DataFrames with SQL

June 21, 2020

1 DataFrames with SQL

We talked about before that Spark is a unified computing engine. Among many things, it also means that Spark is not tied to one specific language. You can have access to the same transformation supported by the engine regardless if you're using the DF API or SQL.

When you express your logic in SQL, spark will compile the expression into an underlying execution plan built from primitives supported by the core engine. After that the job will be executed in the same way as we would have used the legacy spark API.

Let's see an example on how we can use SQL expressions with Dataframes.

[1]:
```
import pyspark
from pyspark.sql import SparkSession

sc = pyspark.SparkContext()
spark = SparkSession(sc)
```

First, we need to create a dataframe in the same way we did in the previous module. We'll use the same sample dataset to create the DF with automated schema inference.

[2]:
```
flightsDF = spark \
    .read \
    .option("inferSchema", "true") \
    .option("header", "true") \
    .csv('../data/flights.csv')
```

The next step is to turn the DF into a table or a view. This can be done with one simple method:

[5]:
```
# create a View on top of the DF which can be used for querying with SQL
flightsDF.createOrReplaceTempView("flights")
```

[8]:
```
res = spark.sql("""
SELECT DEST_COUNTRY_NAME, sum(count) as total_destination
FROM flights
GROUP BY DEST_COUNTRY_NAME
ORDER BY total_destination DESC
LIMIT 5
""")
```

```
res.show()
```

```
+-----------------+-----------------+
|DEST_COUNTRY_NAME|total_destination|
+-----------------+-----------------+
|    United States|           411352|
|           Canada|             8399|
|           Mexico|             7140|
|   United Kingdom|             2025|
|            Japan|             1548|
+-----------------+-----------------+
```

As a comparison, let's see how the same expression would like with using the DataFrames API

[14]:
```
from pyspark.sql.functions import desc
resDF = flightsDF \
    .groupBy("DEST_COUNTRY_NAME") \
    .sum("count") \
    .withColumnRenamed("sum(count)", "total_destination") \
    .sort(desc("total_destination")) \
    .limit(5)
resDF.show()
```

```
+-----------------+-----------------+
|DEST_COUNTRY_NAME|total_destination|
+-----------------+-----------------+
|    United States|           411352|
|           Canada|             8399|
|           Mexico|             7140|
|   United Kingdom|             2025|
|            Japan|             1548|
+-----------------+-----------------+
```

Let's compare the physical execution plan created by the two queries. We can see that the same transformations are triggered in the background.

[17]:
```
res.explain()
resDF.explain()
```

```
== Physical Plan ==
TakeOrderedAndProject(limit=5, orderBy=[total_destination#32L DESC NULLS LAST],
output=[DEST_COUNTRY_NAME#10,total_destination#32L])
+- *(2) HashAggregate(keys=[DEST_COUNTRY_NAME#10], functions=[sum(cast(count#12
as bigint))])
   +- Exchange hashpartitioning(DEST_COUNTRY_NAME#10, 200)
      +- *(1) HashAggregate(keys=[DEST_COUNTRY_NAME#10],
functions=[partial_sum(cast(count#12 as bigint))])
```

```
         +- *(1) FileScan csv [DEST_COUNTRY_NAME#10,count#12] Batched: false,
Format: CSV, Location: InMemoryFileIndex[file:/home/andras/spark_jupyter/spark_t
raining_baseline/data/flights.csv], PartitionFilters: [], PushedFilters: [],
ReadSchema: struct<DEST_COUNTRY_NAME:string,count:int>
== Physical Plan ==
TakeOrderedAndProject(limit=5, orderBy=[total_destination#82L DESC NULLS LAST],
output=[DEST_COUNTRY_NAME#10,total_destination#82L])
+- *(2) HashAggregate(keys=[DEST_COUNTRY_NAME#10], functions=[sum(cast(count#12
as bigint))])
   +- Exchange hashpartitioning(DEST_COUNTRY_NAME#10, 200)
      +- *(1) HashAggregate(keys=[DEST_COUNTRY_NAME#10],
functions=[partial_sum(cast(count#12 as bigint))])
         +- *(1) FileScan csv [DEST_COUNTRY_NAME#10,count#12] Batched: false,
Format: CSV, Location: InMemoryFileIndex[file:/home/andras/spark_jupyter/spark_t
raining_baseline/data/flights.csv], PartitionFilters: [], PushedFilters: [],
ReadSchema: struct<DEST_COUNTRY_NAME:string,count:int>
```

[ ]: