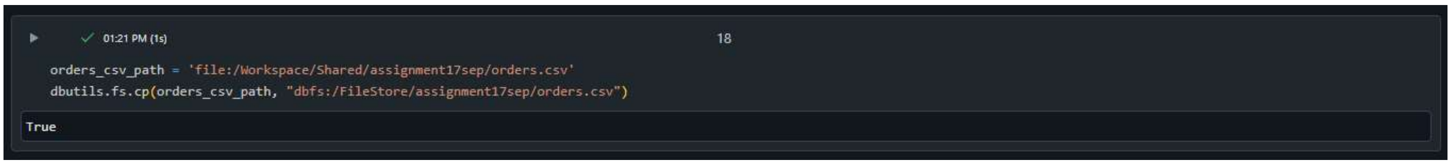
Assignment: Creating a Complete ETL Pipeline using Delta Live Tables (DLT)

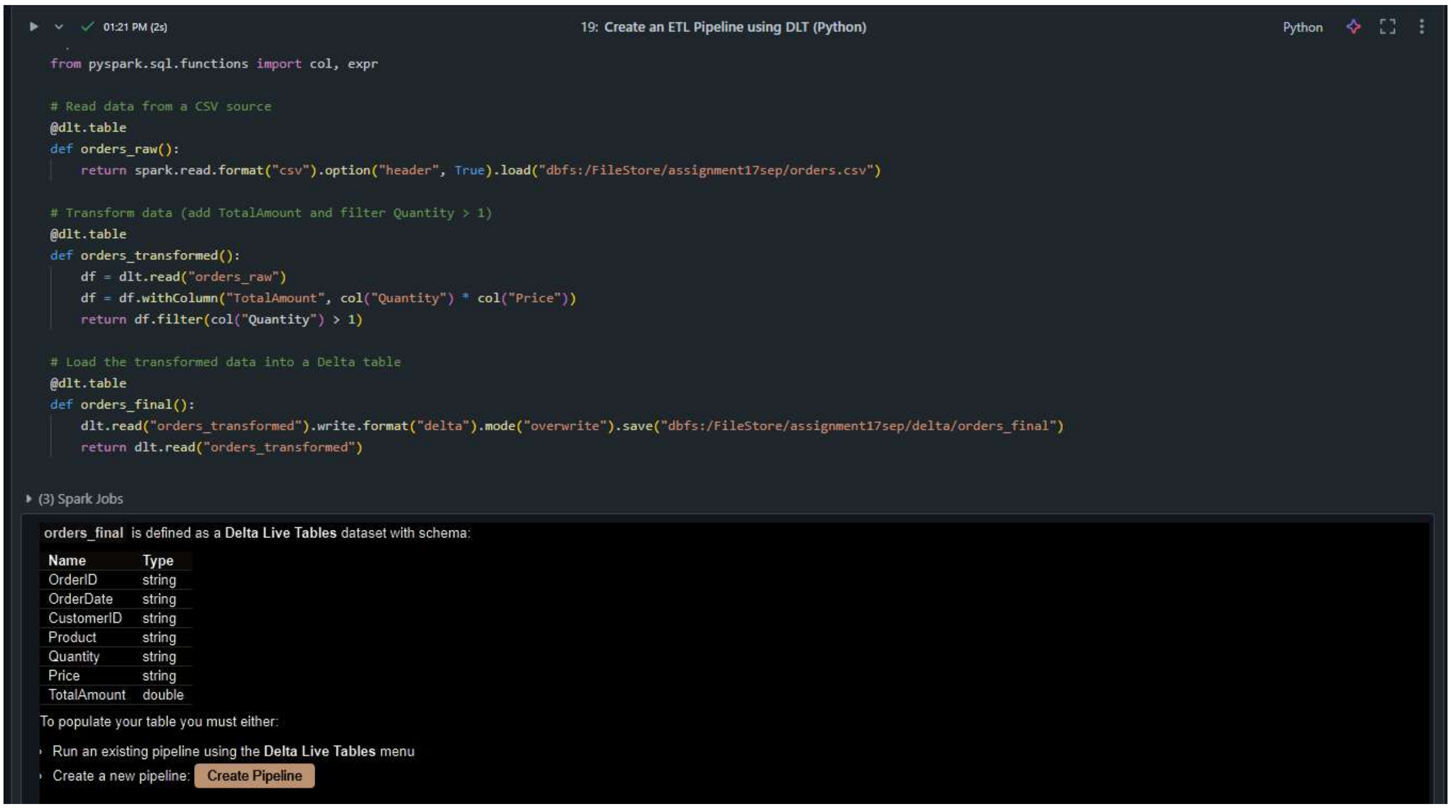
SaiPrabath Chowdary S



01:21 PM (1s) 18

```
orders_csv_path = 'file:/Workspace/Shared/assignment17sep/orders.csv'
dbutils.fs.cp(orders_csv_path, "dbfs:/FileStore/assignment17sep/orders.csv")
```

```
True
```



