Exercise1_MeteHarunAkcay

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 0x7f51ca1c7050>
```

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

This way ensures seamless integration of Delta Lake features with the Spark environment. It includes necessary dependencies automatically, makes the setup process simpler and it dynamically resolves dependencies to work with the current Spark and Delta Lake versions.

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.

[4]:
```
# Load CSV data
csv_path = "../shared/air_quality_data.csv"
df = spark.read.csv(csv_path, header=True, inferSchema=True)

# Display the data
df.show()
```

```
+---------+----------------+-------+------------+-------------+-------------
+----------+----------+--------------------+
|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|
|   177910|General Pollution|  Miles|        UHF42|       Unknown|
Summer|    6/1/12|      10.0|                Good|
|   177952|General Pollution|  Miles|        UHF42|       Unknown|
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|       Unknown|Annual
Average|   12/1/12|       8.2|                Good|
|   178583|General Pollution|  Miles|        UHF42|       Unknown|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|       Unknown|Annual
Average|    1/1/17|       6.8|                Good|
|   177784|General Pollution|  Miles|        UHF42|       Unknown|
Summer|    6/1/09|      10.6|            Moderate|
|   547414|General Pollution|  Miles|        UHF42|       Unknown|Annual
Average|    1/1/17|       7.1|                Good|
|   130413|        Emissions|Density|        UHF42|       Unknown|
Other|    1/1/13|       0.9|               Good|
|   130412|        Emissions|Density|        UHF42|       Unknown|
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
```

[5]:
```
df.printSchema()
print(f"Number of rows: {df.count()}")
```

```
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)

Number of rows: 18016
```

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

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

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

df.write.format("delta").mode("overwrite").option("mergeSchema", "true").
 ↪save(delta_path)

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

```
Data saved to Delta format at mete_delta_table
```

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'",  # Condition to match rows
    set={"Geo_Place_Name": "'Not_Specified'"}  # Update value
)
```

```
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)

Unlike a standard data format, DeltaLake doesn't overwrites versions, it maintains versions for updates and deletes, which is called time travel. It supports ACID transactions. Only the affected files are rewritten, not the whole table. It maintains data wuality and adapts to schema changes.

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
from pyspark.sql.types import IntegerType

# 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"
])
new_data_df = new_data_df.withColumn("Unique_ID", col("Unique_ID").
 ↪cast(IntegerType()))

# 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
spark.sql("""
    SELECT Geo_Place_Name, COUNT(*) AS count, MAX(Source) AS Source
    FROM delta_table_view
    GROUP BY Geo_Place_Name
""").show()
```

```
Append with schema evolution completed!
+--------------+-----+-------+
|Geo_Place_Name|count| Source|
+--------------+-----+-------+
|         Bronx|  918|SensorB|
|      Brooklyn|  280|   NULL|
|     Manhattan|  439|   NULL|
| Not_Specified|14546|   NULL|
|        Queens| 1467|SensorA|
| Staten Island|  368|   NULL|
+--------------+-----+-------+
```

Question:
When appending new data to a Delta table, what benefits does Delta provide compared to other

data formats? (1p)

By saying mergeSchema=true, DeltaLake allows schema to evolve automatically. Atomicity, ensures that if a failure occurs during append, the table remains in a consistent state so that nothing changes. Every append creates a new version that we can roll back to a previous one using time travel. DeltaLake allows multiple users to append data concurrently without conflicts.

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:

[10]:
```
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. As I mentioned in the previous answers, DeltaLake has a feature called time travel, which allows us to roll back previous states. DeltaLake maintains a transaction log that records

deleted. So if we go back to a previous state after deleting rows, we can restore them.

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:

[11]:
```
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 13:42:…|  NULL|    NULL|   DELETE|{predicate ->
["(…|NULL|    NULL|     NULL|          2|  Serializable|
false|{numRemovedFiles …|        NULL|Apache-Spark/3.5…|
|      2|2025-01-16 13:42:…|  NULL|    NULL|    WRITE|{mode -> Append,
…|NULL|    NULL|     NULL|          1|  Serializable|         true|{numFiles
-> 3, n…|        NULL|Apache-Spark/3.5…|
|      1|2025-01-16 13:41:…|  NULL|    NULL|   UPDATE|{predicate ->
["(…|NULL|    NULL|     NULL|          0|  Serializable|
false|{numRemovedFiles …|        NULL|Apache-Spark/3.5…|
|      0|2025-01-16 13:41:…|  NULL|    NULL|    WRITE|{mode ->
Overwrit…|NULL|    NULL|     NULL|       NULL|  Serializable|
false|{numFiles -> 1, n…|        NULL|Apache-Spark/3.5…|
+-------+-------------------+------+--------+---------+--------------------+---
-+--------+---------+-----------+--------------+-------------+-----------------
--+------------+--------------------+
```

[12]:
```
# Query the Delta table as of a previous version
previous_version_df = spark.read.format("delta").option("versionAsOf", 1).
  ↪load(delta_path)
```

```
# Display the data from a previous version
print("Data from version 1:")
previous_version_df.show()
```

```
Data from version 1:
+---------+----------------+-------+------------+-------------+-------------
+----------+----------+--------------------+
|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
```

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

Time travel is better than maintaining data snapshots manually becuase its faster, easier and more organized. With time travel, we can quickly go back to any version of our data or see how it looked at a specific time. We don't need to save separate copies of our data, so it saves space and avoids confusion with multiple files. It's also great for fixing mistakes, like accidentally deleting or changing data, because we can easily recover it. Plus, Delta automatically keeps track of all changes, and it's much easier to audit or check the history of the data.

2.5 5. Vacuum: Clean Up Old Files

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

[13]:
```
spark.conf.set("spark.databricks.delta.retentionDurationCheck.enabled", False)
delta_table.vacuum(retentionHours=0)

print("Vacuuming completed!")
```

```
Vacuuming completed!
```

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

7 days. It matters since if we set this value too low, we lose the ability to recover past versions of our data. If we set it too high, then it is not storage optimization anymore. So we have to pick a value that is balanced.

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

[14]:
```
spark.stop()
```

[ ]:
```

```