Lab 3 - SQL in Spark

Big Data

February 12, 2018

Review: Basic SQL Queries

- A basic SQL query has the form

```
SELECT      [DISTINCT] target-list
FROM        relation-list
WHERE       qualification
```

- target-list: a list of attributes of relations in relation-list
- relation-list: a list of relation names (possibly with correlation name)
- qualification: comparisons using defined operators (e.g., >, <, =), which can be combined using AND, OR, and NOT
- DISTINCT: an optional keyword indicating that answer should not contain duplicates

Log in to DUMBO

Once you have logged in, load the required modules by typing:

```
module load python/gnu/3.4.4
module load spark/2.2.0
```

Then type

```
pyspark
```

to start the pyspark shell

Spark Session

The entry point into all functionality in Spark is the `SparkSession` class. To create a basic `SparkSession`, just use `SparkSession.builder`:

```
from pyspark.sql import SparkSession

spark = SparkSession \
    .builder \
    .appName("Python Spark SQL basic example") \
    .config("spark.some.config.option", "some-value") \
    .getOrCreate()
```

Example for Today: 3 tables; sailors, boats, reserves

Table 1: boats

bid	bname	color
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

Table 2: reserves

sid	bid	day
22	101	10-OCT-18
22	102	10-OCT-18
22	103	10-AUG-18
22	104	10-JUL-18
31	102	10-NOV-18
64	101	05-SEP-18
64	102	08-SEP-18
74	103	08-SEP-18
31	103	06-NOV-18
31	104	12-NOV-18

Table 3: sailors

sid	sname	rating	age
22	dusting	7	45
29	brutus	1	33
31	lubber	8	55.5
32	andy	8	25.5
58	rusty	10	35
64	horatio	7	16
71	zorba	10	35
74	horatio	9	25.5
85	art	3	25.5
95	bob	3	63.5

Creating DataFrame from JSON file

The sailors and reserves tables are stored as json files.


```
{"sid":22, "sname":"dusting", "rating":7, "age":45.0}
{"sid":29, "sname":"brutus", "rating":1, "age":33.0}
{"sid":31, "sname":"lubber", "rating":8, "age":55.5}
{"sid":32, "sname":"andy", "rating":8, "age":25.5}
{"sid":58, "sname":"rusty", "rating":10, "age":35.0}
{"sid":64, "sname":"horatio", "rating":7, "age":16.0}
{"sid":71, "sname":"zorba", "rating":10, "age":35.0}
{"sid":74, "sname":"horatio", "rating":9, "age":25.5}
{"sid":85, "sname":"art", "rating":3, "age":25.5}
{"sid":95, "sname":"bob", "rating":3, "age":63.5}
```



```
{"sid":22, "bid":101, "date":"2018-10-10"}
{"sid":22, "bid":102, "date":"2018-10-10"}
{"sid":22, "bid":103, "date":"2018-8-10"}
{"sid":22, "bid":104, "date":"2018-7-10"}
{"sid":31, "bid":102, "date":"2018-11-10"}
{"sid":31, "bid":103, "date":"2018-11-6"}
{"sid":31, "bid":104, "date":"2018-11-12"}
{"sid":64, "bid":101, "date":"2018-9-5"}
{"sid":64, "bid":102, "date":"2018-9-8"}
{"sid":74, "bid":103, "date":"2018-9-8"}
```


Load them into DataFrame data structures with the commands

```
sailors = spark.read.json("/user/ecc290/sailors.json")
reserves = spark.read.json("/user/ecc290/reserves.json")
```

Creating DataFrame from JSON file

To print the DataFrames, type

```
sailors.show()
reserves.show()
```

DataFrame Operations

In Python it's possible to access a DataFrame's columns either by attribute (df.age) or by indexing (df['age']).

