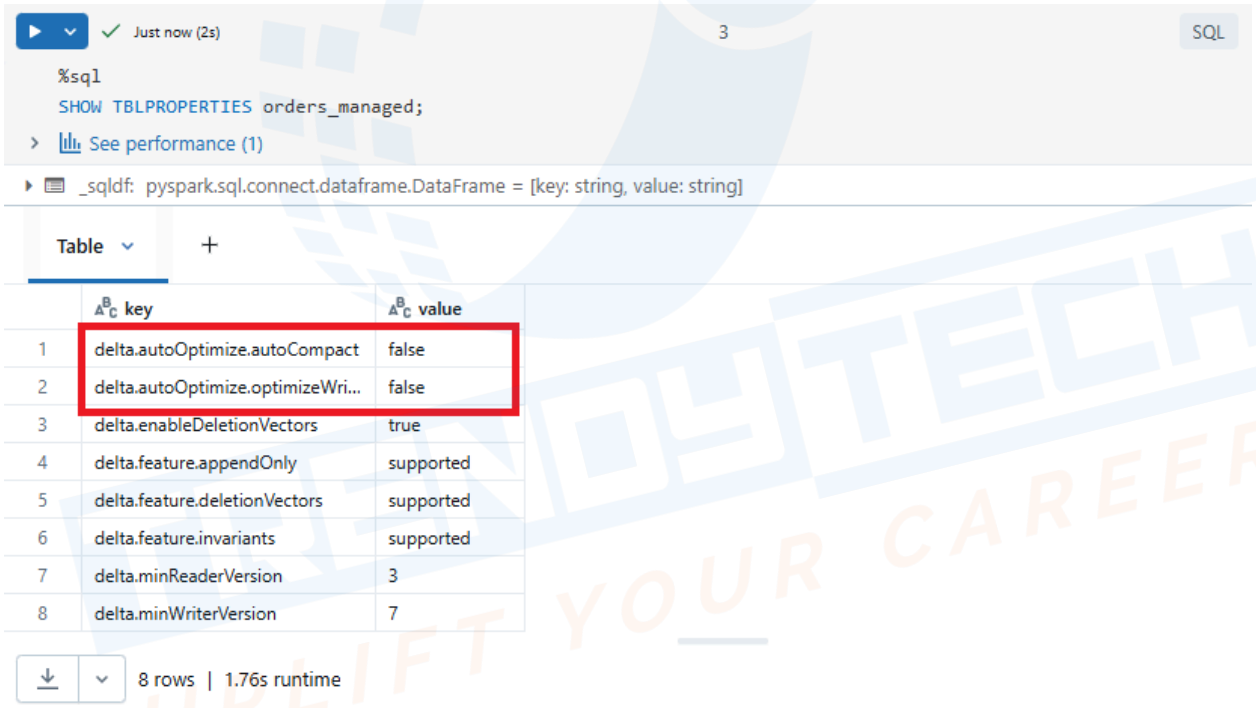
Delta lake compute session 3:

VACUUM works in Free Edition, but you can only remove files older than 7 days — retention cannot be shortened

```
%sql
use catalog deltalake_catalog;
```

Pyspark equivalent code for above sql code:
# Drop table if it exists
spark.sql("DROP TABLE IF EXISTS orders_managed")

```
# Create or replace table with schema and TBLPROPERTIES
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)
)
TBLPROPERTIES (
  delta.autoOptimize.optimizeWrite = false,
  delta.autoOptimize.autoCompact = false
)
""")
```




```
props_df = spark.sql("SHOW TBLPROPERTIES orders_managed")
props_df.show(truncate=False)
```

Just now (6s) 4 SQL

```
%sql
INSERT INTO orders_managed 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);
```

See performance (1)

_sqldf: pyspark.sql.connect.dataframe.DataFrame = [num_affected_rows: long, num_inserted_rows: long]

Table +

	num_affected_rows	num_inserted_rows
1	5	5

Just now (3s) 5 SQL

```
%sql
DESCRIBE HISTORY deltalake_catalog.default.orders_managed
```

See performance (1) Optimize

_sqldf: pyspark.sql.connect.dataframe.DataFrame = [version: long, timestamp: timestamp ... 13 more fields]

Table +

	version	timestamp	userId	userName	operation	operationParameters
1	1	2025-09-11T08:48:50.000+00:...	720133137164...	technicalsupport@trendytech....	WRITE	{"mode":"Append","stat
2	0	2025-09-11T08:40:55.000+00:...	720133137164...	technicalsupport@trendytech....	CREATE OR REPLACE TABLE	{"partitionBy":"[]","cluste

