THINKBIG ANALYTICS

A TERADATA COMPANY

SparkSQL in Python

Lab05: Basic Pyspark SparkSQL Walkthrough

At this point, we assume that you are running a `spark-shell` on your cluster. You should type the commands after the `python>` prompt and you should see the outputs shown.

Begin by reading in a dataset. Hint: It's not exactly big data. In fact, it just looks like this:

```
[hadoop@ip-172-31-57-176 sparkclass]$ hdfs dfs -cat /data/spark-resources-data/people.json
{"name":"Michael"}
{"name":"Andy", "age":30}
{"name":"Justin", "age":19}
```

That said, let's read it in so we have some data to apply SparkSQL to.

```
df = spark.read.json("/data/spark-resources-data/people.json")
```

Now that we have people read in as a DataFrame, we can now query it in different ways.

```
// Displays the content of the DataFrame to stdout
df.show()
+----+-------+
| age|   name|
+----+-------+
|null|Michael|
|  30|   Andy|
|  19| Justin|
+----+-------+
// Print the schema in a tree format
df.printSchema()
root
 |-- age: long (nullable = true)
 |-- name: string (nullable = true)

// Select only the "name" column
df.select("name").show()
+-------+
|   name|
+-------+
|Michael|
|   Andy|
| Justin|
+-------+

// Select everybody, but increment the age by 1
df.select(df.name, df.age + 1).show()
+-------+---------+
|   name|(age + 1)|
+-------+---------+
|Michael|     null|
|   Andy|       31|
| Justin|       20|
+-------+---------+

// Select people older than 21
df.filter(df("age") > 21).show()
+---+----+
|age|name|
+---+----+
| 30|Andy|
+---+----+

// Count people by age
df.groupBy("age").count().show()
+----+-----+
| age|count|
+----+-----+
|  19|    1|
|null|    1|
|  30|    1|
+----+-----+
```

Read A Table From The Hive Warehouse

If you ran the Hive-Tables2 and Hive-LoadingData queries in the Hive exercises, you should have a Hive table called stocks which has a lot of data loaded into 4 partitions

Remember that you if you read that into a database with your name on it (DB), you have to change to that DB using the `use yourname;` command. The following will simply assume you are using the default Hive database.

```
>>> spark.sql("USE default")           ## Select my database
DataFrame[]
>>>
>>> stocks = spark.sql("SELECT * FROM STOCKS")
>>> stocks.show(5)  ## Will only show the first 20 rows
+----------+----------+----------+----------+-----------+-------+---------------+-----+------+
|       ymd|price_open|price_high| price_low|price_close| volume|price_adj_close|exchg|symbol|
+----------+----------+----------+----------+-----------+-------+---------------+-----+------+
|2015-06-22|167.649994|168.339996|167.199997| 167.729996|2210700|     167.729996| NYSE|   IBM|
|2015-06-19|167.619995|168.419998|166.770004| 166.990005|6971200|     166.990005| NYSE|   IBM|
|2015-06-18|167.050003|168.720001|167.050003|     168.25|3329100|         168.25| NYSE|   IBM|
|2015-06-17|       167|167.850006|166.100006| 167.169998|2861100|     167.169998| NYSE|   IBM|
|2015-06-16|166.330002|167.399994|165.910004| 166.839996|3246900|     166.839996| NYSE|   IBM|
+----------+----------+----------+----------+-----------+-------+---------------+-----+------+
only showing top 5 rows

>>> stocks.count() ## How many rows? Should be 40,547
2131092
>>>
```

Reading In A Large Dataset

We'd like to read in an FAA dataset that has all the US airline flights between 2010 and 2016. You could do that in SQL with the following command, but I don't recommend it; it has a lot of columns.

```
// This is one way to create a s3flights dataframe.
// val creates3flightsSQL = "CREATE EXTERNAL TABLE s3flights (
    Year INT,
    Quarter INT,
    Month INT,
    DayofMonth INT,
    DayOfWeek INT,
    FlightDate STRING,
    UniqueCarrier STRING,
    AirlineID INT,
    Carrier STRING,
    TailNum STRING,
    FlightNum INT,
    OriginAirportID INT,
    OriginAirportSeqID INT,
    OriginCityMarketID INT,
    Origin STRING,
    OriginCityName STRING,
    OriginState STRING,
    OriginStateFips INT,
    OriginStateName STRING,
    OriginWac INT,
    DestAirportID INT,
    DestAirportSeqID INT,
    DestCityMarketID INT,
    Dest STRING,
    DestCityName STRING,
    DestState STRING,
    DestStateFips INT,
    DestStateName STRING,
    DestWac INT,
    CRSDepTime INT,
    DepTime INT,
    DepDelay INT,
    DepDelayMinutes INT,
    DepDel15 INT,
    DepartureDelayGroups INT,
    DepTimeBlk STRING,
    TaxiOut INT,
    WheelsOff INT,
    WheelsOn INT,
    TaxiIn INT,
    CRSArrTime INT,
    ArrTime INT,
    ArrDelay INT,
    ArrDelayMinutes INT,
    ArrDel15 INT,
    ArrivalDelayGroups INT,
    ArrTimeBlk STRING,
    Cancelled TINYINT,
    CancellationCode STRING,
    Diverted TINYINT,
    CRSElapsedTime INT,
    ActualElapsedTime INT,
    AirTime INT,
    Flights INT,
    Distance INT,
     DistanceGroup INT,
    CarrierDelay INT,
    WeatherDelay INT,
    NASDelay INT,
    SecurityDelay INT,
    LateAircraftDelay INT,
    FirstDepTime INT,
    TotalAddGTime INT,
    LongestAddGTime INT,
    DivAirportLandings INT,
    DivReachedDest INT,
    DivActualElapsedTime INT,
    DivArrDelay INT,
    DivDistance INT,
    Div1Airport STRING,
    Div1AirportID INT,
    Div1AirportSeqID INT,
    Div1WheelsOn INT,
    Div1TotalGTime INT,
    Div1LongestGTime INT,
    Div1WheelsOff INT,
    Div1TailNum STRING,
    Div2Airport STRING,
    Div2AirportID INT,
    Div2AirportSeqID INT,
    Div2WheelsOn INT,
    Div2TotalGTime INT,
    Div2LongestGTime INT,
    Div2WheelsOff INT,
    Div2TailNum STRING,
    Div3Airport STRING,
    Div3AirportID INT,
    Div3AirportSeqID INT,
    Div3WheelsOn INT,
    Div3TotalGTime INT,
    Div3LongestGTime INT,
    Div3WheelsOff INT,
    Div3TailNum STRING,
    Div4Airport STRING,
    Div4AirportID INT,
    Div4AirportSeqID INT,
    Div4WheelsOn INT,
    Div4TotalGTime INT,
    Div4LongestGTime INT,
    Div4WheelsOff INT,
    Div4TailNum STRING,
    Div5Airport STRING,
    Div5AirportID INT,
    Div5AirportSeqID INT,
    Div5WheelsOn INT,
    Div5TotalGTime INT,
    Div5LongestGTime INT,
    Div5WheelsOff INT,
    Div5TailNum STRING
)
STORED AS PARQUET LOCATION 's3://think.big.academy.aws/ontime/parquet'"
//val s3flights = hiveContext.sql(creates3flightsSQL)
```

The reason I don't recommend this approach is that SQL requires us to specify all the columns and their types. That information is already in the parquet file, so it seems redundant.

We can take advantage of SparkSQL's intelligence by simply loading the parquet file as a dataframe and letting it infer the table schema.

We happen to already have this file in a slightly trimmed down form in HDFS. It's at `hdfs:///data/flightdata/parquet-trimmed` .

```
s3flights = spark.read.parquet("hdfs:///data/flightdata/parquet-trimmed")
```

Now let's select only some of the fields. Further, we're going to filter it to only be the year 2013. Note that the use of a data pipeline allows Spark to lazily evaluate s3flights and only pull those records that are of interest into memory.

The good news: this runs pretty fast, much faster than the Hive equivalent.

```
p>>> flights = s3flights.select("year", "month", "dayofmonth", \
... "carrier", "tailnum","actualelapsedtime", \
... "origin", "dest", "deptime", "arrdelayminutes"). \
... filter(s3flights.year == 2013)
>>> flights.show(5)
+----+-----+----------+-------+-------+-----------------+------+----+-------+---------------+
|year|month|dayofmonth|carrier|tailnum|actualelapsedtime|origin|dest|deptime|arrdelayminutes|
+----+-----+----------+-------+-------+-----------------+------+----+-------+---------------+
|2013|    1|        18|     DL| N325US|              184|   PHL| MSP|    758|              0|
|2013|    1|        18|     DL| N325NB|              172|   FLL| LGA|    657|              0|
|2013|    1|        18|     DL| N649DL|              190|   LGA| ATL|   1657|             24|
|2013|    1|        18|     DL| N130DL|              251|   SLC| ATL|    953|             43|
|2013|    1|        18|     DL| N651DL|              171|   BOS| ATL|    711|              0|
+----+-----+----------+-------+-------+-----------------+------+----+-------+---------------+
only showing top 5 rows

>>> flights.printSchema()
root
 |-- year: integer (nullable = true)
 |-- month: integer (nullable = true)
 |-- dayofmonth: integer (nullable = true)
 |-- carrier: string (nullable = true)
 |-- tailnum: string (nullable = true)
 |-- actualelapsedtime: integer (nullable = true)
 |-- origin: string (nullable = true)
 |-- dest: string (nullable = true)
 |-- deptime: integer (nullable = true)
 |-- arrdelayminutes: integer (nullable = true)
```

Now let's do some simple operations on this DataFrame. How many flights did each carrier fly in 2013? That's a one-liner:

```
>>> flights.groupBy("carrier").count().show()
+-------+-------+
|carrier|  count|
+-------+-------+
|     UA| 505798|
|     AA| 537891|
|     EV| 748696|
|     B6| 241777|
|     DL| 754670|
|     OO| 626359|
|     F9|  75612|
|     YV| 140922|
|     US| 412373|
|     MQ| 439865|
|     HA|  72286|
|     AS| 154743|
|     FL| 173952|
|     VX|  57133|
|     WN|1130704|
|     9E| 296701|
+-------+-------+
```

Let's now compute average delay by carrier and destination

```
>>> flights.groupBy("carrier", "dest").mean("arrdelayminutes").show()
+-------+----+--------------------+
|carrier|dest|avg(arrdelayminutes)|
+-------+----+--------------------+
|     DL| STL|   8.348766061594942|
|     DL| MSY|   8.593764258555133|
|     AS| IAH|   8.534246575342467|
|     EV| JAX|   19.50561403508772|
|     EV| LFT|  12.419161676646707|
|     EV| SYR|   18.71390798519302|
|     B6| SRQ|  12.782278481012659|
|     US| ROC|   9.662983425414364|
|     UA| JFK|  14.447821229050279|
|     VX| MCO|   4.666026871401152|
|     VX| LAS|  11.598548621190131|
|     WN| ALB|   14.44125144843569|
|     WN| BWI|  11.236715445573436|
|     OO| TUL|  12.621647058823529|
|     AS| SLC|   4.045955882352941|
|     DL| OAK|  3.1056768558951964|
|     MQ| HSV|  17.594059405940595|
|     US| ORD|  15.945858981533297|
|     WN| MAF|  11.323456790123457|
|     OO| EAU|  13.131884057971014|
+-------+----+--------------------+
only showing top 20 rows
```

Now by carrier, destination and origin!

```
>>> delays = flights.groupBy("carrier", "dest", "origin").mean("arrdelayminutes")
>>> delays.printSchema()
root
 |-- carrier: string (nullable = true)
 |-- dest: string (nullable = true)
 |-- origin: string (nullable = true)
 |-- avg(arrdelayminutes): double (nullable = true)
```

Note that we've created a new column named avg(arrdelayminutes) Let's sort this to show the worst flights in terms of average delays in 2013

```
>>> delays.sort(desc("avg(arrdelayminutes)")).show(5)
+-------+----+------+--------------------+
|carrier|dest|origin|avg(arrdelayminutes)|
+-------+----+------+--------------------+
|     EV| EYW|   MIA|               375.0|
|     EV| GPT|   MSY|               315.0|
|     UA| DEN|   MSN|               285.0|
|     EV| RIC|   PIT|               145.0|
|     EV| AEX|   LFT|               138.0|
+-------+----+------+--------------------+
only showing top 5 rows
```

The worst delays were between Key West and Miami, FL Who knew?

This step concludes this lab.