Exercise1_MasaCirkovic

January 16, 2025

1 Exercise 1: Introduction to Delta Lake with PySpark

This exercise demonstrates the basic functionalities of Delta Lake using PySpark. We'll work with a dataset on New York air quality (`air_quality_data.csv`) to showcase the following operations:

1. Reading and Writing Delta Tables
2. Update
3. Append
4. Delete
5. Time Travel
6. Vacuuming (Cleanup)

Helpful links:

https://docs.delta.io/latest/quick-start.html#read-data&language-python

https://docs.delta.io/latest/index.html

[1]:
```
# Install required libraries
!pip install delta-spark==3.0.0
```

```
Requirement already satisfied: delta-spark==3.0.0 in
/opt/conda/lib/python3.11/site-packages (3.0.0)
Requirement already satisfied: pyspark<3.6.0,>=3.5.0 in /usr/local/spark/python
(from delta-spark==3.0.0) (3.5.1)
Requirement already satisfied: importlib-metadata>=1.0.0 in
/opt/conda/lib/python3.11/site-packages (from delta-spark==3.0.0) (7.1.0)
Requirement already satisfied: zipp>=0.5 in /opt/conda/lib/python3.11/site-
packages (from importlib-metadata>=1.0.0->delta-spark==3.0.0) (3.17.0)
Requirement already satisfied: py4j==0.10.9.7 in /opt/conda/lib/python3.11/site-
packages (from pyspark<3.6.0,>=3.5.0->delta-spark==3.0.0) (0.10.9.7)
```

1.1 Step 1: Initializing PySpark and Delta Lake Environment

We'll configure the Spark session with Delta Lake support.

[2]:
```
from delta import configure_spark_with_delta_pip
from pyspark.sql import SparkSession

# Configure the Spark session with Delta support
```

```
builder = SparkSession.builder \
    .appName("Exercise1") \
    .config("spark.sql.extensions", "io.delta.sql.DeltaSparkSessionExtension") \
    .config("spark.sql.catalog.spark_catalog", "org.apache.spark.sql.delta.
  ↪catalog.DeltaCatalog") \
    .config("spark.jars.packages", "io.delta:delta-core_2.12:3.0.0")

# Create the Spark session
spark = configure_spark_with_delta_pip(builder).getOrCreate()

print("Spark session with Delta Lake configured successfully!")
spark
```

```
Spark session with Delta Lake configured successfully!
```

```
[2]: <pyspark.sql.session.SparkSession at 0x7f86240cd010>
```

Question: Why are we using `configure_spark_with_delta_pip` to configure Spark instead of just running it as is? (1p)

In order to use Spark with Delta Lake support, we need to configure it, as it is not the default configuration. configure_spark_with_delta_pip sets up everything for us

1.2 Step 2: Loading Air Quality Data (1p)

We'll load the air quality dataset (`air_quality_data.csv`) and inspect its structure. After that, we save it as a Spark DataFrame.

```
[3]: # Load CSV data
csv_path = "air_quality_data.csv"
df = spark.read.csv(csv_path, header=True, inferSchema=True)

# Inspect the structure of the DataFrame
print("Schema of the DataFrame:")
df.printSchema()

# Display the data
print("First 5 rows of the DataFrame:")
df.show(5)
```

```
Schema of the DataFrame:
root
 |-- Unique_ID: integer (nullable = true)
 |-- Name: string (nullable = true)
 |-- Measure: string (nullable = true)
 |-- Geo_Type_Name: string (nullable = true)
 |-- Geo_Place_Name: string (nullable = true)
 |-- Time_Period: string (nullable = true)
 |-- Start_Date: string (nullable = true)
 |-- Data_Value: double (nullable = true)
 |-- Air_Quality_Category: string (nullable = true)

First 5 rows of the DataFrame:
+---------+-----------------+-------+------------+-------------+-------------
+----------+----------+--------------------+
|Unique_ID|             Name|Measure|Geo_Type_Name|Geo_Place_Name|
Time_Period|Start_Date|Data_Value|Air_Quality_Category|
+---------+-----------------+-------+------------+-------------+-------------
+----------+----------+--------------------+
|   179772|        Emissions|Density|       UHF42|       Queens|
Other|    1/1/15|       0.3|                Good|
|   179785|        Emissions|Density|       UHF42|      Unknown|
Other|    1/1/15|       1.2|                Good|
|   178540|General Pollution|  Miles|       UHF42|      Unknown|Annual
Average|   12/1/11|       8.6|                Good|
|   178561|General Pollution|  Miles|       UHF42|       Queens|Annual
Average|   12/1/11|       8.0|                Good|
|   823217|General Pollution|  Miles|       UHF42|       Queens|
Summer|    6/1/22|       6.1|                Good|
+---------+-----------------+-------+------------+-------------+-------------
+----------+----------+--------------------+
only showing top 5 rows
```