Just now (2s) 6

```
%sql
describe detail deltalake_catalog.default.orders_managed
```

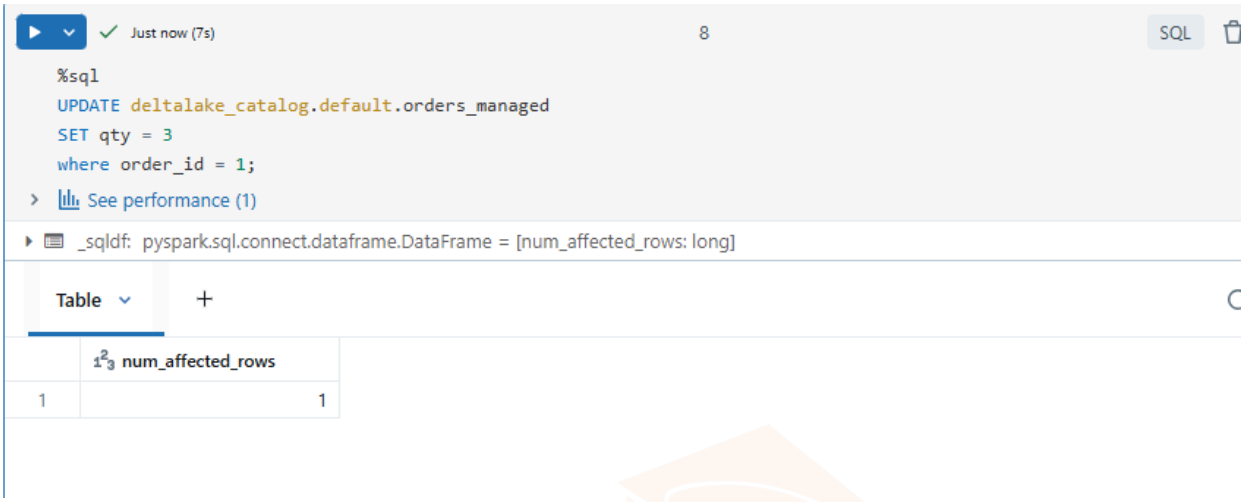
See performance (1) Optimize

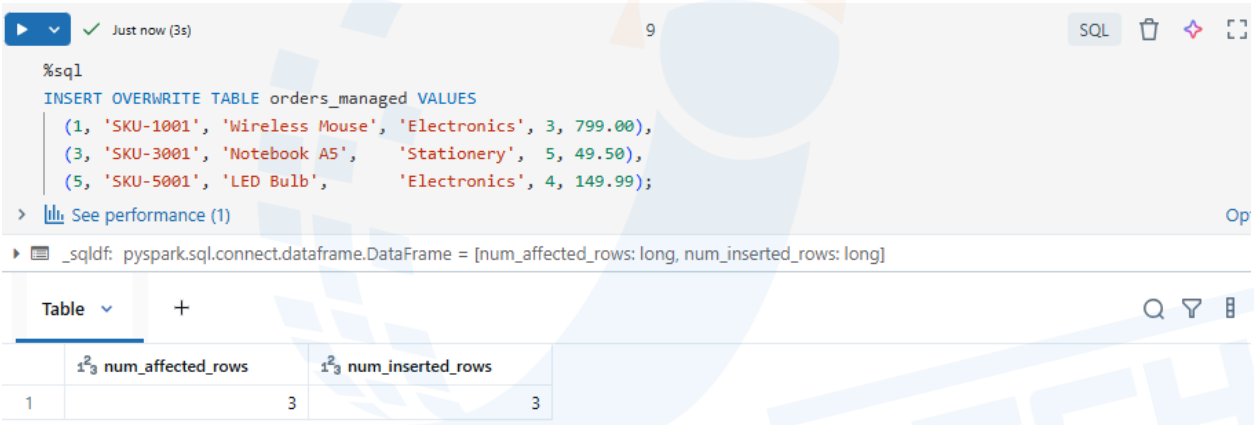
_sqldf: pyspark.sql.connect.dataframe.DataFrame = [format: string, id: string ... 15 more fields]