01:21 PM (2s) 19: Create an ETL Pipeline using DLT (Python)

```
from pyspark.sql.functions import col, expr

# Read data from a CSV source
@dlt.table
def orders_raw():
    return spark.read.format("csv").option("header", True).load("dbfs:/FileStore/assignment17sep/orders.csv")

# Transform data (add TotalAmount and filter Quantity > 1)
@dlt.table
def orders_transformed():
    df = dlt.read("orders_raw")
    df = df.withColumn("TotalAmount", col("Quantity") * col("Price"))
    return df.filter(col("Quantity") > 1)

# Load the transformed data into a Delta table
@dlt.table
def orders_final():
    dlt.read("orders_transformed").write.format("delta").mode("overwrite").save("dbfs:/FileStore/assignment17sep/delta/orders_final")
    return dlt.read("orders_transformed")
```

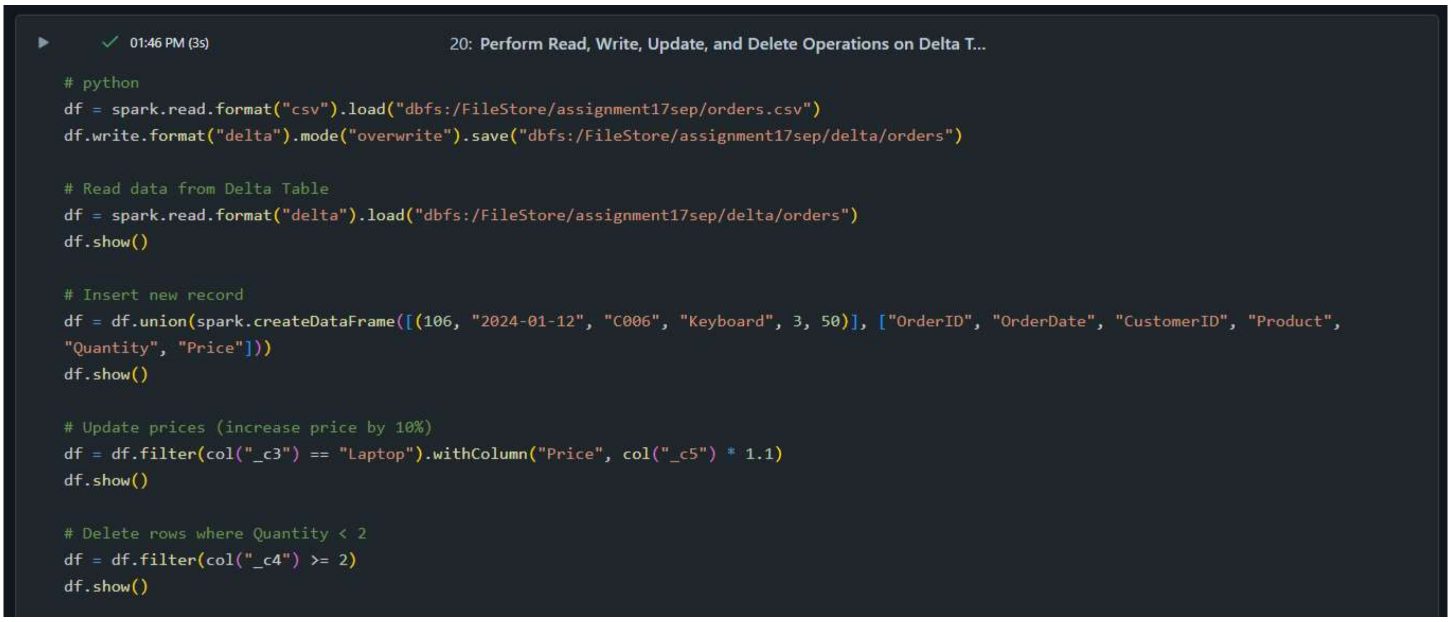
(3) Spark Jobs

orders_final is defined as a **Delta Live Tables** dataset with schema:

Name	Type
OrderID	string
OrderDate	string
CustomerID	string
Product	string
Quantity	string
Price	string
TotalAmount	double