[4]:
```python
print(df.columns)
print(df.describe())
```

```
['Unique_ID', 'Name', 'Measure', 'Geo_Type_Name', 'Geo_Place_Name',
'Time_Period', 'Start_Date', 'Data_Value', 'Air_Quality_Category']
DataFrame[summary: string, Unique_ID: string, Name: string, Measure: string,
Geo_Type_Name: string, Geo_Place_Name: string, Time_Period: string, Start_Date:
string, Data_Value: string, Air_Quality_Category: string]
```

1.3 Step 3: Writing Data to Delta Format (1p)

We will save the dataset as a Delta table for further operations.

[7]:
```python
# Save DataFrame to Delta format
delta_path = "delta_file"

df.write.format("delta").mode("overwrite").save(delta_path)

print(f"Data saved to Delta format at {delta_path}")
```

```
Data saved to Delta format at delta_file
```

2 Delta Lake Operations: Update, Append, Delete, and More (16p)

Now that we have saved our data as a delta table, let's run some basic operations on it.

- **Update**: Modifying rows based on conditions.
- **Append with Schema Evolution**: Adding new data while evolving the schema.
- **Delete**: Removing rows based on conditions.
- **Time Travel**: Querying historical versions of the table.
- **Vacuum**: Cleaning up unreferenced files to optimize storage.

We'll use a Delta table at `delta_path` to showcase these features.

2.1 1. Update Rows in the Delta Table (2p)

This operation demonstrates how to update specific rows in the Delta table. In this case, we replace the value `'Unknown'` in the `Geo_Place_Name` column with `'Not_Specified'`. (2p)

Code:

[8]:
```
from delta.tables import DeltaTable

# Load Delta Table
delta_table = DeltaTable.forPath(spark, delta_path)

# Update operation: Update rows where Geo_Place_Name is 'Unknown'
delta_table.update(
    condition="Geo_Place_Name = 'Unknown'",
    set={"Geo_Place_Name": "'Not_Specified'"}
)

print("Update completed!")

# Create a temporary view to query the Delta table
delta_table.toDF().createOrReplaceTempView("delta_table_view")

# Use spark.sql to visualize the changes
spark.sql("""
    SELECT Geo_Place_Name, COUNT(*) AS count
    FROM delta_table_view
    GROUP BY Geo_Place_Name
""").show()
```

```
Update completed!
+--------------+-----+
|Geo_Place_Name|count|
+--------------+-----+
|        Queens| 1466|
|      Brooklyn|  280|
| Staten Island|  368|
| Not_Specified|14546|
|     Manhattan|  439|
|         Bronx|  917|
+--------------+-----+
```

Question:
What happens when we update rows in a Delta table? How does Delta handle changes differently compared to a standard data format? (1p)

Compared to a standard data format, delta enforces ACID properties (Atomicity, Consistency, Isolation, Durability) on transactions. All updates are atomic and consistent, and if the update operation fails midway, the table remains in its original state in order to avoid partial or corrupt updates. Standard data formats do not possess such a guarantee. Delta also uses data versioning and maintains a transaction log that tracks all changes. This log allows for time travel and the ability to revert to previous versions of the table. Standard data formats do not support versioning, so updates overwrite the data and prior information (state) is lost. One other thing to note is that delta enforces schema consistency during updates, ensuring that only valid changes are applied. Standard data formats lack this feature and this can lead to corrupt data or mismatched data.

2.2 2. Append Data with Schema Evolution (2p)

Here, we demonstrate appending new rows to the Delta table. Additionally, we include a new column, `Source`, to showcase Delta Lake's schema evolution capabilities.