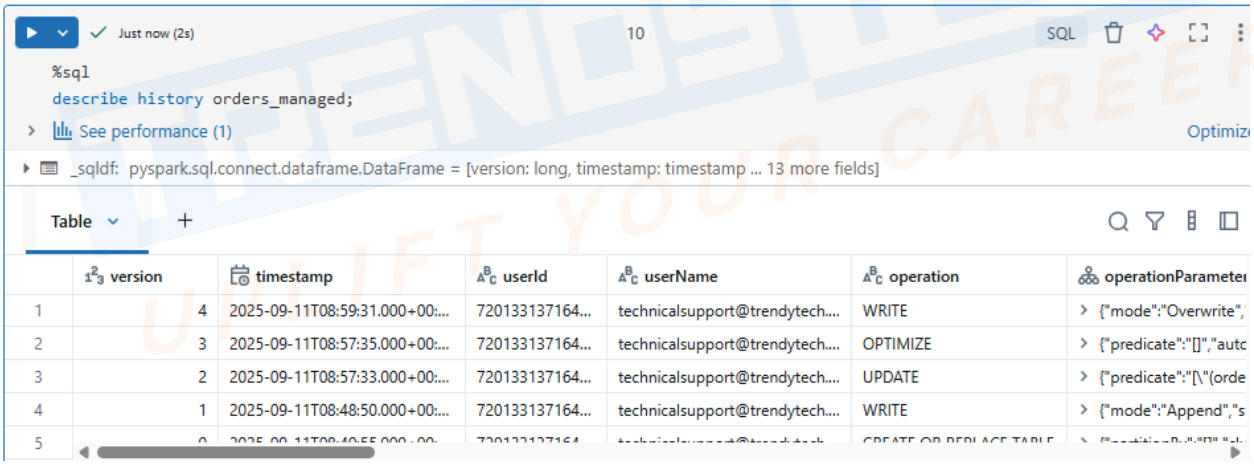
Table +

	format	id	name	description	location	created
1	delta	fb3ec147-5128-4bae-a1d4-4e103e84e5...	deltalake_catalog.default.orders_manag...	null		2025-09-11

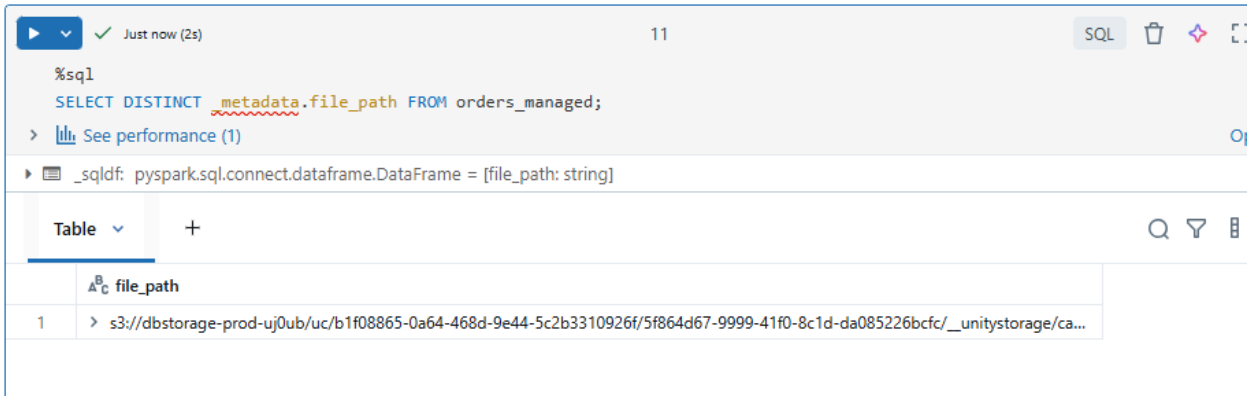
TRENDYTECH UPLIFT YOUR CAREER!



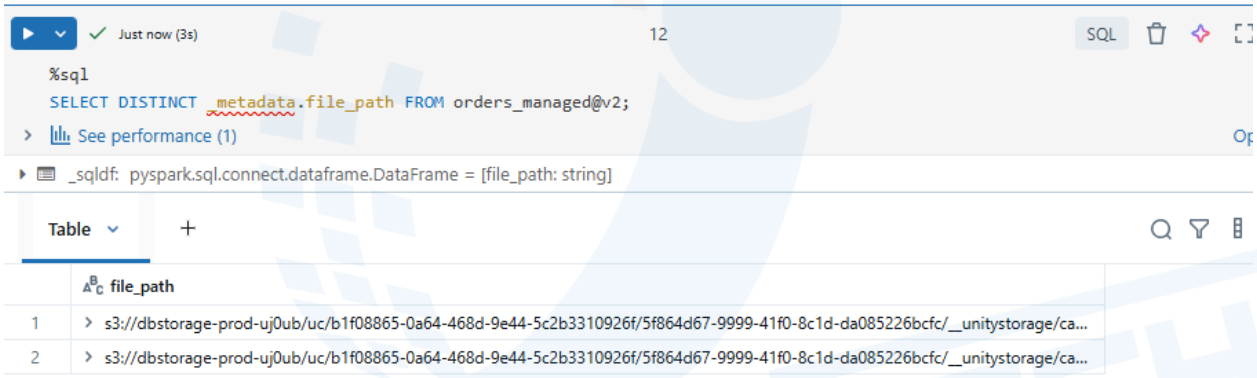




You still see OPTIMIZE in the history because on serverless clusters, Databricks runs background compaction jobs automatically for performance, even if Auto Optimize is disabled

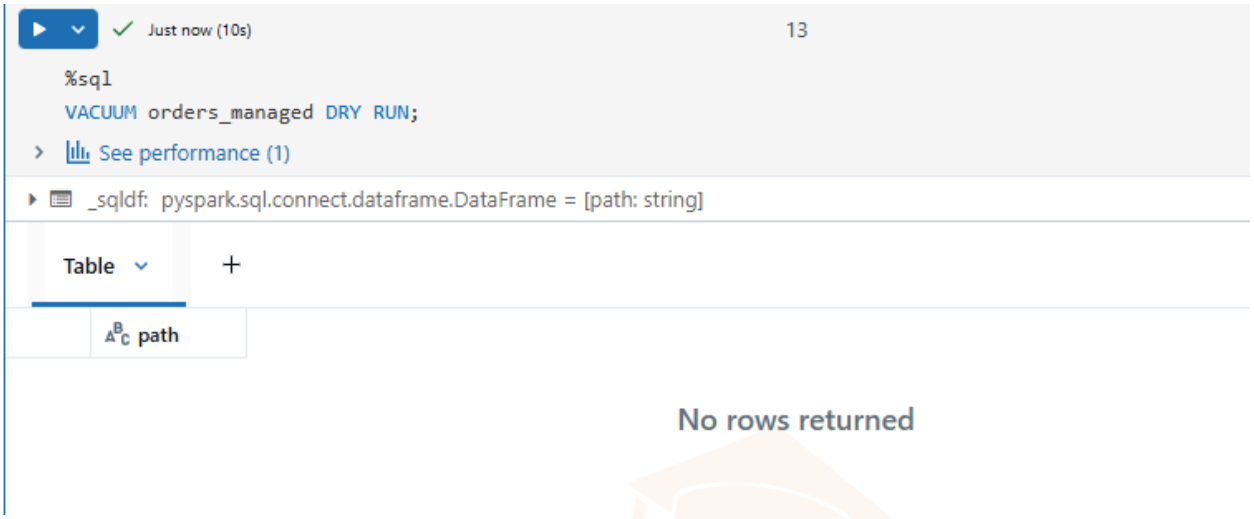



```
file_paths = spark.read.table("orders_managed") \
    .selectExpr("DISTINCT _metadata.file_path")

file_paths.show(truncate=False)
```

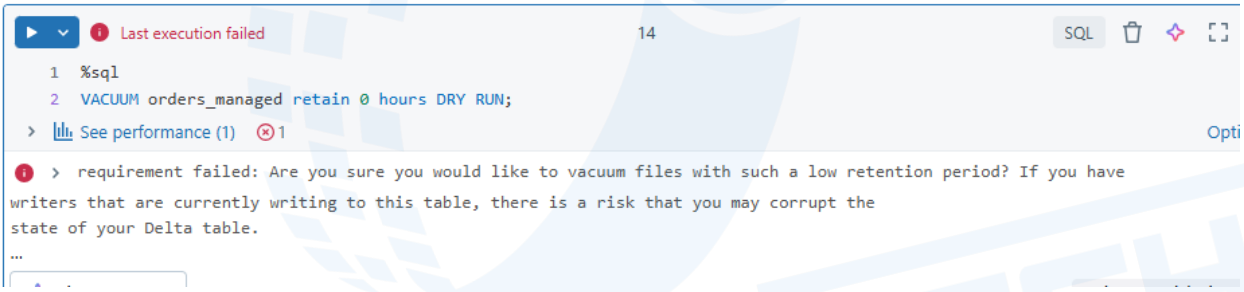


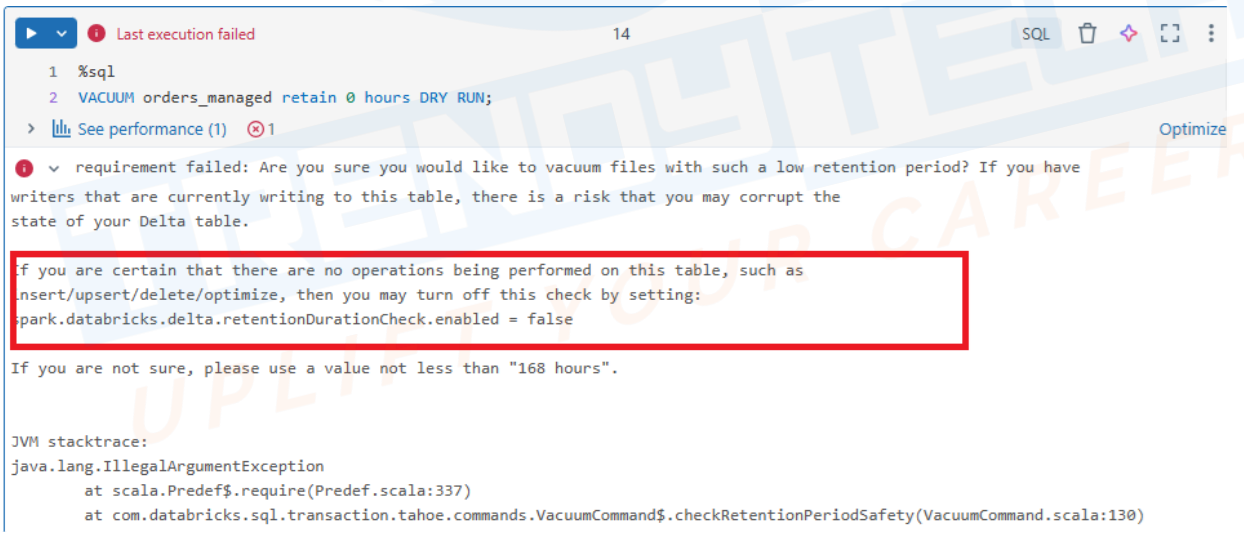

```
file_paths_df = spark.sql("""
SELECT DISTINCT _metadata.file_path
FROM orders_managed VERSION AS OF 2
""")

file_paths_df.show(truncate=False)
```

TRENDYTECH
UPLIFT YOUR CAREER!



because there are no files created before 7 days
Pyspark equivalent code for above sql code:
vacuum_result = spark.sql("VACUUM orders_managed DRY RUN")
vacuum_result.show(truncate=False)





Pyspark equivalent code for above sql code:
vacuum_result = spark.sql("VACUUM orders_managed RETAIN 0 HOURS DRY RUN")
vacuum_result.show(truncate=False)

Free edition of Databricks doesn't support changing certain Delta table configurations like spark.databricks.delta.retentionDurationCheck.enabled.

VACUUM works in Free Edition, but you can only remove files older than 7 days — retention cannot be shortened

Delta lake compute session 4: (need azure account)

```
%sql
use catalog deltalake_catalog;
```

```
%sql
DROP TABLE IF EXISTS orders_ext_01;

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)
)
LOCATION
'abfss://externaldata@ttmystorageaccount001.dfs.core.windows.net/orders'
```

```
for i in range(100):   # 100 small commits — adjust smaller/larger as needed
    order_id = 1000 + i
    sku = f"SKU-BURST-{i}"
    name = f"Burst-{i}"
    category = "Demo"
    qty = 1
```

```
    price = float(i % 10 + 1) * 1.0
    sql = f"""
    INSERT INTO orders_ext_01 VALUES ({order_id}, '{sku}', '{name}',
'{category}', {qty}, {price});
    """
    spark.sql(sql)
```

Above code is already in python

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

```
%sql
SELECT * FROM
PARQUET.`abfss://externaldata@ttmystorageaccount001.dfs.core.windows.net/orders
/_delta_log/00000000000000000072.checkpoint.parquet`
```

Pyspark:
checkpoint_path =
"abfss://externaldata@ttmystorageaccount001.dfs.core.windows.net/orders/_delta_log/00000000000000000072.checkpoint.parquet"

checkpoint_df = spark.read.format("parquet").load(checkpoint_path)
checkpoint_df.show(truncate=False)

```
%sql
SELECT * FROM
JSON.`abfss://externaldata@ttmystorageaccount001.dfs.core.windows.net/orders/_d
elta_log/00000000000000000073.00000000000000000078.compacted.json`
```

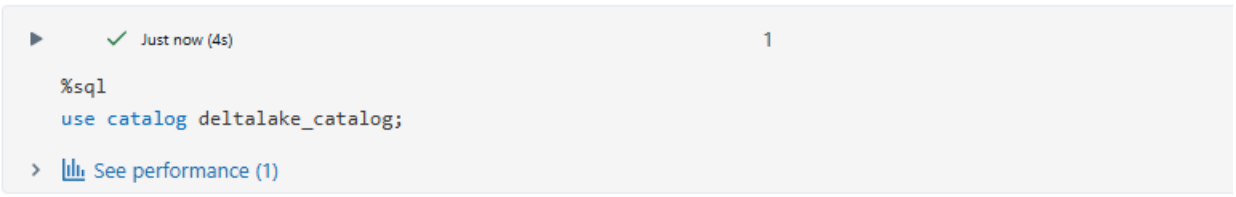
Pyspark:
compacted_path =
"abfss://externaldata@ttmystorageaccount001.dfs.core.windows.net/orders/_delta_log/00000000000000000073.00000000000000000078.compacted.json"

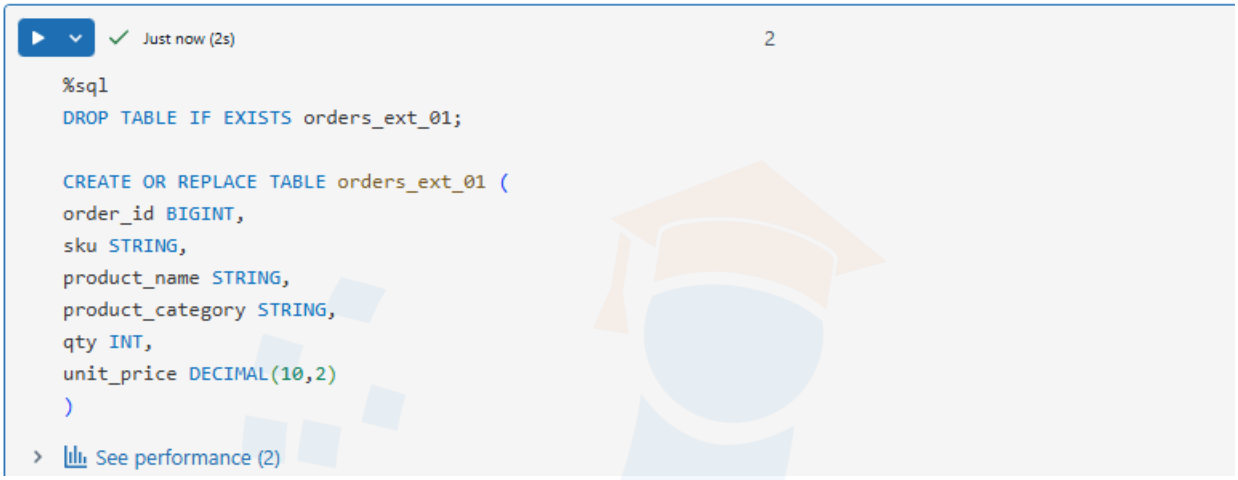
compacted_df = spark.read.json(compacted_path)
compacted_df.show(truncate=False)

Session 5 - can follow steps followed by sir in the video
Session 6 - need azure account
Session 7 - can follow sir's video

Session 8:

You can create managed table using free edition -






```
%sql
INSERT INTO orders_ext_01 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 FORMATTED orders_ext_01
```

TRENDTECH UPLIFT YOUR CAREER!

```
%sql
ALTER TABLE orders_ext_01
ALTER COLUMN order_id SET NOT NULL;
```

See performance (1) Optimize

Just now (2s) 6 SQL

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

See performance (1) Optimize

_sqldf: pyspark.sql.connect.dataframe.DataFrame = [version: long, timestamp: timestamp ... 13 more fields]

Table +

	version	timestamp	userId	userName	operation	operationParameters
1	3	2025-09-16T10:42:18.000+00:...	720133137164...	technicalsupport@trendytech....	CHANGE COLUMN	{"column":"{\"name\":\"
2	2	2025-09-16T10:41:12.000+00:...	720133137164...	technicalsupport@trendytech....	WRITE	{"mode":"Append","sta
3	1	2025-09-16T10:40:54.000+00:...	720133137164...	technicalsupport@trendytech....	WRITE	{"mode":"Append","sta
4	0	2025-09-16T10:40:08.000+00:...	720133137164...	technicalsupport@trendytech....	CREATE OR REPLACE TABLE	{"partitionBy":"[]","clust

```
%sql
ALTER TABLE orders_ext_01
ADD CONSTRAINT chk_qty_price CHECK (qty > 0 AND unit_price > 0)
```

See performance (1) Optimize

Just now (2s) 8 SQL

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

See performance (1) Optimize

_sqldf: pyspark.sql.connect.dataframe.DataFrame = [version: long, timestamp: timestamp ... 13 more fields]