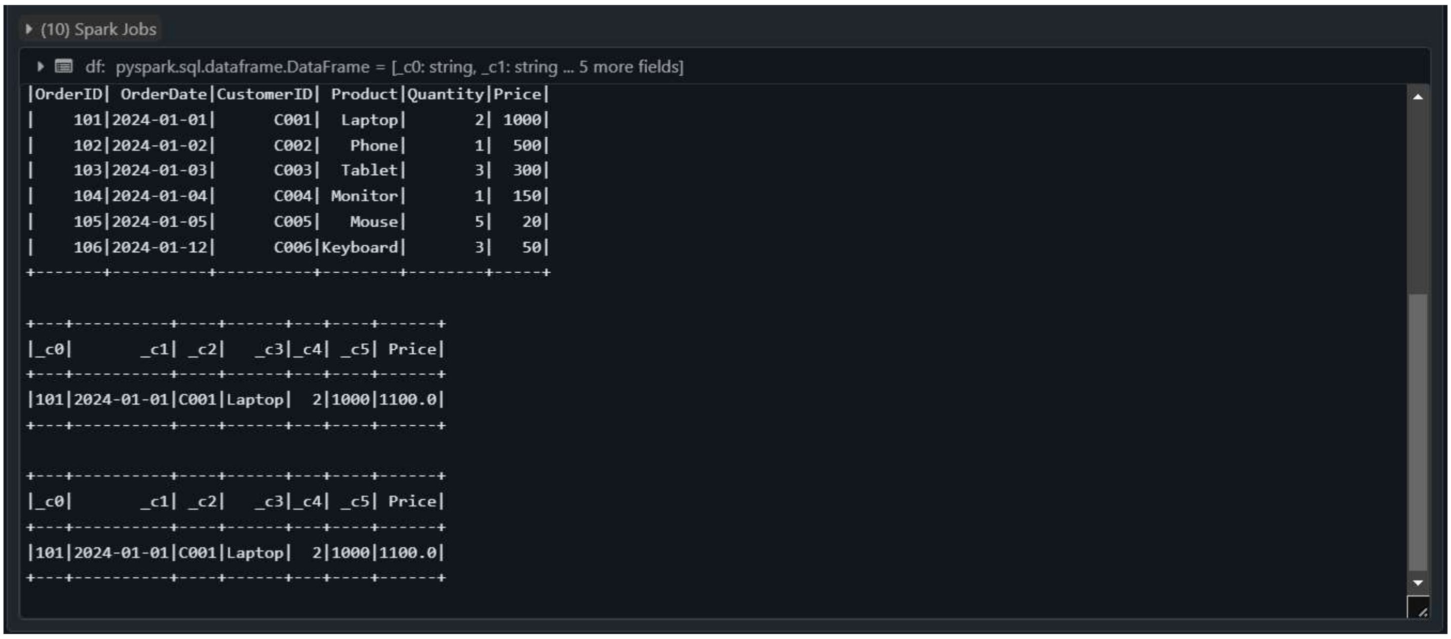
To populate your table you must either:

- Run an existing pipeline using the **Delta Live Tables** menu
- Create a new pipeline: Create Pipeline



01:46 PM (3s) 20: Perform Read, Write, Update, and Delete Operations on Delta T...