Steps: 1. Create a new DataFrame with an additional column (`Source`). 2. Use `mergeSchema=True` to allow schema evolution. 3. Append the new data to the Delta table. 4. Query the table using spark.sql to visualize changes

Code:

```
[9]: from pyspark.sql.functions import col
from delta.tables import DeltaTable

# Create new data directly
new_data = [
    (179808, "Emissions", "Density", "UHF42", "Queens", "Other", "2015-01-05",␣
 ↪0.7, "Good", "SensorA"),
    (179809, "Emissions", "Density", "UHF42", "Bronx", "Other", "2015-01-05", 1.
 ↪4, "Moderate", "SensorB")
]

# Convert the list to a DataFrame
new_data_df = spark.createDataFrame(new_data, [
    "Unique_ID", "Name", "Measure", "Geo_Type_Name", "Geo_Place_Name",
    "Time_Period", "Start_Date", "Data_Value", "Air_Quality_Category", "Source"
])

# Cast the Unique_ID column to LongType
```

```
new_data_df = new_data_df.withColumn("Unique_ID", col("Unique_ID").
 ↪cast("integer"))

# Append new data with schema evolution
new_data_df.write.format("delta").mode("append").option("mergeSchema", "true").
 ↪save(delta_path)
print("Append with schema evolution completed!")

# Load the Delta Table
delta_table = DeltaTable.forPath(spark, delta_path)

# Create a temporary view for querying
delta_table.toDF().createOrReplaceTempView("delta_table_view")

# Use spark.sql to visualize the updates
print("Visualizing updates in the Delta table:")
spark.sql("SELECT * FROM delta_table_view").show()
```

```
Append with schema evolution completed!
Visualizing updates in the Delta table:
+---------+-----------------+-------+-------------+--------------+--------------
+----------+----------+--------------------+------+
|Unique_ID|             Name|Measure|Geo_Type_Name|Geo_Place_Name|
Time_Period|Start_Date|Data_Value|Air_Quality_Category|Source|
+---------+-----------------+-------+-------------+--------------+--------------
+----------+----------+--------------------+------+
|   179772|        Emissions|Density|        UHF42|        Queens|
Other|     1/1/15|       0.3|                Good|  NULL|
|   179785|        Emissions|Density|        UHF42| Not_Specified|
Other|     1/1/15|       1.2|                Good|  NULL|
|   178540|General Pollution|  Miles|        UHF42| Not_Specified|Annual
Average|    12/1/11|       8.6|                Good|  NULL|
|   178561|General Pollution|  Miles|        UHF42|        Queens|Annual
Average|    12/1/11|       8.0|                Good|  NULL|
|   823217|General Pollution|  Miles|        UHF42|        Queens|
Summer|     6/1/22|       6.1|                Good|  NULL|
|   177910|General Pollution|  Miles|        UHF42| Not_Specified|
Summer|     6/1/12|      10.0|                Good|  NULL|
|   177952|General Pollution|  Miles|        UHF42| Not_Specified|
Summer|     6/1/13|       9.8|                Good|  NULL|
|   177973|General Pollution|  Miles|        UHF42|        Queens|
Summer|     6/1/13|       9.8|                Good|  NULL|
|   177931|General Pollution|  Miles|        UHF42|        Queens|
Summer|     6/1/12|       9.6|                Good|  NULL|
|   742274|General Pollution|  Miles|        UHF42|        Queens|
Summer|     6/1/21|       7.2|                Good|  NULL|
|   178582|General Pollution|  Miles|        UHF42| Not_Specified|Annual
```

```
Average|   12/1/12|       8.2|               Good|  NULL|
|   178583|General Pollution|  Miles|        UHF42| Not_Specified|Annual
Average|   12/1/12|       8.1|               Good|  NULL|
|   547477|General Pollution|  Miles|        UHF42|        Queens|Annual
Average|    1/1/17|       6.8|               Good|  NULL|
|   547417|General Pollution|  Miles|        UHF42| Not_Specified|Annual
Average|    1/1/17|       6.8|               Good|  NULL|
|   177784|General Pollution|  Miles|        UHF42| Not_Specified|
Summer|    6/1/09|      10.6|           Moderate|  NULL|
|   547414|General Pollution|  Miles|        UHF42| Not_Specified|Annual
Average|    1/1/17|       7.1|               Good|  NULL|
|   130413|        Emissions|Density|        UHF42| Not_Specified|
Other|    1/1/13|       0.9|               Good|  NULL|
|   130412|        Emissions|Density|        UHF42| Not_Specified|
Other|    1/1/13|       1.7|               Good|  NULL|
|   130434|        Emissions|Density|        UHF42|        Queens|
Other|    1/1/13|       0.0|               Good|  NULL|
|   410847|General Pollution|  Miles|        UHF42|        Queens|
Summer|    6/1/16|       6.9|               Good|  NULL|
+---------+-----------------+-------+-------------+--------------+-------------
+----------+----------+--------------------+------+
only showing top 20 rows
```