Table +

	version	timestamp	userId	userName	operation	operationParameter
1	4	2025-09-17T08:25:22.000+00:...	720133137164...	technicalsupport@trendytech....	ADD CONSTRAINT	{"name":"chk_qty_pri
2	3	2025-09-16T10:42:18.000+00:...	720133137164...	technicalsupport@trendytech....	CHANGE COLUMN	{"column":"{\"name\"
3	2	2025-09-16T10:41:12.000+00:...	720133137164...	technicalsupport@trendytech....	WRITE	{"mode":"Append","s
4	1	2025-09-16T10:40:54.000+00:...	720133137164...	technicalsupport@trendytech....	WRITE	{"mode":"Append","s

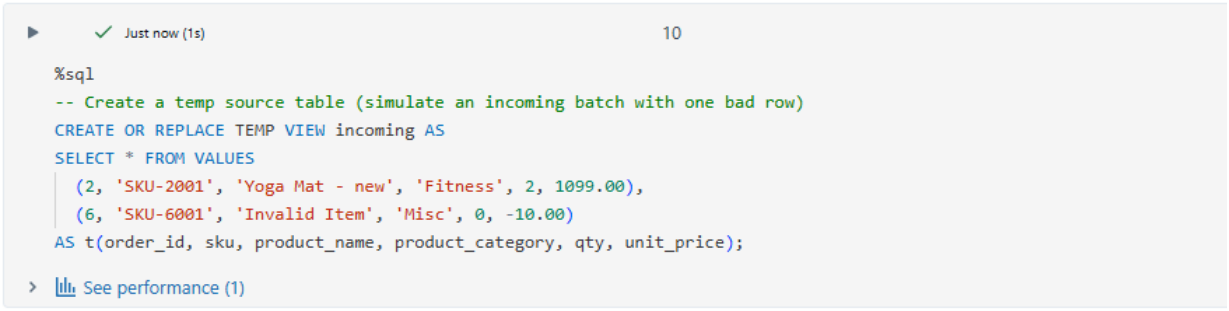
Last execution failed 9 SQL

```
1 %sql
2 INSERT INTO orders_ext_01 VALUES (6, 'SKU-6001', 'Faulty Item', 'Misc', 0, -10.00);
```

See performance (1) 1 Optimize

```
[DELTA_VIOLATE_CONSTRAINT_WITH_VALUES] CHECK constraint chk_qty_price ((qty > 0) AND (unit_price > 0)) violated by row with values:
- qty : 0
- unit_price : -10.00 SQLSTATE: 23001
```

Diagnose error Assistant Quick Fix: ON




```
%sql
select sum(qty) as total_qty from orders_ext_01;
update orders_ext_01 set qty = qty + 10 where order_id = 1
```

Session 9:

Sir has used serverless, you can follow the same steps.

Python equivalent code for the sql in session 9 are given below -

```
%sql
DROP TABLE IF EXISTS orders_ext_01;

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)
)
```

Pyspark:

```
# Drop table if it exists
spark.sql("DROP TABLE IF EXISTS orders_ext_01")

# Create or replace table with schema
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)
)
""")
```

```
%sql
INSERT INTO orders_ext_01 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 orders_ext_01 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 FORMATTED orders_ext_01
```

Pyspark:

```
desc_df = spark.sql("DESCRIBE FORMATTED orders_ext_01")
desc_df.show(truncate=False)
```

```
%sql
ALTER TABLE orders_ext_01
ALTER COLUMN order_id SET NOT NULL;
```

Pyspark:

spark.sql("""
ALTER TABLE orders_ext_01
ALTER COLUMN order_id SET NOT NULL
""")

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

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

```
%sql
ALTER TABLE orders_ext_01
ADD CONSTRAINT chk_qty_price CHECK (qty > 0 AND unit_price > 0)
```

Pyspark:
spark.sql("""
ALTER TABLE orders_ext_01
ADD CONSTRAINT chk_qty_price CHECK (qty > 0 AND unit_price > 0)
""")

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

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

```
%sql
INSERT INTO orders_ext_01 VALUES (6, 'SKU-6001', 'Faulty Item', 'Misc', 0,
-10.00);
```

Pyspark:
spark.sql("""
    INSERT INTO orders_ext_01 VALUES
    (6, 'SKU-6001', 'Faulty Item', 'Misc', 0, -10.00)
  """)

```
%sql
-- Create a temp source table (simulate an incoming batch with one bad row)
CREATE OR REPLACE TEMP VIEW incoming AS
SELECT * FROM VALUES
  (2, 'SKU-2001', 'Yoga Mat - new', 'Fitness', 2, 1099.00),
  (6, 'SKU-6001', 'Invalid Item', 'Misc', 0, -10.00)
AS t(order_id, sku, product_name, product_category, qty, unit_price);
```

