INCEPTEZ TECHNOLOGIES Spark SQL WORKOUTS

Use DataFrames to load data into Spark using CSV data

```
#import SQLContext and pyspark SQL functions

from pyspark.sql import SQLContext, Row
import pyspark.sql.functions as func
sqlContext = SQLContext(sc)


inputRDD = sc.textFile("/user/hduser/sparkdata/auctiondata.csv").map(lambda l: l.split(","))


auctions = inputRDD.map(lambda p:Row(auctionid=p[0], bid=float(p[1]), bidtime=float(p[2]),
bidder=p[3], bidrate=int(p[4]), openbid=float(p[5]), price=float(p[6]), itemtype=p[7], dtl=int(p[8])))

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

auctiondf = sqlContext.createDataFrame(auctions)
auctiondf.registerTempTable("auctions")

auctiondf.show()

auctiondf.printSchema()
```

Inspect the Data

We are going to query the DataFrame to answer the questions that will give us more insight into our data.

1. What is the total number of bids?

```
totbids = auctiondf.count()
print totbids
#10654
```

2. What is the number of distinct auctions?

```
totalauctions = auctiondf.select("auctionid").distinct().count()
print totalauctions
#627
```

3. What is the number of distinct itemtypes?

```
itemtypes = auctiondf.select("itemtype").distinct().count()
print itemtypes
#3
```

4. We would like a count of bids per auction and the item type (as shown below). How would you do this? (HINT: Use groupBy.)

```
auctiondf.groupBy("itemtype","auctionid").count().show()
```

5. For each auction item and item type, we want the max, min and average number of bids.

```
auctiondf.groupBy("itemtype","auctionid").count().agg(func.min("count"), func.max("count"),
func.avg("count")).show()
```

6. For each auction item and item type, we want the following information: (HINT: Use groupBy and agg)

```
auctiondf.groupBy("itemtype", "auctionid").agg(func.min("bid"), func.max("bid"),
func.avg("bid")).show()
```

7. What is the number of auctions with final price greater than 200?

```
auctiondf.filter(auctiondf.price>200).count()
#7685L
```

8. We want to run some basic statistics on all auctions that are of type xbox. What is one way of doing this? (HINT: We have registered the DataFrame as a table so we can use sql queries. The result will be a DataFrame and we can apply actions to it.)

```
xboxes = sqlContext.sql("SELECT auctionid, itemtype,bid,price,openbid FROM auctions WHERE itemtype
= 'xbox'").show()
```

Creating DataFrames

```
from pyspark.sql import SQLContext
sqlContext = SQLContext(sc)
df = sqlContext.read.json("file:///home/hduser/people.json")

# Displays the content of the DataFrame to stdout
df.show()

# Select only the "name" column
df.select("name").show()


# Select everybody, but increment the age by 1

df.select(df['name'], df['age'] + 1).show()

# Select people older than 21
```

```
df.filter(df['age'] > 21).show()

# Count people by age

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

Inferring the Schema Using Reflection

```
# sc is an existing SparkContext.
from pyspark.sql import SQLContext, Row
sqlContext = SQLContext(sc)

# Load a text file and convert each line to a Row.
lines = sc.textFile("file:///home/hduser/people.txt")
parts = lines.map(lambda l: l.split(","))
people = parts.map(lambda p: Row(name=p[0], age=int(p[1])))

# Infer the schema, and register the DataFrame as a table.
schemaPeople = sqlContext.createDataFrame(people)
schemaPeople.registerTempTable("people")

# SQL can be run over DataFrames that have been registered as a table.
teenagers = sqlContext.sql("SELECT name FROM people WHERE age >= 13 AND age <= 19")

# The results of SQL queries are RDDs and support all the normal RDD operations.
3

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

Data Sources

Generic Load/Save Functions

```
df = sqlContext.read.load("file:///home/hduser/users.parquet")
df.select("name", "favorite_color").write.save("file:///home/hduser/namesAndFavColors.parquet")
```

Manually Specifying Options to read json and write as parquet

```
df = sqlContext.read.load("file:///home/hduser/people.json", format="json")
df.select("name", "age").write.save("file:///home/hduser/namesAndAges.parquet", format="parquet")
```

#Run SQL on files directly

```
df = sqlContext.sql("SELECT * FROM parquet.`file:///home/hduser/users.parquet`")
```