[10]:
```
spark.sql("SELECT * FROM delta_table_view WHERE Source IS NOT NULL").show()
```

```
+---------+---------+-------+-------------+--------------+-----------+----------
+----------+--------------------+-------+
|Unique_ID|     Name|Measure|Geo_Type_Name|Geo_Place_Name|Time_Period|Start_Date
|Data_Value|Air_Quality_Category| Source|
+---------+---------+-------+-------------+--------------+-----------+----------
+----------+--------------------+-------+
|   179809|Emissions|Density|        UHF42|         Bronx|
Other|2015-01-05|       1.4|            Moderate|SensorB|
|   179808|Emissions|Density|        UHF42|        Queens|
Other|2015-01-05|       0.7|                Good|SensorA|
+---------+---------+-------+-------------+--------------+-----------+----------
+----------+--------------------+-------+
```

Question:
When appending new data to a Delta table, what benefits does Delta provide compared to other data formats? (1p)

Compared to a standard data format, delta enforces ACID properties (Atomicity, Consistency, Isolation, Durability) on transactions. All inserts are atomic and consistent, and if the insert operation fails midway, the table remains in its original state in order to avoid partial or corrupt inserts. Standard data formats do not possess such a guarantee. Delta also uses data versioning and maintains a transaction log that tracks all changes. This log allows for time travel and the ability to revert to previous versions of the table. Standard data formats do not support versioning, so any new inserts create new data, without the ability to view it as it was in the past without those inserts. Delta also allows schema changes, such as adding new columns, while maintaining compatibility with the existing data. This makes it easier to handle changing data structures without needing to rewrite or manually manage schema changes.

2.3 3. Delete Rows from the Delta Table (2p)

This operation removes rows from the Delta table based on a condition. Here, we delete rows where the `Geo_Place_Name` column has the value `'Not_Specified'`.

Code:

[11]:
```python
from delta.tables import DeltaTable

# Load the Delta table
delta_table = DeltaTable.forPath(spark, delta_path)

# Delete rows where Geo_Place_Name is 'Not_Specified'
delta_table.delete("Geo_Place_Name = 'Not_Specified'")
print("Rows with Geo_Place_Name = 'Not_Specified' have been deleted!")

# Create a temporary view to query the Delta table
delta_table.toDF().createOrReplaceTempView("delta_table_view")

# Query to visualize the changes
spark.sql("""
    SELECT Geo_Place_Name, COUNT(*) AS count
    FROM delta_table_view
    GROUP BY Geo_Place_Name
""").show()
```

```
Rows with Geo_Place_Name = 'Not_Specified' have been deleted!
+--------------+-----+
|Geo_Place_Name|count|
+--------------+-----+
|        Queens| 1467|
|      Brooklyn|  280|
| Staten Island|  368|
|     Manhattan|  439|
|         Bronx|  918|
+--------------+-----+
```

Question:
What if we accidentally delete rows in a Delta table? Can we recover them? (1p)

Yes, we can recover them using time travel, which allows us to query previous versions of the table that existed in specific point in time. By using the Delta table's version history or a timestamp, we can retrieve the data as it was before the accidental deletion.

2.4 4. Time Travel: Query a Previous Version (2p)

Delta Lake allows you to query historical versions of the table using the `versionAsOf` option. Visualize the previous versions of the table and query one of the historical versions.

