Create an Apache Spark table

Note:

You are not required to complete the processes, tasks, activities, or steps presented in this example. The various samples provided are for illustrative purposes only and it's likely that if you try this out you will encounter issues in your system.

In this exercise, you will create an Apache Spark table.

1. The Apache Spark engine can analyze the Parquet files and infer the schema. To perform this analysis, enter the following code in the new cell and **run** it:

```
data_path.printSchema()
```

Your output should look as follows:

```
root
    |-- TransactionId: string (nullable = true)
    |-- CustomerId: integer (nullable = true)
    |-- ProductId: short (nullable = true)
    |-- Quantity: short (nullable = true)
    |-- Price: decimal(29,2) (nullable = true)
    |-- TotalAmount: decimal(29,2) (nullable = true)
    |-- TransactionDate: integer (nullable = true)
    |-- ProfitAmount: decimal(29,2) (nullable = true)
    |-- Hour: byte (nullable = true)
    |-- Minute: byte (nullable = true)
    |-- StoreId: short (nullable = true)
```

Apache Spark evaluates the file contents to infer the schema. This automatic inference is sufficient for data exploration and most transformation tasks.

However, when you load data to an external resource like a SQL pool table, sometimes you need to declare your own schema and apply that to the dataset. For now, the schema looks good.

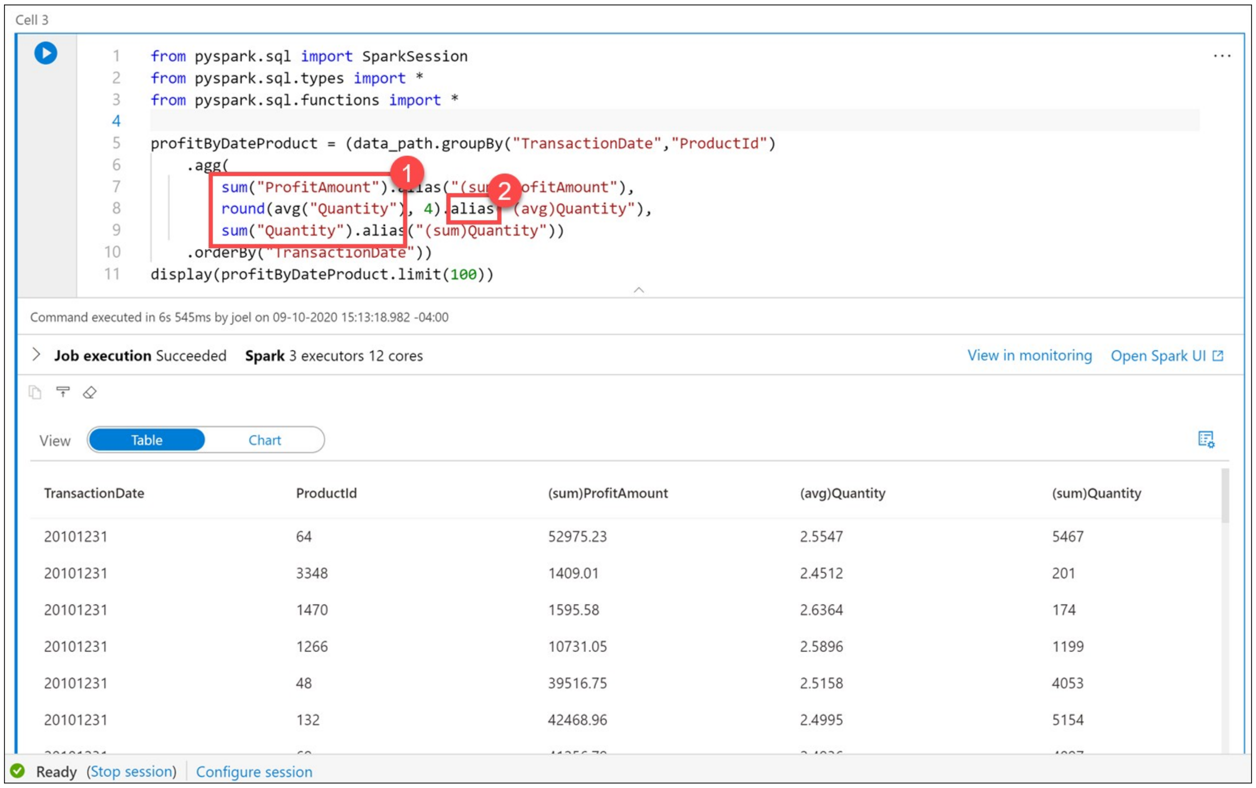
2. Next, use the DataFrame to use aggregates and grouping operations in order to better understand the data. Create a new cell and enter the following, then **run** the cell:

```
from pyspark.sql import SparkSession
from pyspark.sql.types import *
from pyspark.sql.functions import *

profitByDateProduct = (data_path.groupBy("TransactionDate","ProductId")
    .agg(
        sum("ProfitAmount").alias("(sum)ProfitAmount"),
        round(avg("Quantity"), 4).alias("(avg)Quantity"),
        sum("Quantity").alias("(sum)Quantity"))
    .orderBy("TransactionDate"))
display(profitByDateProduct.limit(100))
```

Note: We import required Python libraries to use aggregation functions and types defined in the schema to successfully execute the query.

The output shows the same data we saw in the chart above, but now with **sum** and **avg** aggregates **(1)**. Notice that we use the **alias** method **(2)** to change the column names.



```python
from pyspark.sql import SparkSession
from pyspark.sql.types import *
from pyspark.sql.functions import *

profitByDateProduct = (data_path.groupBy("TransactionDate","ProductId")
    .agg(
        sum("ProfitAmount").alias("(sum)ProfitAmount"),
        round(avg("Quantity"), 4).alias("(avg)Quantity"),
        sum("Quantity").alias("(sum)Quantity"))
    .orderBy("TransactionDate"))
display(profitByDateProduct.limit(100))
```

```
Command executed in 6s 545ms by joel on 09-10-2020 15:13:18.982 -04:00
```

TransactionDate	ProductId	(sum)ProfitAmount	(avg)Quantity	(sum)Quantity
20101231	64	52975.23	2.5547	5467
20101231	3348	1409.01	2.4512	201
20101231	1470	1595.58	2.6364	174
20101231	1266	10731.05	2.5896	1199
20101231	48	39516.75	2.5158	4053
20101231	132	42468.96	2.4995	5154

The aggregates output is displayed.