

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

The screenshot shows the Databricks workspace interface. On the left, there is a code editor with two cells. Cell 1 is selected and contains the text 'Python'. Below it, cell 2 is visible. At the bottom of the code editor, a terminal prompt is shown: 'at 0x7f9c13ad9730>'. On the right, the 'Environment' configuration pane is open. It includes a 'Send feedback' button, a 'Memory' dropdown set to 'Standard (16GB)', a 'Serverless budget policy' dropdown set to 'None', a 'Base environment' text input field containing '/path/to/base_environment.y...', and an 'Environment version' dropdown set to '3'. Below these, there is a 'Dependencies' section with 'Added' and 'Available' tabs. A yellow callout box with the word 'Environment' and a gear icon points to the 'Base environment' field.

To create budget policy -

User → settings → compute → policies

The screenshot shows the Databricks 'Settings' page. The left sidebar contains a list of settings categories: 'Workspace admin', 'Appearance', 'Identity and access', 'Security', 'Compute' (highlighted in yellow), 'Development', 'Notifications', and 'Advanced'. Below these, there is a 'User' section with 'Profile', 'Preferences', 'Developer', 'Linked accounts', and 'Notifications'. The main content area is divided into three sections: 'SQL warehouses and serverless compute' with a 'Manage' button, 'Environments' with a 'Manage' button, and 'Policies'. The 'Policies' section is further divided into 'Serverless budget policies' (highlighted in yellow) and 'Default package repositories'. The 'Serverless budget policies' section contains a description: 'Serverless budget policies allow administrators to enforce certain cost attribution tags for serverless compute in notebooks, workflows, and DLT pipelines. Each policy can be assigned to users, groups, or service principals in order to enforce tagging requirements. [Learn more](#)'. A yellow callout box with the word 'Manage' and a gear icon points to the 'Manage' button in the 'Serverless budget policies' section.

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) X

Tags

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

Tag

Value

project

databricks_project

topic

compute

+ Add tag

Environment

Memory

Serverless budget policy

Base environment

Environment version

Dependencies

Standard (16GB)

demo-budget-policy

databricks_project compute

/path/to/base_environment.y...

3

Added

simplejson==3.19.*

Send feedback

X

Installing dependencies:

Dependencies ⓘ

Added Available

⋮

[+ Add dependency](#) ⓘ

Click 'enter' after entering the name and then click on apply

To Schedule the notebook:

Python ▾ Tabs: ON ▾ ☆ Last edit was 1 minute ago ⌵ ▶ Run all ● Connected ▾ Schedule

New schedule

Job name*

Simple Advanced

Schedule

Every ▾ ▾

Compute* ⓘ

Autoscaling ▾

Performance optimization ⓘ

☐ Performance optimized

More options ▾

Cancel Create

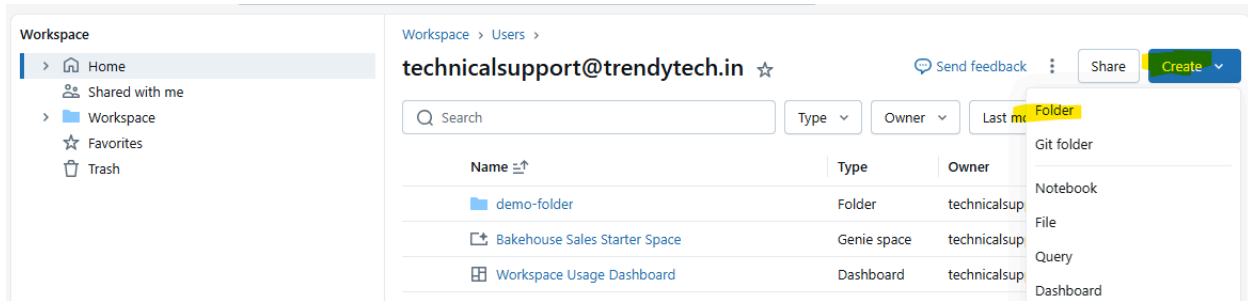
SQL warehouses:

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

The screenshot shows the Databricks SQL interface. The query executed is `DESCRIBE DETAIL orders_managed;`. The result is a table with 5 columns: format, id, name, description, and location. The name column contains the value `workspace.default.orders_managed`.

	format	id	name	description	location
1	delta	d4cc0501-28d2-49ea-a5c2-f143880e25...	workspace.default.orders_managed	null	

SQL code:

The screenshot shows the Databricks SQL interface. The query executed is `DESCRIBE FORMATTED orders_managed;`. The result is a table with 2 columns: col_name and data_type. The table provides detailed information about the `orders_managed` table, including its catalog, database, table name, creation time, last access, creator, statistics, type, and location.

	col_name	data_type
12	# Detailed Table Information	
13	Catalog	workspace
14	Database	default
15	Table	orders_managed
16	Created Time	Tue Sep 02 10:56:23 UTC 2025
17	Last Access	UNKNOWN
18	Created By	Spark
19	Statistics	1833 bytes, 3 rows
20	Type	MANAGED
21	Location	

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.

The first screenshot shows the execution of the SQL command `create catalog deltalake_catalog;` which completed successfully 'Just now (7s)'. The second screenshot shows the execution of `use catalog deltalake_catalog;` which also completed successfully 'Just now (1s)'. The third screenshot shows the execution of `SELECT current_catalog();` which completed successfully 'Just now (6s)'. Below the SQL results, a table is displayed with the following data:

Table	current_catalog()
1	deltalake_catalog

Pyspark:

```
df = spark.sql("SELECT current_catalog()")
df.show(truncate=False)
```


Table will be created in below location catalog → default

Catalog Explorer > deltalake_catalog > default

default

Use with BI tools Share Create


Overview Details Permissions

Description 

Default schema (auto-created)

Filter tables

Tables 1 Volumes 0 Models 0 Functions 0 Sort

Name	Owner	Created at	Popularity
 orders_managed	technicalsupport@trendyt...	Sep 02, 2025, 04:58 PM	----


About this schema

Catalog Explorer > deltalake_catalog > default > orders_managed




orders_managed

Open in a dashboard Share Create

Overview Sample Data Details Permissions History Lineage Insights Quality

 AI Suggested Description

The table contains data related to product orders. It includes details such as order IDs, product SKUs, names, categories, quantities, and unit prices. This information can be used for analyzing sales performance, tracking inventory levels, and understanding customer purchasing patterns.

 Accept  Edit  Send feedback

Filter columns...

AI generate

Column	Type	Comment	Tags	Column masking rule
order_id	bigint			
sku	string			
product_name	string			

Delta builder APIs

```
< > + Code + Text + Assistant
Just now (3s) 15 Python
from delta.tables import DeltaTable

DeltaTable.createIfNotExists(spark) \
    .tableName("orders_managed_new") \
    .addColumn("order_id", "BIGINT") \
    .addColumn("sku", "STRING") \
    .addColumn("product_name", "STRING") \
    .addColumn("product_category", "STRING", comment = "this is the product category") \
    .addColumn("qty", "INT") \
    .addColumn("unit_price", "DECIMAL(10,2)") \
    .execute()

> See performance \(1\) Optimize
<delta.connect.tables.DeltaTable at 0x7ff5eed9bd40>
```

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

```
# Define schema
```

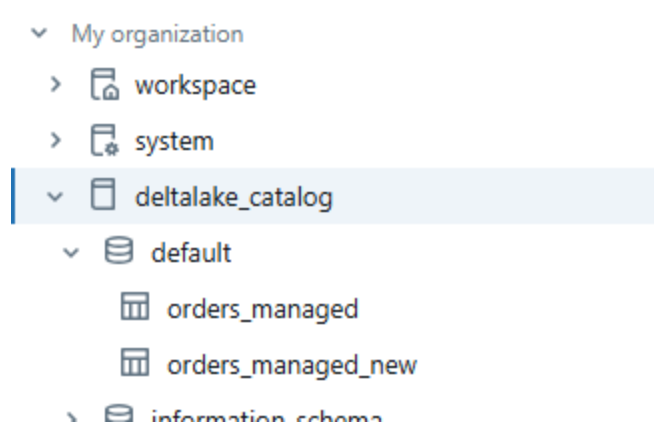
```
schema = StructType([
    StructField("order_id", LongType(), True),
    StructField("sku", StringType(), True),
    StructField("product_name", StringType(), True),
    StructField("product_category", StringType(), True), # comment not supported in PySpark
schema
    StructField("qty", IntegerType(), 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")
```