Code:

[12]:
```
from delta.tables import DeltaTable

# Load the Delta table
delta_table = DeltaTable.forPath(spark, delta_path)

# Show the full history of the table
history_df = delta_table.history()  # Returns a DataFrame of operations
print("Table History:")
history_df.show()
```

```
Table History:
+-------+--------------------+------+--------+---------+--------------------+---
-+--------+---------+-----------+--------------+-------------+-----------------
--+------------+--------------------+
|version|           timestamp|userId|userName|operation| operationParameters|
job|notebook|clusterId|readVersion|isolationLevel|isBlindAppend|
operationMetrics|userMetadata|          engineInfo|
+-------+--------------------+------+--------+---------+--------------------+---
-+--------+---------+-----------+--------------+-------------+-----------------
--+------------+--------------------+
|      3|2025-01-16 13:53:…|  NULL|    NULL|   DELETE|{predicate ->
["(…|NULL|    NULL|     NULL|          2|  Serializable|
false|{numRemovedFiles …|        NULL|Apache-Spark/3.5…|
|      2|2025-01-16 13:45:…|  NULL|    NULL|    WRITE|{mode -> Append,
…|NULL|    NULL|     NULL|          1|  Serializable|         true|{numFiles
-> 3, n…|        NULL|Apache-Spark/3.5…|
|      1|2025-01-16 13:45:…|  NULL|    NULL|   UPDATE|{predicate ->
["(…|NULL|    NULL|     NULL|          0|  Serializable|
false|{numRemovedFiles …|        NULL|Apache-Spark/3.5…|
|      0|2025-01-16 13:45:…|  NULL|    NULL|    WRITE|{mode ->
Overwrit…|NULL|    NULL|     NULL|       NULL|  Serializable|
false|{numFiles -> 1, n…|        NULL|Apache-Spark/3.5…|
+-------+--------------------+------+--------+---------+--------------------+---
-+--------+---------+-----------+--------------+-------------+-----------------
--+------------+--------------------+
```

[18]:
```
# Query the Delta table as of a previous version
df = spark.read.format("delta").option("versionAsOf", 1).load(delta_path)

# Display the data from a previous version
df.show()
```

```python
print("Column Source is not there!")
```

```
+---------+-----------------+-------+------------+-------------+-------------
+----------+----------+--------------------+
|Unique_ID|             Name|Measure|Geo_Type_Name|Geo_Place_Name|
Time_Period|Start_Date|Data_Value|Air_Quality_Category|
+---------+-----------------+-------+------------+-------------+-------------
+----------+----------+--------------------+
|   179772|        Emissions|Density|       UHF42|       Queens|
Other|    1/1/15|       0.3|                Good|
|   179785|        Emissions|Density|       UHF42| Not_Specified|
Other|    1/1/15|       1.2|                Good|
|   178540|General Pollution|  Miles|       UHF42| Not_Specified|Annual
Average|   12/1/11|       8.6|                Good|
|   178561|General Pollution|  Miles|       UHF42|       Queens|Annual
Average|   12/1/11|       8.0|                Good|
|   823217|General Pollution|  Miles|       UHF42|       Queens|
Summer|    6/1/22|       6.1|                Good|
|   177910|General Pollution|  Miles|       UHF42| Not_Specified|
Summer|    6/1/12|      10.0|                Good|
|   177952|General Pollution|  Miles|       UHF42| Not_Specified|
Summer|    6/1/13|       9.8|                Good|
|   177973|General Pollution|  Miles|       UHF42|       Queens|
Summer|    6/1/13|       9.8|                Good|
|   177931|General Pollution|  Miles|       UHF42|       Queens|
Summer|    6/1/12|       9.6|                Good|
|   742274|General Pollution|  Miles|       UHF42|       Queens|
Summer|    6/1/21|       7.2|                Good|
|   178582|General Pollution|  Miles|       UHF42| Not_Specified|Annual
Average|   12/1/12|       8.2|                Good|
|   178583|General Pollution|  Miles|       UHF42| Not_Specified|Annual
Average|   12/1/12|       8.1|                Good|
|   547477|General Pollution|  Miles|       UHF42|       Queens|Annual
Average|    1/1/17|       6.8|                Good|
|   547417|General Pollution|  Miles|       UHF42| Not_Specified|Annual
Average|    1/1/17|       6.8|                Good|
|   177784|General Pollution|  Miles|       UHF42| Not_Specified|
Summer|    6/1/09|      10.6|            Moderate|
|   547414|General Pollution|  Miles|       UHF42| Not_Specified|Annual
Average|    1/1/17|       7.1|                Good|
|   130413|        Emissions|Density|       UHF42| Not_Specified|
Other|    1/1/13|       0.9|                Good|
|   130412|        Emissions|Density|       UHF42| Not_Specified|
Other|    1/1/13|       1.7|                Good|
|   130434|        Emissions|Density|       UHF42|       Queens|
Other|    1/1/13|       0.0|                Good|
|   410847|General Pollution|  Miles|       UHF42|       Queens|
Summer|    6/1/16|       6.9|                Good|
+---------+-----------------+-------+-------------+--------------+--------------
+----------+----------+--------------------+
only showing top 20 rows

Column Source is not there!
```