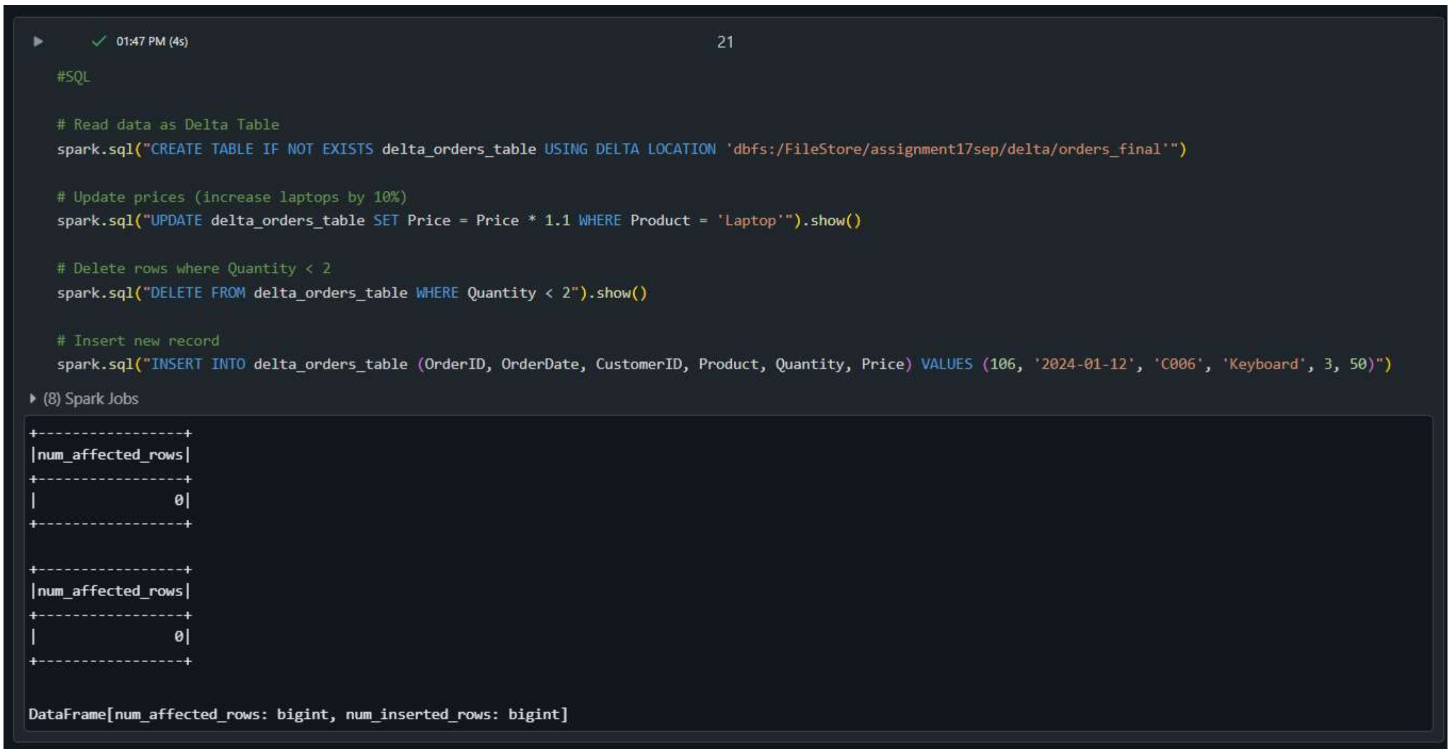
```
# python
df = spark.read.format("csv").load("dbfs:/FileStore/assignment17sep/orders.csv")
df.write.format("delta").mode("overwrite").save("dbfs:/FileStore/assignment17sep/delta/orders")

# Read data from Delta Table
df = spark.read.format("delta").load("dbfs:/FileStore/assignment17sep/delta/orders")
df.show()

# Insert new record
df = df.union(spark.createDataFrame([(106, "2024-01-12", "C006", "Keyboard", 3, 50)], ["OrderID", "OrderDate", "CustomerID", "Product",
"Quantity", "Price"]))
df.show()

# Update prices (increase price by 10%)
df = df.filter(col("_c3") == "Laptop").withColumn("Price", col("_c5") * 1.1)
df.show()

# Delete rows where Quantity < 2
df = df.filter(col("_c4") >= 2)
df.show()
```



(10) Spark Jobs

df: pyspark.sql.dataframe.DataFrame = [_c0: string, _c1: string ... 5 more fields]

```
|OrderID| OrderDate|CustomerID| Product|Quantity|Price|
|    101|2024-01-01|      C001|  Laptop|       2| 1000|
|    102|2024-01-02|      C002|   Phone|       1|  500|
|    103|2024-01-03|      C003|  Tablet|       3|  300|
|    104|2024-01-04|      C004| Monitor|       1|  150|
|    105|2024-01-05|      C005|   Mouse|       5|   20|
|    106|2024-01-12|      C006|Keyboard|       3|   50|
+-------+----------+----------+--------+--------+-----+

+---+----------+----+------+---+----+------+
|_c0|       _c1| _c2|   _c3|_c4| _c5| Price|
+---+----------+----+------+---+----+------+
|101|2024-01-01|C001|Laptop|  2|1000|1100.0|
+---+----------+----+------+---+----+------+

+---+----------+----+------+---+----+------+
|_c0|       _c1| _c2|   _c3|_c4| _c5| Price|
+---+----------+----+------+---+----+------+
|101|2024-01-01|C001|Laptop|  2|1000|1100.0|
+---+----------+----+------+---+----+------+
```