- My organization
 - workspace
 - system
 - deltalake_catalog**
 - default
 - Tables (2)
 - Volumes (1)**
 - information_schema
 - llm
 - Delta Shares Received
 - samples

Catalog

Serverless Starter Warehouse Serverless 2XS

Catalog Explorer > deltalake_catalog > default >

delta_volume1

Share

Upload to this volume

Type to search...

- My organization
 - workspace
 - system
 - deltalake_catalog
 - default
 - Tables (2)
 - Volumes (1)
 - delta_volume1**
 - information_schema

Overview

Files

Details

Permissions

Resource Name	/metastores/5f864d67-9999-41f0-8c1d-da085226bfc/volumes/58e49764-4b72-4142-a9ae-26d341e3ed76
Volume Type	MANAGED
Storage Location	Default Storage
Volume Id	58e49764-4b72-4142-a9ae-26d341e3ed76
Created At	Sep 02, 2025, 05:21 PM
Created By	technicalsupport@trendytech.in

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

[See performance \(1\)](#)
Optimize

```
true
```

Just now (1s)

4

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

[See performance \(1\)](#)

```
true
```

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)
```

```
▶ Just now (3s) 7
volume_path = "/Volumes/deltalake_catalog/default/delta_volume1/ordersdata1"
df.write.format("delta").mode("overwrite").save(volume_path)
> See performance \(1\) Optimiz
```

Above is already a valid python code.

SQL code:

```
▶ Just now (3s) 8
%sql
SELECT * FROM DELTA.`/Volumes/deltalake_catalog/default/delta_volume1/ordersdata1`
> See performance \(1\) Optimiz
```

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

	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)
```

Catalog Explorer > deltalake_catalog > default > delta_volume1

Share Upload to this volume

Overview Files Details Permissions

Description

Add description

/Volumes/deltalake_catalog/default/delta_volume1 / ordersdata1

Filter files and directories... Create directory Copy path

Name	Size	Last modified
part-00000-9ba78e7d-db2a-4d15-96b6-a4e0fbf05c	1.93 KB	1 minute ago
_delta_log		

About this volume

Owner technicalsupport@trendytech.in

Tags

Just now (1s) 9 SQL

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

See performance (1) Optimize

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

format	id	name	descrip...	location
delta	edbcaec0-441b-4978-96c5-ff0b888910...	null	null	dbfs:/Volumes/deltalake_catalog/default/delta_volume1/ordersdata1

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

Just now (1s) 11 Python

```
%fs ls /Volumes/deltalake_catalog/default/delta_volume1/ordersdata1/
```

> [See performance \(2\)](#)

Table +

	path	
1	dbfs:/Volumes/deltalake_catalog/default/delta_volume1/ordersdata1/_delta_log/	_delta
2	dbfs:/Volumes/deltalake_catalog/default/delta_volume1/ordersdata1/part-00000-9ba78e7d-db2a-4d15-96b6-a4e0bf05db9.c000.snappy.parquet	part-(

Copy the path of the json file

Just now (1s) 11

```
%fs ls /Volumes/deltalake_catalog/default/delta_volume1/ordersdata1/_delta_log
```

> [See performance \(1\)](#)

Table +

	path
1	dbfs:/Volumes/deltalake_catalog/default/delta_volume1/ordersdata1/_delta_log/00000000000000000000.crc
2	dbfs:/Volumes/deltalake_catalog/default/delta_volume1/ordersdata1/_delta_log/00000000000000000000.json
3	dbfs:/Volumes/deltalake_catalog/default/delta_volume1/ordersdata1/_delta_log/_staged_commits/

1 minute ago (2s) 12

```
log_file_path = "dbfs:/Volumes/deltalake_catalog/default/delta_volume1/ordersdata1/_delta_log/00000000000000000000.json"
df_log = spark.read.json(log_file_path)
display(df_log)
```

> [See performance \(1\)](#) Optimize

df_log: pyspark.sql.connect.dataframe.DataFrame = [add: struct, commitInfo: struct ... 2 more fields]

Table +

	commitInfo
1	object clusterId: "0902-105425-jnu6hoyr-v2n" engineInfo: "Databricks-Runtime/17.1.x-photon-scala2.13" isBlindAppend: false isolationLevel: "WriteSerializable" ...

