Delta Lake Operations (11.08.2025)

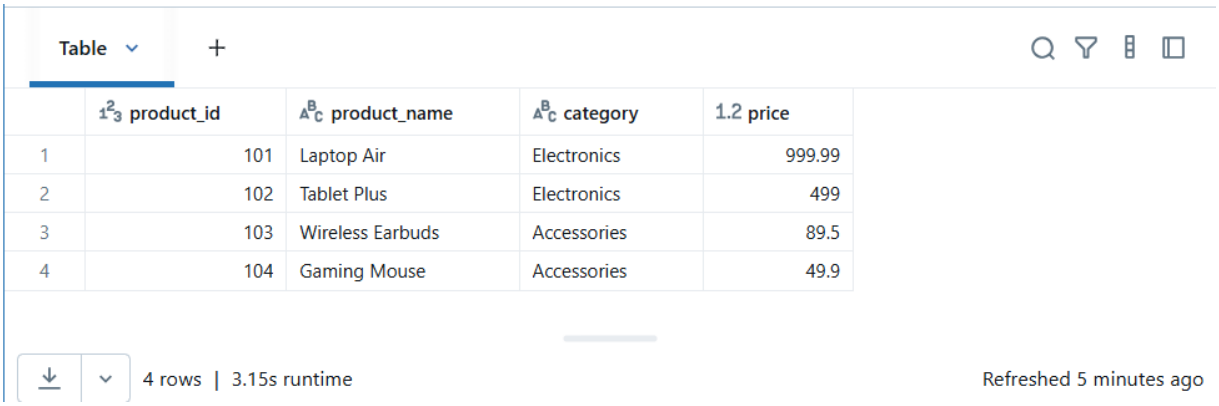
Bavatharani S

1. Create a Delta Table

Instead of reading from a CSV, we directly create a Spark DataFrame with sample records.

```
# Create DataFrame
data = [
    (101, "Laptop Air", "Electronics", 999.99),
    (102, "Tablet Plus", "Electronics", 499.00),
    (103, "Wireless Earbuds", "Accessories", 89.50),
    (104, "Gaming Mouse", "Accessories", 49.90)
]
columns = ["product_id", "product_name", "category", "price"]
df = spark.createDataFrame(data, columns)
display(df)

# Save as managed Delta table (no catalog/schema in CE)
df.write.format("delta").mode("overwrite").saveAsTable("products")
```




```
spark.sql("SELECT * FROM products").show()
```

```
+----------+----------------+-----------+------+
|product_id|    product_name|   category| price|
+----------+----------------+-----------+------+
|       101|      Laptop Air|Electronics|999.99|
|       102|     Tablet Plus|Electronics| 499.0|
|       103|Wireless Earbuds|Accessories|  89.5|
|       104|    Gaming Mouse|Accessories|  49.9|
+----------+----------------+-----------+------+
```

2. Upsert (Merge) Data into the Delta Table

We prepare new data to update an existing product and insert a new one.

```
from delta.tables import DeltaTable
# New data: one update + one new insert
new_data = [
    (102, "Tablet Plus", "Electronics", 550.00),   # Update price for id=102
    (105, "Smartwatch X", "Wearables", 250.00)     # New product with id=105
]
columns = ["product_id", "product_name", "category", "price"]
new_df = spark.createDataFrame(new_data, columns)
# Load existing Delta table
deltaTable = DeltaTable.forName(spark, "products")
# Perform merge (upsert)
deltaTable.alias("old").merge(
    new_df.alias("new"),
    "old.product_id = new.product_id"
).whenMatchedUpdate(set={
    "product_name": "new.product_name",
    "category": "new.category",
    "price": "new.price"
```

}).whenNotMatchedInsert(values={

"product_id": "new.product_id",

"product_name": "new.product_name",

"category": "new.category",

"price": "new.price"

}).execute()

```
new_df:  pyspark.sql.connect.dataframe.DataFrame = [product_id: long, product_name: string ... 2 more fields]
DataFrame[num_affected_rows: bigint, num_updated_rows: bigint, num_deleted_rows: bigint, num_inserted_rows: bigint]
```

spark.sql("SELECT * FROM products ORDER BY product_id").show()

3. Read Data from the Delta Table

spark.sql("SELECT * FROM products ORDER BY product_id").show()

```
+----------+----------------+-----------+------+
|product_id|    product_name|   category| price|
+----------+----------------+-----------+------+
|       101|      Laptop Air|Electronics|999.99|
|       102|     Tablet Plus|Electronics| 499.0|
|       103|Wireless Earbuds|Accessories|  89.5|
|       104|    Gaming Mouse|Accessories|  49.9|
+----------+----------------+-----------+------+
```

4. Display the Table History

history_df = deltaTable.history()

history_df.show(truncate=False)

```
history_df:  pyspark.sql.connect.dataframe.DataFrame = [version: long, timestamp: timestamp ... 13 more fields]
|NULL|NULL    |0817-145439-7olt0x6c-v2n|0          |WriteSerializable|false        |{numFiles -> 1, numR
movedFiles -> 1, numRemovedBytes -> 1476, numOutputRows -> 4, numOutputBytes -> 1476}
|NULL        |Databricks-Runtime/17.0.x-aarch64-photon-scala2.13|
|0      |2025-08-17 14:57:13|76853913142180|727623mca062@mcet.in|CREATE OR REPLACE TABLE AS SELECT|{parti
tionBy -> [], clusterBy -> [], description -> NULL, isManaged -> true, properties -> {"delta.enableDeleti
onVectors":"true"}, statsOnLoad -> true}
|NULL|NULL    |0817-145439-7olt0x6c-v2n|NULL       |WriteSerializable|false        |{numFiles -> 1, numRe
movedFiles -> 0, numRemovedBytes -> 0, numOutputRows -> 4, numOutputBytes -> 1476}
|NULL        |Databricks-Runtime/17.0.x-aarch64-photon-scala2.13|
```