You can always find more details about AQE from Microsoft’s documentation which can be found here: https://docs.microsoft.com/en-us/azure/databricks/spark/latest/spark-sql/aqe

Here is the Scala code that you will need to run to generate the 2019 NYC Taxi dataset into a data frame. This code builds upon the code in the previous section with the addition of the ID column using ‘monotonically\_increasing\_id’.

%scala

import org.apache.spark.sql.functions.\_

val Data = "/databricks-datasets/nyctaxi/tripdata/yellow/yellow\_tripdata\_2019-\*"

val SchemaDF = spark.read.format("csv").option("header", "true").option("inferSchema", "true").load("/databricks-datasets/nyctaxi/tripdata/yellow/yellow\_tripdata\_2019-02.csv.gz")

val df = spark.read.format("csv").option("header", "true").schema(SchemaDF.schema).load(Data)

val nyctaxiDF\_stage = df

.withColumn("VendorID", explode(array((1 until 14).map(lit): \_\*)))

.selectExpr(df.columns: \_\*)

val nyctaxiDF = nyctaxiDF\_stage.withColumn("ID", monotonically\_increasing\_id)

Next, you could simply run the following code nyctaxiDF.write.format(“delta”).saveAsTable(“dlhcore.nyctaxi\_A”) to save your dataframe as a Delta formatted table.

Run the following Scala code twice. For the first iteration, run the code with ‘nyctaxi\_A’ specified in the file path. And for the second iteration, run the code with ‘nyctaxi\_B’ specified in the file path. This will persist two large datasets, over 1 billion rows, in your specified ADLS gen2 folder path. These two datasets will be used to join to each other within a query.

%scala

val nyctaxiDF\_delta = nyctaxiDF.write

.format("delta")

.mode("overwrite")

.save("dbfs:/mnt/rcpdlhcore/datalakehouse/dlhcore/raw/delta/nyctaxi\_A")

Similarly, run the following SQL code twice, once for nyctaxi\_A and the second for nyctaxi\_B to create two Delta tables that you will use in query joins.

CREATE TABLE nyctaxi\_A

USING DELTA

LOCATION 'dbfs:/mnt/rcpdlhcore/datalakehouse/dlhcore/raw/delta/nyctaxi\_A'

To test performance of AQE turned off, go ahead and run the following command to set spark.sql.adaptive.enabled = false;.

Here is the code that you will need to run to explain the physical plan of the SQL query.

EXPLAIN FORMATTED

SELECT a.VendorID, SUM(a.total\_amount) as sum\_total

FROM nyctaxi\_A a

JOIN nyctaxi\_B b ON a.ID = b.ID

WHERE a.tpep\_pickup\_datetime BETWEEN '2019-05-01 00:00:00' AND '2019-05-03 00:00:00'

GROUP BY a.VendorID

ORDER BY sum\_total DESC;