Pyspark:
from pyspark.sql import Row

```
# Create DataFrame directly
incoming_data = [
    Row(2, "SKU-2001", "Yoga Mat - new", "Fitness", 2, 1099.00),
    Row(6, "SKU-6001", "Invalid Item", "Misc", 0, -10.00)        ]

incoming_df = spark.createDataFrame(incoming_data,
    ["order_id", "sku", "product_name", "product_category", "qty", "unit_price"])

# Register as temp view (so you can use it in SQL)
incoming_df.createOrReplaceTempView("incoming")

# Quick check
spark.sql("SELECT * FROM incoming").show()
```

```
%sql
MERGE INTO orders_ext_01 AS TARGET
USING incoming AS src
ON TARGET.order_id = src.order_id
WHEN MATCHED THEN UPDATE SET *
WHEN NOT MATCHED THEN INSERT *;
```

Pyspark:

```
# Filter only valid rows
valid_source = spark.table("incoming").filter("qty > 0 AND unit_price > 0")

# Perform merge
target = DeltaTable.forName(spark, "orders_ext_01")

(
   target.alias("TARGET")
   .merge(
      valid_source.alias("src"),
      "TARGET.order_id = src.order_id"
   )
   .whenMatchedUpdateAll()
   .whenNotMatchedInsertAll()
   .execute()
)
```

```
%sql
select sum(qty) as total_qty from orders_ext_01;
update orders_ext_01 set qty = qty + 10 where order_id = 1
```

Pyspark:

```
spark.sql("""
UPDATE orders_ext_01
SET qty = qty + 10
```

WHERE order_id = 1
""")

Session 10:

Sir has used serverless, you can follow the same steps.

Python equivalent code for the sql in session 10 are given below -

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

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

schema = StructType([
    StructField("order_id", LongType(), False),
    StructField("sku", StringType(), True),
    StructField("product_name", StringType(), True),
    StructField("product_category", StringType(), True),
    StructField("qty", IntegerType(), True),
    StructField("unit_price", DecimalType(10,2), True)
])

df = spark.createDataFrame(data, schema=schema)

display(df)
df.printSchema()
```

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

Is already in pyspark format

```
%sql
```

```
SELECT * FROM
DELTA.`/Volumes/deltalake_catalog/default/delta_volume1/ordersdata1`
```

Pyspark:

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

```
%sql
DESCRIBE HISTORY
DELTA.`/Volumes/deltalake_catalog/default/delta_volume1/ordersdata1`
```

Pyspark:

```
history_df = spark.sql("""
DESCRIBE HISTORY DELTA.`/Volumes/deltalake_catalog/default/delta_volume1/ordersdata1`
""")

history_df.show(truncate=False)
```

```
%sql
ALTER TABLE
DELTA.`/Volumes/deltalake_catalog/default/delta_volume1/ordersdata1`
RENAME COLUMN qty to quantity;
```

Pyspark:

```
spark.sql("""
ALTER TABLE DELTA.`/Volumes/deltalake_catalog/default/delta_volume1/ordersdata1`
RENAME COLUMN qty TO quantity
""")
```

```
%sql
ALTER TABLE
DELTA.`/Volumes/deltalake_catalog/default/delta_volume1/ordersdata1`
SET TBLPROPERTIES ('delta.columnMapping.mode' = 'name')
```

Pyspark:

```
spark.sql("""
ALTER TABLE DELTA.`/Volumes/deltalake_catalog/default/delta_volume1/ordersdata1`
SET TBLPROPERTIES ('delta.columnMapping.mode' = 'name')
""")
```

```
%sql
describe formatted
DELTA.`/Volumes/deltalake_catalog/default/delta_volume1/ordersdata1`
```

Pyspark:

```
desc_df = spark.sql("""
```

DESCRIBE FORMATTED DELTA.`/Volumes/deltalake_catalog/default/delta_volume1/ordersdata1`
""")
desc_df.show(truncate=False)

```
%sql
ALTER TABLE
DELTA.`/Volumes/deltalake_catalog/default/delta_volume1/ordersdata1`
DROP COLUMN product_category;
```

Pyspark:
spark.sql("""
ALTER TABLE DELTA.`/Volumes/deltalake_catalog/default/delta_volume1/ordersdata1`
DROP COLUMN product_category
""")