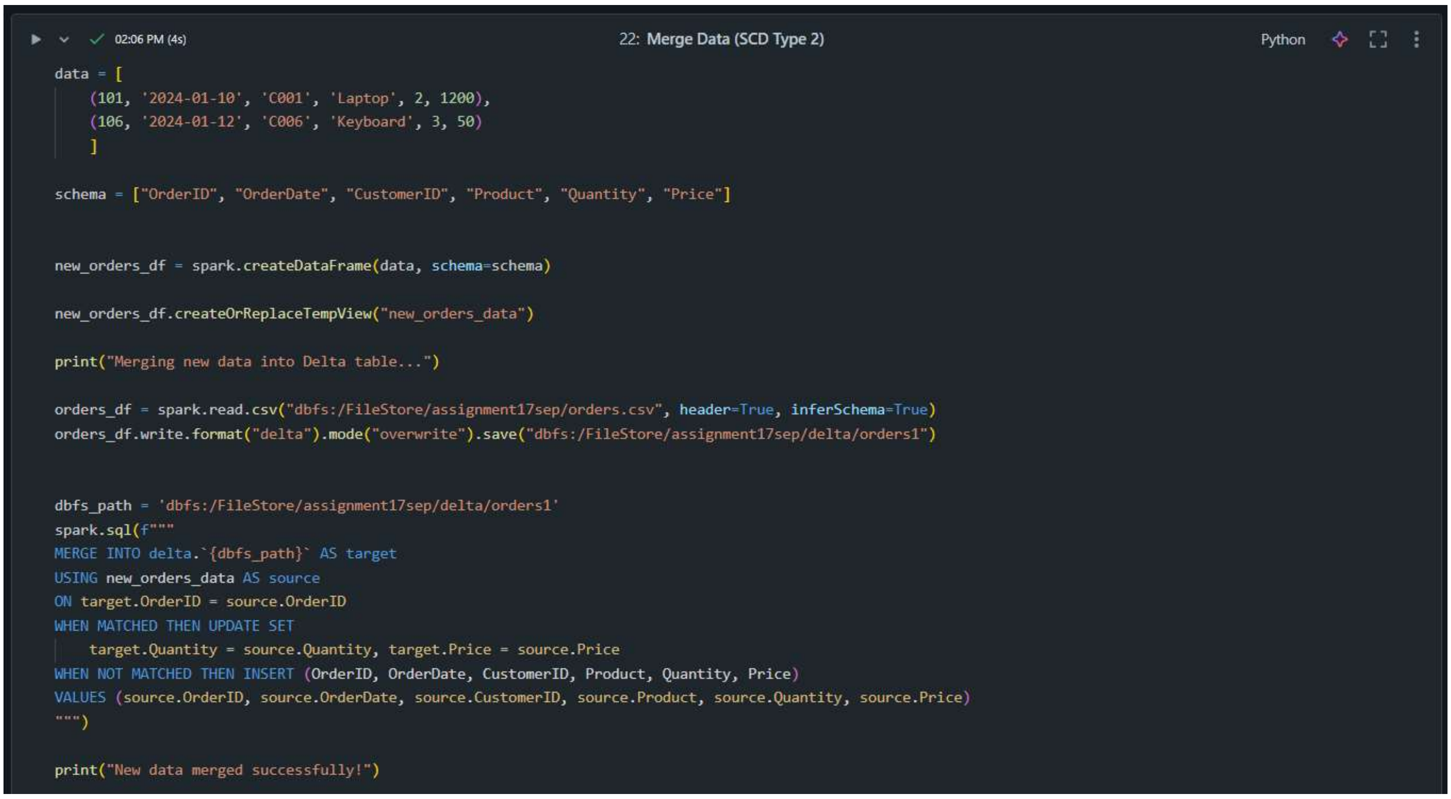


01:47 PM (4s) 21

```
#SQL

# Read data as Delta Table
spark.sql("CREATE TABLE IF NOT EXISTS delta_orders_table USING DELTA LOCATION 'dbfs:/FileStore/assignment17sep/delta/orders_final'")

# Update prices (increase laptops by 10%)
spark.sql("UPDATE delta_orders_table SET Price = Price * 1.1 WHERE Product = 'Laptop'").show()

# Delete rows where Quantity < 2
spark.sql("DELETE FROM delta_orders_table WHERE Quantity < 2").show()

# Insert new record
spark.sql("INSERT INTO delta_orders_table (OrderID, OrderDate, CustomerID, Product, Quantity, Price) VALUES (106, '2024-01-12', 'C006', 'Keyboard', 3, 50)")
```