Sql way:

```
%sql
SELECT * FROM JSON.`dbfs:/Volumes/deltalake_catalog/default/delta_volume1/ordersdata/_delta_log/00000000000000000000.json`

> | See performance (1)

_sqldf: pyspark.sql.connect.dataframe.DataFrame = [add: struct, commitInfo: struct ... 2 more fields]
```

Table	add	commitInfo
1	null	> { "clusterId": "0902-10542"
2	null	null
3	null	null
4	> { "dataChange": true, "modificationTime": 1756814135000, "path": "part-00000-9ba78e7d-db2a-4d15-96b6-a4e0fbf05db9.c000.snappy.pa...	null

4 rows | 4.02s runtime

This result is stored as `_sqldf` and can be used in other Python and SQL cells.

Pyspark code:

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

```
.load("dbfs:/Volumes/deltalake_catalog/default/delta_volume1/ordersdata1/_delta_log/00000000  
00000000000000.json")
```

```
df.show(truncate=False)
```

Parquet path:

SQL code:

```
▶ Last execution failed 14
```

```
1 %sql
2 SELECT * FROM PARQUET.`dbfs:/Volumes/deltalake_catalog/default/delta_volume1/ordersdata1/
part-00000-9ba78e7d-db2a-4d15-96b6-a4e0fbf05db9.c000.snappy.parquet`
```

[DELTA_INVALID_FORMAT] Incompatible format detected.

A transaction log for Delta was found at `dbfs:/Volumes/deltalake_catalog/default/delta_volume1/ordersdata1/_delta_log`, but you are trying to read from `dbfs:/Volumes/deltalake_catalog/default/delta_volume1/ordersdata1/part-00000-9ba78e7d-db2a-4d15-96b6-a4e0fbf05db9.c000.snappy.parquet` using format("PARQUET"). You must use 'format("delta")' when reading and writing to a delta table.

To learn more about Delta, see <https://docs.databricks.com/delta/index.html> SQLSTATE: 22000

 Diagnose error

Assistant Quick Fix: ON ▼

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 \(1\)](#) Optimize

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

	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

```
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

Catalog Explorer > deltalake_catalog > default >

delta_volume1 Share Upload to this volume

Overview Files Details Permissions

Description
Add description

/Volumes/detalake_catalog/default/delta_volume1 / ordersdata1

Filter files and directories... Create directory Copy path

Name	Size	Last modified
part-00000-63e76649-0a36-469c-a24a-4d557d25e	1.79 KB	1 minute ago
part-00000-9ba78e7d-db2a-4d15-96b6-a4e0fbf05r	1.93 KB	12 minutes ago
_delta_log		

2 minutes ago (3s) 18 SQL

```
%sql
describe history DELTA.`/Volumes/detalake_catalog/default/delta_volume1/ordersdata1`
> See performance \(1\) Optimize
```

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

Table +

	operation	operationParameters	job	notebook	clusterId
1	WRITE	> {"mode":"Append","statsOnLoad":"false","partitionBy":...	null	> {"notebookId":"483500620014...	0902-105425-jnu6hojr-v...
2	WRITE	> {"mode":"Overwrite","statsOnLoad":"false","partitionBy":...	null	> {"notebookId":"483500620014...	0902-105425-jnu6hojr-v...

```
df = spark.sql("""
DESCRIBE HISTORY DELTA.`/Volumes/detalake_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/orders/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/_d
elta_log/00000000000000000002.json`
```

Pyspark:

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/000
00000000000000002.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
```

```
DROP TABLE orders_ext_01;
```

Pyspark:

```
spark.sql("DROP TABLE IF EXISTS orders_ext_01")
```

```
%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
```

```
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")
```

