Exercise1

January 21, 2025

Name: Tesfay Tesfay
Email: tesfay.tesfay@abo.fi
S_ID: 2402263

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

[15]:
```
# 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.

[16]:
```python
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!
```

[16]:
```
<pyspark.sql.session.SparkSession at 0x7eff33fb6890>
```

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

Ans: To insure that the Spark session is correctly configured with Delta Lake-specific configurations which are not included by default. This allows Spark to handle Delta Lake operations.

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.

[17]:
```python
# Load CSV data
csv_path = "../shared/air_quality_data.csv"
df = spark.read.csv(csv_path, header=True)

# Display the data
df.show()

# Get the number of rows
row_count = df.count()

# Get the number of columns
column_count = len(df.columns)
print(f"Shape: {row_count} x {column_count}")
```

```
+---------+-----------------+-------+------------+-------------+-------------
+----------+----------+--------------------+
|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|                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|                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|                Good|
|   410847|General Pollution|  Miles|       UHF42|       Queens|
Summer|    6/1/16|       6.9|                Good|
+---------+-----------------+-------+------------+-------------+-------------
+----------+----------+--------------------+
only showing top 20 rows

Shape: 18016 x 9
```

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

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

[18]:
```
# Save DataFrame to Delta format
delta_path = "delta-table-v-02"

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

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

```
Data saved to Delta format at delta-table-v-02
```

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:

[19]:
```
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)

Ans: Delta Lake saves the changes as a new version of the data instead of overwriting the existing data. This ensures ACID properties compliance and maintains a transaction log. Compared to a standard data format, Delta Lake enables features like time travel, data lineage, and recovery, which are not available in traditional data formats.

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:

[20]:
```
from pyspark.sql.functions import col
from delta.tables import DeltaTable

# Create new data directly
# Values of column 1 and 8 are changed to String
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"
])

# Due to data type incompatibility the following two column data types are␣
↪changed to string to match existing Delta table schema
new_data_df = new_data_df.withColumn("Unique_ID", col("Unique_ID").
↪cast("string"))

new_data_df = new_data_df.withColumn("Data_Value", col("Data_Value").
↪cast("string"))

# 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 * FROM delta_table_view").show()

# Check if the dimention has been changed

# Get the number of rows
row_count = delta_table.toDF().count()

# Get the number of columns
column_count = len(delta_table.toDF().columns)

print(f"Shape: {row_count} x {column_count}")
```

```
Append with schema evolution completed!
+---------+-----------------+-------+-------------+--------------+--------------
+----------+----------+--------------------+-------+
|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|                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|                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|                Good|  NULL|
|   410847|General Pollution|  Miles|        UHF42|       Queens|
Summer|    6/1/16|       6.9|                Good|  NULL|
+---------+----------------+-------+------------+-------------+-------------
+----------+----------+-------------------+------+
only showing top 20 rows

Shape: 18018 x 10
```

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

Ans: Delta Lake supports schema evolution and ensures ACID transactions during appends. This minimizes the risk of data corruption and allows for the seamless integration of new columns or data types, making it easier to adapt to evolving data schemas without disrupting existing data pipelines.

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:

[21]:
```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()

# Get the number of rows
row_count = delta_table.toDF().count()

# Get the number of columns
column_count = len(delta_table.toDF().columns)
print(f"Shape: {row_count} x {column_count}")
```

```
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|
+--------------+-----+

Shape: 3472 x 10
```

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

Ans: Yes, Delta Lake's time travel feature allows us to query historical versions of the table and recover accidentally deleted rows.

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:

[22]:
```python
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|2025-01-21 16:40:…|  NULL|    NULL|   DELETE|{predicate ->
["(…|NULL|    NULL|     NULL|          2|  Serializable|
false|{numRemovedFiles …|        NULL|Apache-Spark/3.5…|
|      2|2025-01-21 16:40:…|  NULL|    NULL|    WRITE|{mode -> Append,
…|NULL|    NULL|     NULL|          1|  Serializable|         true|{numFiles
-> 3, n…|        NULL|Apache-Spark/3.5…|
|      1|2025-01-21 16:40:…|  NULL|    NULL|   UPDATE|{predicate ->
["(…|NULL|    NULL|     NULL|          0|  Serializable|
false|{numRemovedFiles …|        NULL|Apache-Spark/3.5…|
|      0|2025-01-21 16:40:…|  NULL|    NULL|    WRITE|{mode ->
```

```
Overwrit…|NULL|     NULL|      NULL|        NULL|  Serializable|
false|{numFiles -> 1, n…|        NULL|Apache-Spark/3.5…|
+-------+--------------------+------+--------+---------+--------------------+---
-+---------+---------+-----------+--------------+-------------+------------------
--+------------+--------------------+
```

[23]:

```python
# 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()

# Get the number of rows
row_count = df.count()

# Get the number of columns
column_count = len(df.columns)
print(f"Shape: {row_count} x {column_count}")
```

```
+---------+-----------------+-------+-------------+--------------+-------------
+----------+----------+--------------------+
|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|                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|                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|              Good|
|   410847|General Pollution|  Miles|        UHF42|        Queens|
Summer|    6/1/16|       6.9|               Good|
+---------+-----------------+-------+-------------+--------------+-------------
+----------+----------+--------------------+
only showing top 20 rows

Shape: 18016 x 9
```

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

Ans: I would use Delta Lake's time travel in the following scenarios over manual snapshots:

- Data Recovery: Quickly revert to previous versions without restoring snapshots manually.
- Auditing and Compliance: Query historical data easily for audits, avoiding snapshot tracking.
- Storage Efficiency: Store only data changes, unlike snapshots, which duplicate datasets.
- Version Management: Automatically track data versions without manual effort.

2.5 5. Vacuum: Clean Up Old Files

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

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

Ans: It is 7 days.
And, it matters because it ensures older versions and unreferenced files are retained long enough to support operations like **time travel** , **auditing**, and **recovery** before they are permanently deleted to optimize disk space.

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

[24]:
```
spark.stop()
```