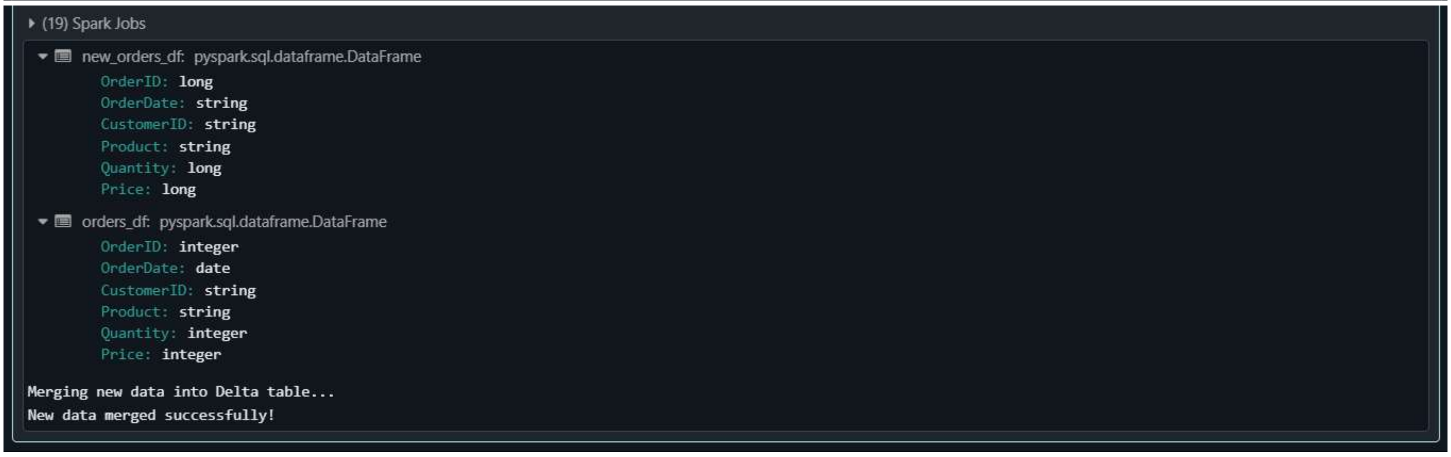
(8) Spark Jobs

```
+-----------------+
|num_affected_rows|
+-----------------+
|                0|
+-----------------+

+-----------------+
|num_affected_rows|
+-----------------+
|                0|
+-----------------+

DataFrame[num_affected_rows: bigint, num_inserted_rows: bigint]
```