Question: In what scenarios would you use Delta Lake's time travel over simply maintaining snapshots of data manually? (1p)

Delta Lake automatically tracks all changes to the data through its transaction log, which means we don't need to manually manage snapshots or versions. This reduces operational overhead and ensures that historical data is always available without additional setup, meaning it makes things simpler for us. Time travel allows us to query previous versions of the data without the need to store or manage separate copies of the dataset. This is more storage-efficient and avoids redundancy compared to maintaining full snapshots. Storing full snapshots manually can become expensive and difficult to manage over time, especially for large datasets. Delta Lake's time travel feature is optimized to only store changes, which reduces storage costs and complexity. Delta Lake guarantees ACID transactions, ensuring that all changes to the data are consistent and reliable. Manual snapshots may lack consistency and could be prone to partial data captures.

2.5 5. Vacuum: Clean Up Old Files

Vacuuming removes unreferenced files from the Delta table directory to optimize storage.

[19]:
```python
spark.conf.set("spark.databricks.delta.retentionDurationCheck.enabled", False)
delta_table.vacuum(retentionHours=0)
print("Vacuuming completed!")
```

```
Vacuuming completed!
```

[21]:
```python
# Load the Delta table
delta_table = DeltaTable.forPath(spark, delta_path)

# Show the full history of the table
history_df = delta_table.history()  # Returns a DataFrame of operations
print("Table History:")
history_df.show()
```

```
Table History:
+-------+--------------------+------+--------+------------+--------------------
+----+--------+---------+------------+-----------------+-------------+----------
---------+------------+--------------------+
|version|           timestamp|userId|userName|   operation| operationParameters|
job|notebook|clusterId|readVersion|   isolationLevel|isBlindAppend|
operationMetrics|userMetadata|          engineInfo|
+-------+--------------------+------+--------+------------+--------------------
+----+--------+---------+------------+-----------------+-------------+----------
---------+------------+--------------------+
|      5|2025-01-16 14:31:…|  NULL|    NULL|  VACUUM END|{status ->
COMPLE…|NULL|    NULL|     NULL|          4|SnapshotIsolation|
true|{numDeletedFiles …|        NULL|Apache-Spark/3.5…|
|      4|2025-01-16 14:31:…|  NULL|    NULL|VACUUM
START|{retentionCheckEn…|NULL|    NULL|     NULL|
3|SnapshotIsolation|         true|{numFilesToDelete…|        NULL|Apache-
Spark/3.5…|
|      3|2025-01-16 13:53:…|  NULL|    NULL|      DELETE|{predicate ->
["(…|NULL|    NULL|     NULL|          2|     Serializable|
false|{numRemovedFiles …|        NULL|Apache-Spark/3.5…|
|      2|2025-01-16 13:45:…|  NULL|    NULL|       WRITE|{mode -> Append,
…|NULL|    NULL|     NULL|          1|     Serializable|
true|{numFiles -> 3, n…|        NULL|Apache-Spark/3.5…|
|      1|2025-01-16 13:45:…|  NULL|    NULL|      UPDATE|{predicate ->
["(…|NULL|    NULL|     NULL|          0|     Serializable|
false|{numRemovedFiles …|        NULL|Apache-Spark/3.5…|
|      0|2025-01-16 13:45:…|  NULL|    NULL|       WRITE|{mode ->
Overwrit…|NULL|    NULL|     NULL|       NULL|     Serializable|
false|{numFiles -> 1, n…|        NULL|Apache-Spark/3.5…|
+-------+--------------------+------+--------+------------+--------------------
+----+--------+---------+-----------+-----------------+-------------+-----------
---------+------------+--------------------+
```

Question:
What is the default retention period for Delta table vacuuming, and why does it matter? (1p)

The default retention period for Delta table vacuuming is 7 days. The 7-day retention period means that Delta Lake will only delete data files that are older than 7 days, ensuring that enough time is given for time travel. While retaining data for time travel is important, the vacuum operation helps to free up storage space by cleaning up outdated files that are no longer required.

2.5.1 6. When finished, remember to close the spark session.

[22]:
```
spark.stop()
```