

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 interface. On the left, there is a notebook with two cells labeled 1 and 2. Cell 1 is selected and has a Python icon. The right side of the interface features the "Environment" sidebar. The sidebar includes settings for "Memory" (Standard (16GB)), "Serverless budget policy" (None), "Base environment" (set to a local path), "Environment version" (version 3), and "Dependencies" (Added tab selected). A yellow box highlights the "Environment" tab in the sidebar.

To create budget policy -

User → settings → compute → policies

The screenshot shows the Databricks Settings page. The left sidebar has sections for Workspace admin, Appearance, Identity and access, Security, Compute (selected), Development, Notifications, Advanced, User, Profile, Preferences, Developer, Linked accounts, and Notifications. The main content area is titled "SQL warehouses and serverless compute". It contains sections for "Environments" (with a "Default package repositories" link and a "Manage" button) and "Policies" (with a "Serverless budget policies" link and a "Manage" button highlighted with a yellow box).

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) X

X

Tags

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

Tag	Value	
project	databricks_project	
topic	compute	
+ Add tag		

Environment Send feedback X

Memory Standard (16GB)

Serverless budget policy demo-budget-policy

databricks_project compute

Base environment /path/to/base_environment.y...

Environment version 3

Dependencies

Added Available

simplejson==3.19.*

Installing dependencies:

The screenshot shows the 'Dependencies' section of the Databricks UI. At the top, there are tabs for 'Added' and 'Available'. Below them, a search bar contains the text 'faker'. To the right of the search bar is a three-dot menu icon. At the bottom left is a blue '+ Add dependency' button.

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

To Schedule the notebook:

The screenshot shows the 'New schedule' dialog. At the top, it says 'Job name*' with the value 'demo-notebook'. Below that are two buttons: 'Simple' (which is selected) and 'Advanced'. Under 'Schedule', there are dropdown menus for 'Every' (set to '1') and 'Day'. In the 'Compute*' section, 'Serverless' is selected under 'Autoscaling'. Under 'Performance optimization', there is an unchecked checkbox for 'Performance optimized'. At the bottom right are 'Cancel' and 'Create' buttons.

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

The screenshot shows the Databricks workspace interface. On the left, there's a sidebar with links to Home, Shared with me, Workspace, Favorites, and Trash. The main area shows a list of items under the user 'technicalsupport@trendytech.in'. A search bar at the top has 'Search' and dropdowns for 'Type' and 'Owner'. A 'Create' button is visible. A context menu is open over a row, with 'Folder' highlighted in yellow. Other options in the menu include 'Git folder', 'Notebook', 'File', 'Query', and 'Dashboard'. The list includes 'demo-folder' (Folder, Owner: technicalsup), 'Bakehouse Sales Starter Space' (Genie space, Owner: technicalsup), and 'Workspace Usage Dashboard' (Dashboard, Owner: technicalsup).

Create a notebook inside this folder

This screenshot shows a Databricks notebook cell. The cell has a green checkmark icon and the text '2 minutes ago (2m)'. It contains the following SQL code:

```
%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;
```

Below the code, there's a link 'See performance (1)'.

This screenshot shows another Databricks notebook cell. It has an 'Interrupt' button and a timer '00:21'. It contains the following SQL code:

```
%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);
```

Below the code, there's a link 'See performance' and a status bar showing 'Statement 1/1 | Tasks 1/1 (0 running)'.

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 a SQL editor interface with the following details:

- SQL Statement:** %sql DESCRIBE DETAIL orders_managed;
- Table View:** A table titled "orders_managed" is displayed with the following columns and data:

	format	id	name	description	location
1	delta	d4cc0501-28d2-49ea-a5c2-f143880e25...	workspace.default.orders_managed	null	
- Performance:** An "Optimize" button is visible.

SQL code:

The screenshot shows a SQL editor interface with the following details:

- SQL Statement:** %sql DESCRIBE FORMATTED orders_managed;
- Table View:** A table titled "# Detailed Table Information" is displayed with the following columns and data:

col_name	data_type
Catalog	workspace
Database	default
Table	orders_managed
Created Time	Tue Sep 02 10:56:23 UTC 2025
Last Access	UNKNOWN
Created By	Spark
Statistics	1833 bytes, 3 rows
Type	MANAGED
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.

```
▶ ✓ Just now (7s) 6  
%sql  
create catalog deltalake_catalog;  
> See performance (1)
```

```
▶ ✓ Just now (1s) 7  
%sql  
use catalog deltalake_catalog;  
> See performance (1)
```

```
⋮ ▶ ✓ Just now (6s) 8  
%sql  
SELECT current_catalog();  
> See performance (1)  
↳ _sqldf: pyspark.sql.connect.DataFrame = [current_catalog(): string]  
  
Table +  
-----  
B 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 > deltlake_catalog >

default ⚙ ☆

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 > deltlake_catalog > default >

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...

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

Delta builder APIs



A screenshot of the PyCharm IDE interface. The code editor shows Python code for creating a DeltaTable. The code uses the `DeltaTable.createIfNotExists` method to create a table named "orders_managed_new" with columns for order_id (BIGINT), sku (STRING), product_name (STRING), product_category (STRING), qty (INT), and unit_price (DECIMAL(10,2)). The code is run in Python mode, and the status bar indicates the code was run just now (3s) and has 15 lines.

```
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)

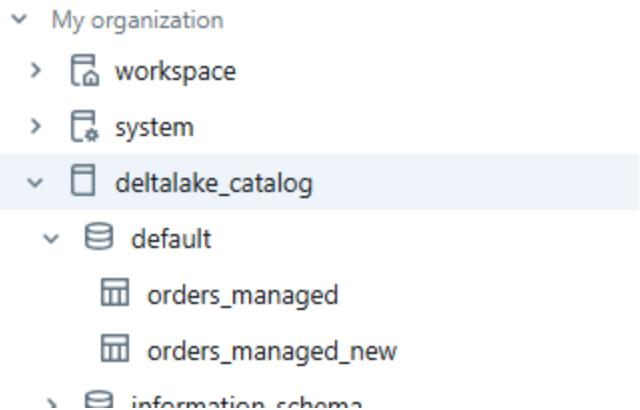
<delta.connect.tables.DeltaTable at 0x7ff5eed9bd40>
```

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
    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")
```

Type to search... Filter

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

Catalog Share Upload to this volume

Catalog Explorer > **deltalake_catalog** > default > **delta_volume1**

Overview	Files	Details	Permissions
Resource Name	/metastores/5f864d67-9999-41f0-8c1d-da085226bcfc/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

Just now (1s) 4

```
%fs mkdirs /Volumes/deltalake_catalog/default/delta_volume1/ordersdata2
> See performance (1)
```

Just now (5s) 5

```
%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	1756814031205

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:

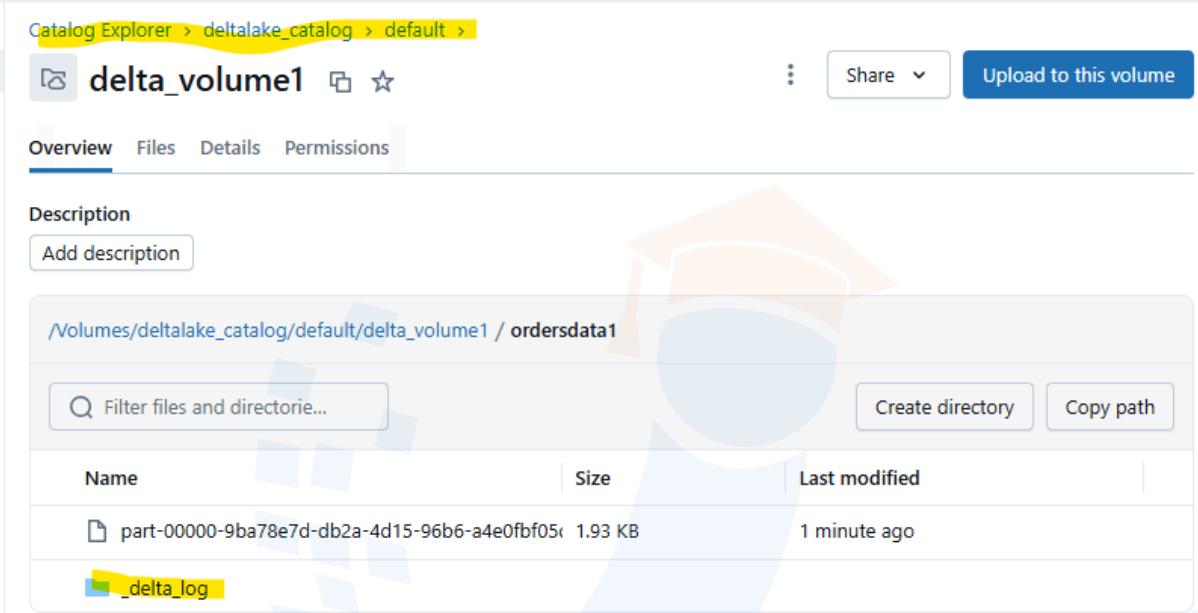
▶ ✓ Just now (3s) 8 SQL 🗑️ ⚙️ ⌂ ⌂
%sql
SELECT * FROM DELTA.`/Volumes/deltalake_catalog/default/delta_volume1/ordersdata1`
› See performance (1) Optimiz...
‣ _sqldf: pyspark.sql.connect.DataFrame = [order_id: long, sku: string ... 4 more fields]
Table +

1	2	3	4	5	6	7	8
1	2	3	4	5	6	7	8
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	

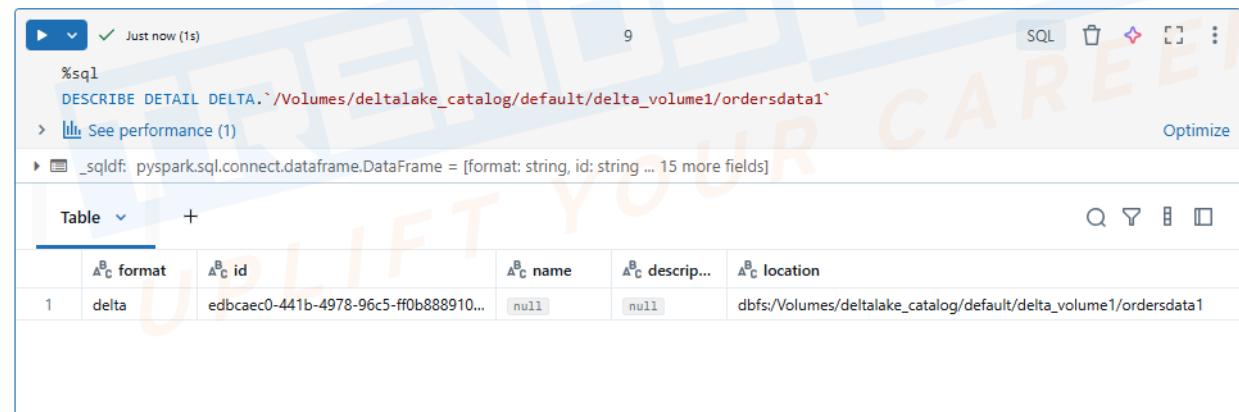
Pyspark equivalent:

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

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



The screenshot shows the Delta Lake Catalog Explorer interface. At the top, it displays the path: Catalog Explorer > deltalake_catalog > default > delta_volume1. Below this, there are tabs for Overview, Files, Details, and Permissions, with Overview selected. The Overview section includes a Description field with an 'Add description' button, a URL field showing /Volumes/deltalake_catalog/default/delta_volume1/ordersdata1, a search bar labeled 'Filter files and directorie...', and buttons for 'Create directory' and 'Copy path'. A table lists files and directories: 'part-00000-9ba78e7d-db2a-4d15-96b6-a4e0fbf05c' (1.93 KB) was modified 1 minute ago, and a '_delta_log' directory is also listed. Below the table, the 'About this volume' section shows the owner as technicalsupport@trendytech.in. The 'Tags' section is present but empty.



The screenshot shows the PySpark SQL shell interface. The command entered is:

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

The results are displayed in a table:

	format	id	name	descrip...	location
1	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 +

	A ^B C path	A ^B C n
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-a4e0fbf05db9.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	+
	A ^B path	
1	dbfs:/Volumes/deltalake_catalog/default/delta_volume1/ordersdata1/_delta_log/000000000000000000000000.crc	
2	dbfs:/Volumes/deltalake_catalog/default/delta_volume1/ordersdata1/_delta_log/000000000000000000000000.json	
3	dbfs:/Volumes/deltalake_catalog/default/delta_volume1/ordersdata1/_delta_log/_staged_commits/	

▶ ✓ 1 minute ago (2s) 12

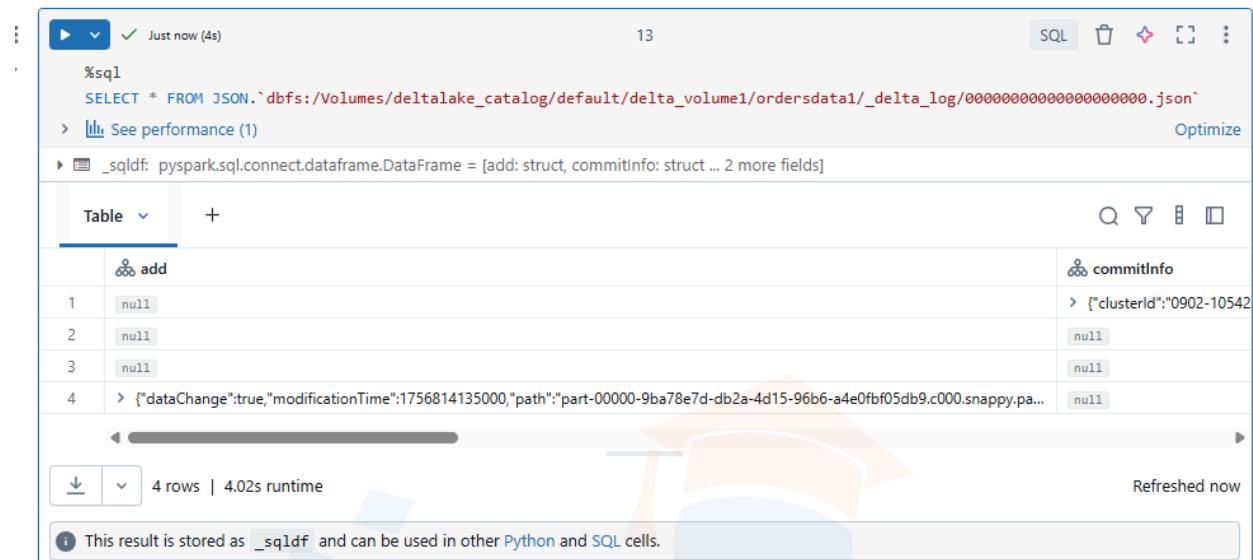
```
log_file_path = "dbfs:/Volumes/deltalake_catalog/default/delta_volume1/ordersdata1/_delta_log/00000000000000000000.json"
df_log = spark.read.json(log_file_path)
display(df_log)
```

▶ See performance (1) Optimize

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

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

Sql way:



The screenshot shows a Databricks SQL notebook interface. At the top, there's a toolbar with a play button, a green checkmark, and the text "Just now (4s)". To the right are icons for SQL, trash, and other operations. Below the toolbar, the cell number "13" is displayed. The code cell contains the following SQL command:

```
%sql  
SELECT * FROM JSON.`dbfs:/Volumes/deltalake_catalog/default/delta_volume1/ordersdata1/_delta_log/00000000000000000000.json`
```

Below the code, it says "See performance (1)" and "Optimize". The result pane shows a table with two columns: "add" and "commitInfo". The "add" column has four rows, all containing "null". The "commitInfo" column also has four rows, all containing "null". At the bottom of the result pane, it says "4 rows | 4.02s runtime" and "Refreshed now". A note at the bottom states: "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/00000000000000000000.json")  
  
df.show(truncate=False)
```

Parquet path:

SQL code:



The screenshot shows a Databricks SQL notebook interface. At the top, there's a toolbar with a play button, a red error icon, and the text "Last execution failed". To the right are icons for SQL, trash, and other operations. Below the toolbar, the cell number "14" is displayed. The code cell contains the following SQL command:

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

The result pane shows an error message: "[DELTA_INVALID_FORMAT] Incompatible format detected." Below the error message, there's a detailed explanation: "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." At the bottom of the result pane, it says "To learn more about Delta, see <https://docs.databricks.com/delta/index.html> SQLSTATE: 22000". There are "Diagnose error" and "Assistant Quick Fix: ON" buttons at the bottom.

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)]
```

> new_df: pyspark.sql.connect.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

Catalog Explorer > deltalake_catalog > default >

delta_volume1

[Overview](#) [Files](#) [Details](#) [Permissions](#)

Description

Add description

/Volumes/deltalake_catalog/default/delta_volume1 / ordersdata1

Filter files and directories...

Name	Size	Last modified
part-00000-63e76649-0a36-469c-a24a-4d557d25e	1.79 KB	1 minute ago
part-00000-9ba78e7d-db2a-4d15-96b6-a4e0fbf05e	1.93 KB	12 minutes ago
_delta_log		

2 minutes ago (3s) 18 SQL

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

See performance (1) Optimize

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

Table +

operation	operationParameters	job	notebook	clusterId
WRITE	> {"mode":"Append","statsOnLoad":false,"partitionBy":...}	null	> {"notebookId":483500620014...}	0902-105425-jnu6hoyr-v...
WRITE	> {"mode":"Overwrite","statsOnLoad":false,"partitionBy":...}	null	> {"notebookId":483500620014...}	0902-105425-jnu6hoyr-v...

```
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/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/000000000000000000000005.json`
```

Pyspark:

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

```
.load("abfss://externaldata@ttmystorageaccount001.dfs.core.windows.net/orders/_delta_log/000000000000000000000005.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/000000000000000000000002.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/000000000000000000000002.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")
```