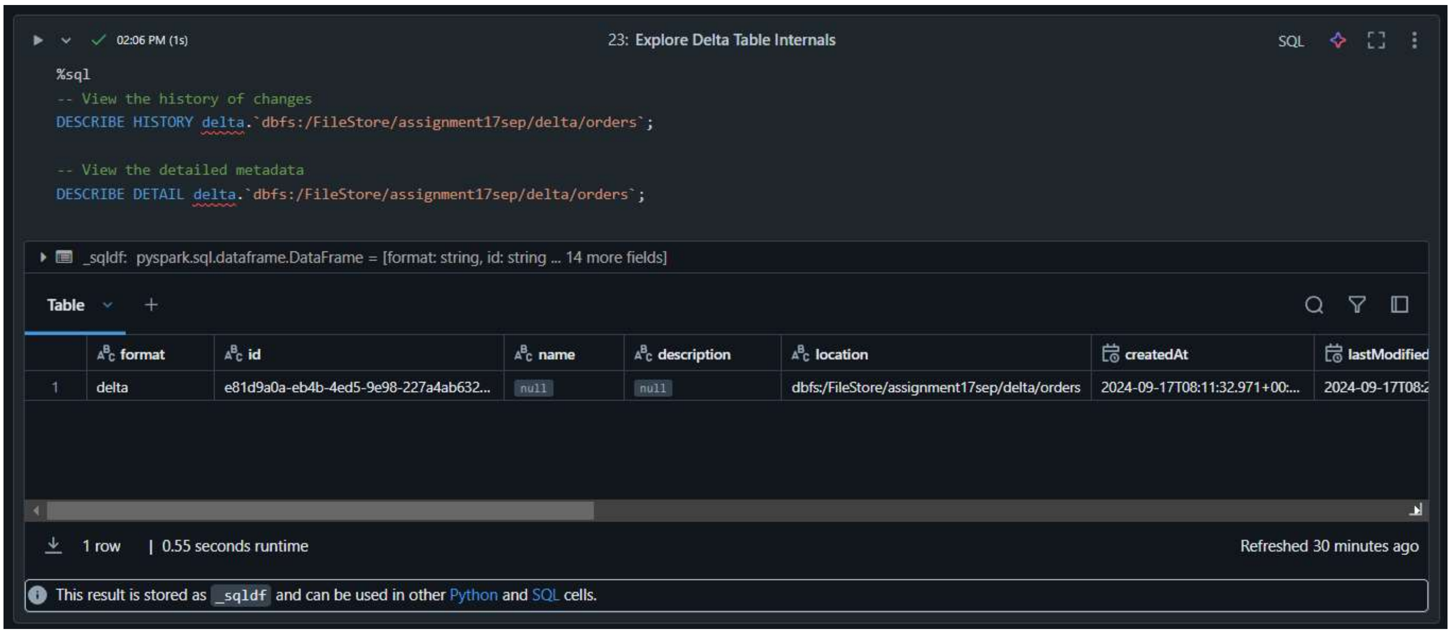


✓ 02:06 PM (4s)   22: Merge Data (SCD Type 2)   Python

```python
data = [
    (101, '2024-01-10', 'C001', 'Laptop', 2, 1200),
    (106, '2024-01-12', 'C006', 'Keyboard', 3, 50)
    ]

schema = ["OrderID", "OrderDate", "CustomerID", "Product", "Quantity", "Price"]


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

new_orders_df.createOrReplaceTempView("new_orders_data")

print("Merging new data into Delta table...")

orders_df = spark.read.csv("dbfs:/FileStore/assignment17sep/orders.csv", header=True, inferSchema=True)
orders_df.write.format("delta").mode("overwrite").save("dbfs:/FileStore/assignment17sep/delta/orders1")


dbfs_path = 'dbfs:/FileStore/assignment17sep/delta/orders1'
spark.sql(f"""
MERGE INTO delta.`{dbfs_path}` AS target
USING new_orders_data AS source
ON target.OrderID = source.OrderID
WHEN MATCHED THEN UPDATE SET
    target.Quantity = source.Quantity, target.Price = source.Price
WHEN NOT MATCHED THEN INSERT (OrderID, OrderDate, CustomerID, Product, Quantity, Price)
VALUES (source.OrderID, source.OrderDate, source.CustomerID, source.Product, source.Quantity, source.Price)
""")

print("New data merged successfully!")
```



```
▸ (19) Spark Jobs
▾ new_orders_df:  pyspark.sql.dataframe.DataFrame
      OrderID:  long
      OrderDate:  string
      CustomerID:  string
      Product:  string
      Quantity:  long
      Price:  long
▾ orders_df:  pyspark.sql.dataframe.DataFrame
      OrderID:  integer
      OrderDate:  date
      CustomerID:  string
      Product:  string
      Quantity:  integer
      Price:  integer

Merging new data into Delta table...
New data merged successfully!
```


23: Explore Delta Table Internals

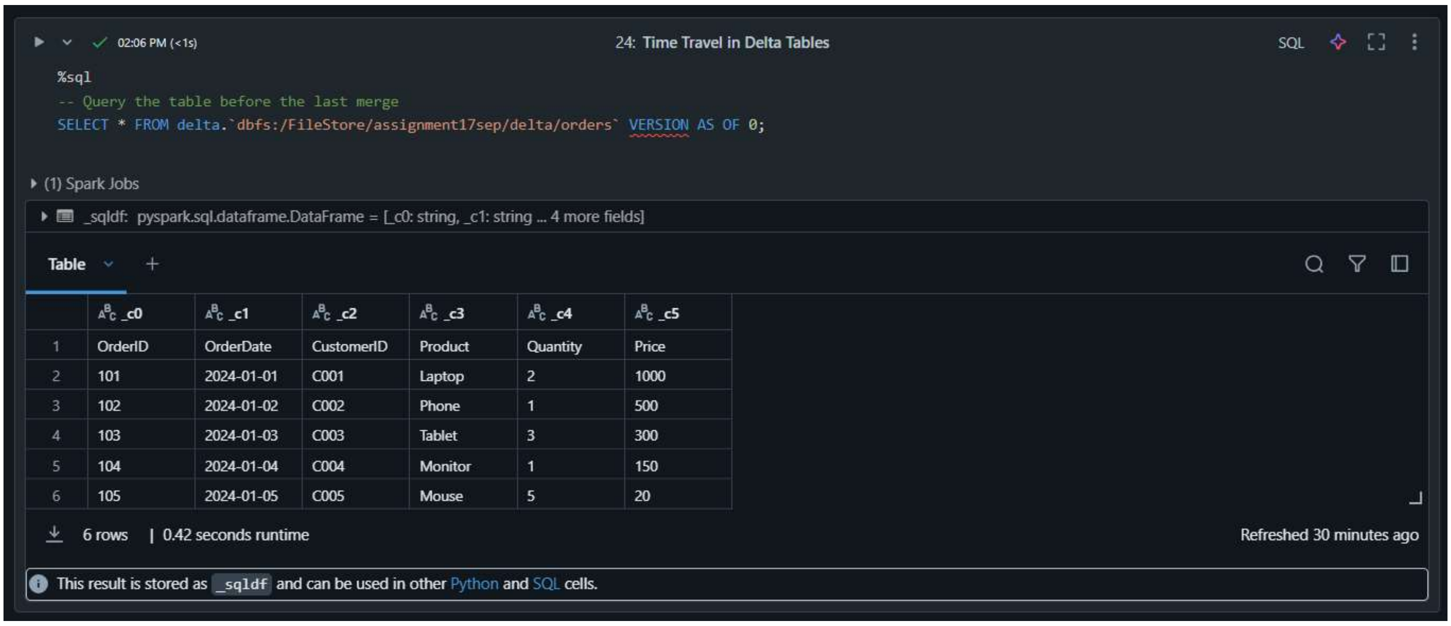
```
%sql
-- View the history of changes
DESCRIBE HISTORY delta.`dbfs:/FileStore/assignment17sep/delta/orders`;

-- View the detailed metadata
DESCRIBE DETAIL delta.`dbfs:/FileStore/assignment17sep/delta/orders`;
```