(encouraged to use the latter which is future proof and won't break with column names that are also attributes on the DataFrame class)

```
sailors.printSchema()

sailors.select("sname").show()

sailors.select(sailors['sname'], sailors['age'] + 1).show()

sailors.filter(sailors['age'] > 21).show()

sailors.groupBy("age").count().show()
```

Creating SQL temp view from DF

Register the DataFrame as a SQL temporary view

```
sailors.createOrReplaceTempView("sailors")
reserves.createOrReplaceTempView("reserves")


spark.sql("SELECT * FROM sailors").show()
```

Creating DataFrame from RDD

We will create the boats DataFrame by first creating an RDD from the text file boats.txt

```
101, Interlake, blue
102, Interlake, red
103, Clipper, green
104, Marine, red
```

```
from pyspark.sql import Row
sc = spark.sparkContext


# Load a text file and convert each line to a Row.
lines = sc.textFile("/user/ecc290/boats.txt")
parts = lines.map(lambda l: l.split(","))
boatsRDD = parts.map(lambda p: Row(bid=int(p[0]), \
    name=p[1], color=p[2]))
```

Creating DataFrame from RDD

#Infer the schema, and register the DataFrame as a table.

```
boats = spark.createDataFrame(boatsRDD)
boats.show()
```

Register the DataFrame as a SQL temporary view

```
boats.createOrReplaceTempView("boats")
spark.sql("SELECT * FROM boats").show()
```

```
# SQL can be run over DataFrames that have been registered as a table.
teenagers = spark.sql("SELECT sname FROM sailors \
       WHERE age >= 13 AND age <= 19")



boats.filter("color like '%red%'").show()
spark.sql("select * from boats \
       where color like '%red%'").show()
```

Return type

The results of SQL queries are Dataframe objects.

rdd returns the content as an :class:`pyspark.RDD` of :class:`Row`.

```
teenNames = teenagers.rdd.map(lambda p: "Name: " + p.sname).collect()
for name in teenNames:
    print(name)
```

Write DataFrame to file

Built-in sources: json, parquet, jdbc, orc, libsvm, csv, text

DataFrames loaded from any data source type can be converted into other types using this syntax

ex:

```
sailors.select("*").write.save("sailorscsv.csv", format="csv")
```

*after exiting pyspark, to move the written file from HDFS to your home directory, you type

```
    hfs -getmerge sailorscsv.csv sailorscsv.csv
```

Practice SQL Queries

- We have given you a long list of SQL queries to write using this data

- We will work through a few select queries together

Practice Writing SQL Queries

- Find the names and ages of all sailors

Practice Writing SQL Queries

- Find the names and ages of all sailors

```
spark.sql("SELECT sname, age FROM sailors").show()
```

Practice Writing SQL Queries

- Find the names and ages of all sailors

```
spark.sql("SELECT sname, age FROM sailors").show()
```

```
spark.sql("SELECT S.sname, S.age FROM sailors S").show()
```

Correlation name. Not always necessary, but good practice to use this.

Practice Writing SQL Queries

- Find all sailors with a rating above 7.

Practice Writing SQL Queries

- Find all sailors with a rating above 7.

```
spark.sql("SELECT * FROM sailors WHERE rating > 7").show()
```

Practice Writing SQL Queries

- Find the names of sailors who have reserved boat number 103

Practice Writing SQL Queries

- Find the names of sailors who have reserved boat number 103

```
spark.sql("SELECT sname FROM sailors S, reserves R \
      WHERE S.sid = R.sid AND bid = 103").show()
```

Practice Writing SQL Queries

- Find the names of sailors who have reserved boat number 103

```
spark.sql("SELECT sname FROM sailors S, reserves R \
      WHERE S.sid = R.sid AND bid = 103").show()
```

Using a nested query:

```
spark.sql("SELECT sname FROM sailors S \
      WHERE S.sid in (SELECT R.sid FROM reserves R \
            WHERE R.bid = 103)").show()
```

Practice Writing SQL Queries

- Find the sids of sailors who have reserved a red boat.

Practice Writing SQL Queries

- Find the sids of sailors who have reserved a red boat.

```
spark.sql("SELECT sid FROM reserves R, boats B \
      WHERE R.bid = B.bid AND color like '%red%'").show()
```

Practice Writing SQL Queries

- Find the sids of sailors who have reserved a red boat.

```
spark.sql("SELECT sid FROM reserves R, boats B \
      WHERE R.bid = B.bid AND color like '%red%'").show()
```

This contains duplicates. To remove duplicates, use DISTINCT keyword:

```
spark.sql("SELECT distinct(sid) FROM reserves R, boats B \
      WHERE R.bid = B.bid AND color like '%red%'").show()
```

Practice Writing SQL Queries

- Find the names of sailors who have reserved both a red and a green boat.