_sqldf: pyspark.sql.dataframe.DataFrame = [format: string, id: string ... 14 more fields]

	format	id	name	description	location	createdAt	lastModified
1	delta	e81d9a0a-eb4b-4ed5-9e98-227a4ab632...	null	null	dbfs:/FileStore/assignment17sep/delta/orders	2024-09-17T08:11:32.971+00:...	2024-09-17T08:2

1 row | 0.55 seconds runtime — Refreshed 30 minutes ago

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


24: Time Travel in Delta Tables

```
%sql
-- Query the table before the last merge
SELECT * FROM delta.`dbfs:/FileStore/assignment17sep/delta/orders` VERSION AS OF 0;
```

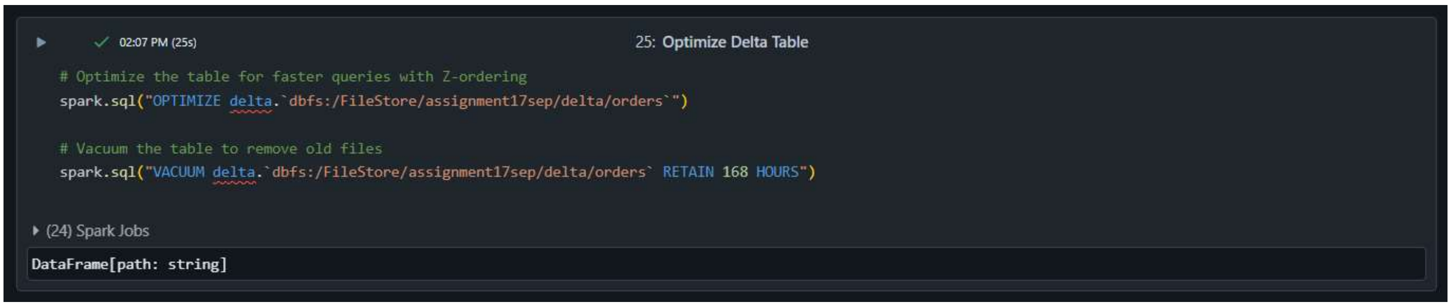
(1) Spark Jobs

_sqldf: pyspark.sql.dataframe.DataFrame = [_c0: string, _c1: string ... 4 more fields]

	_c0	_c1	_c2	_c3	_c4	_c5
1	OrderID	OrderDate	CustomerID	Product	Quantity	Price
2	101	2024-01-01	C001	Laptop	2	1000
3	102	2024-01-02	C002	Phone	1	500
4	103	2024-01-03	C003	Tablet	3	300
5	104	2024-01-04	C004	Monitor	1	150
6	105	2024-01-05	C005	Mouse	5	20

6 rows | 0.42 seconds runtime — Refreshed 30 minutes ago

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


25: Optimize Delta Table

```
# Optimize the table for faster queries with Z-ordering
spark.sql("OPTIMIZE delta.`dbfs:/FileStore/assignment17sep/delta/orders`")

# Vacuum the table to remove old files
spark.sql("VACUUM delta.`dbfs:/FileStore/assignment17sep/delta/orders` RETAIN 168 HOURS")
```

(24) Spark Jobs

```
DataFrame[path: string]
```