Practice Writing SQL Queries

- Find the names of sailors who have reserved both a red and a green boat.

Here is one incorrect query:

```
spark.sql("SELECT sname FROM sailors S, reserves R, boats B
WHERE S.sid = R.sid AND R.bid = B.bid AND (color like
'%red%' AND color like '%green%')").show()
```

What happens?

Practice Writing SQL Queries

- Find the names of sailors who have reserved both a red and a green boat.

Here is one incorrect query:

```
spark.sql("SELECT sname FROM sailors S, reserves R, boats B
WHERE S.sid = R.sid AND R.bid = B.bid AND (color like
'%red%' AND color like '%green%')").show()
```

What happens?

Practice Writing SQL Queries

- Find the names of sailors who have reserved both a red and a green boat.

Another mistake: a sailor named Horatio has reserved a red boat, and a *different* sailor named Horatio has reserved a green boat – make sure to write your query such that Horatio is not returned as a sailor that has reserved both a red and green boat!

Practice Writing SQL Queries

- Find the names of sailors who have reserved both a red and a green boat.

Another mistake: a sailor named Horatio has reserved a red boat, and a *different* sailor named Horatio has reserved a green boat – make sure to write your query such that Horatio is not returned as a sailor that has reserved both a red and green boat!

Need to use SIDs rather than name since these are the primary key in the sailors table (i.e., they are unique)

Practice Writing SQL Queries

- Find the names of sailors who have reserved both a red and a green boat.

Here is one example of a correct query:

```
spark.sql("SELECT DISTINCT(S.sname) FROM sailors S,
boats B, reserves R WHERE S.sid = R.sid AND R.bid =
B.bid AND B.color like '%red%' AND S.sid IN (SELECT
S2.sid FROM sailors S2, boats B2, reserves R2 WHERE
S2.sid=R2.sid AND R2.bid=B2.bid AND B2.color like
'%green%')").show()
```

Practice Writing SQL Queries

- Find the names of sailors who have not reserved boat number 103.

Practice Writing SQL Queries

- Find the names of sailors who have not reserved boat number 103.

```
spark.sql("SELECT sname FROM sailors S \
       WHERE S.sid NOT IN \
              (SELECT sid FROM Reserves WHERE bid = 103)").show()
```

Practice Writing SQL Queries

- Find the names of sailors whose rating is better than some sailor called Horatio.

Practice Writing SQL Queries

- Find the names of sailors whose rating is better than some sailor called Horatio.

```
spark.sql("SELECT S1.sname FROM sailors S1 \
       WHERE S1.rating > (SELECT MIN(S2.rating) FROM sailors S2 \
              WHERE S2.sname like '%horatio%')").show()
```

Practice Writing SQL Queries

- Find the average age of sailors with a rating of 10.

Practice Writing SQL Queries

- Find the average age of sailors with a rating of 10.

```
spark.sql("SELECT AVG(age) FROM Sailors WHERE rating = 10").show()
```

Practice Writing SQL Queries

- Find the name and age of the oldest sailor.

Practice Writing SQL Queries

- Find the name and age of the oldest sailor.

  What happens if we write the query:

```
spark.sql("SELECT sname, MAX(age) FROM sailors").show()
```

Practice Writing SQL Queries

- Find the name and age of the oldest sailor.

A correct query:

```
spark.sql("SELECT S.sname, S.age FROM sailors S \
       WHERE S.age = (SELECT MAX(S2.age) FROM sailors S2)").show();
```

Deliverable

(due Monday, February 26, 2018, 6pm):

Write SQL queries for the following:

1. Find the names of sailors who do not have any boat reservations
2. Find the sids of all sailors who have reserved a red boat but not a green boat.
3. Find the names of sailors whose rating is better than all sailors called Horatio.

- **You only have to submit the queries you wrote, not the output tables.**
- **Can submit via text box or upload text file.**
- **You are encouraged to work with partners or in small groups**

Resources

Documentation:

https://spark.apache.org/docs/2.2.0/api/python/pyspark.sql.html#

Programming Guide:

https://spark.apache.org/docs/2.2.0/sql-programming-guide.html