# 732A54 - Big Data Analytics

BDA2 Spark SQL Exercises

*Ashraf Sarhan (ashsa762)*

## Question 1:

- `year, station with the max, maxValue ORDER BY maxValue DESC`
- `year, station with the min, minValue ORDER BY minValue DESC`

**Code:**

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
"""
Created on Mon Dec 12 17:06:24 2016

@author: ashraf
"""
from pyspark import SparkContext
from pyspark.sql import SQLContext, Row

iFile = 'data/temperature-readings.csv'
oFile1 = 'data/sql_max_temperature'
oFile2 = 'data/sql_max_temperature'
oFile3 = 'data/sql_min_temperature'
oFile4 = 'data/sql_min_temperature_2'

sc = SparkContext(appName = "MinMaxTempExtractorSparkSQLJob")

sqlContext = SQLContext(sc)

inFile = sc.textFile(iFile) \
            .map(lambda line: line.split(";")) \
            .map(lambda obs: \
                Row(station = obs[0], date = obs[1],  \
                    year = obs[1].split("-")[0], time = obs[2],
                    temp = float(obs[3]), quality = obs[4]))

tempSchema = sqlContext.createDataFrame(inFile)

tempSchema.registerTempTable("TempSchema")

"""
Q1.1 year, station with the max, maxValue ORDER BY maxValue DESC
"""
maxTemp = sqlContext.sql("""
        SELECT DISTINCT(table1.year) AS year,
                FIRST(table1.station) AS station,
                FIRST(temp) AS temp
        FROM TempSchema AS table1
        INNER JOIN
        (
        SELECT year, MAX(temp) AS max_temp
        FROM TempSchema
        GROUP BY year
        ) AS table2
        ON table1.year = table2.year
        WHERE table1.temp = table2.max_temp
        GROUP BY table1.year
        ORDER BY temp DESC
        """
        )

maxTemp.rdd.repartition(1) \
             .sortBy(ascending=False, keyfunc=lambda (year, temp): temp)

maxTemp.saveAsTextFile(oFile1)

"""
Another Solution
Q1.2 year, station with the max, maxValue ORDER BY maxValue DESC
"""
maxTemp2 = sqlContext.sql(
        """
        SELECT year, MAX(temp) AS temp
        FROM TempSchema
        GROUP BY year
        ORDER BY temp DESC
        """
    )

maxTemp2.rdd.repartition(1) \
             .sortBy(ascending=False, keyfunc=lambda (year, temp): temp)

maxTemp2..saveAsTextFile(oFile2)

"""
Q2.1 year, station with the min, minValue ORDER BY minValue DESC
"""
inFile = sc.textFile(iFile) \
            .map(lambda line: line.split(";")) \
            .map(lambda obs: \
                Row(station = obs[0], \
                    date = obs[1],  \
                    year = int(obs[1].split("-")[0]), \
                    time = obs[2],
                    temp = float(obs[3]), \
                    quality = obs[4]))

tempSchema = sqlContext.createDataFrame(inFile)
```

```
tempSchema.registerTempTable("TempSchema")

minTemp = tempSchema.filter(tempSchema.year <= toYear) \
                        .filter(tempSchema.year >= fromYear) \
                        .groupBy(tempSchema.year, tempSchema.station) \
                        .agg(F.min(tempSchema.temp).alias("minTemp")) \
                        .select("year", "station", "minTemp")

minTemp = minTemp.rdd.repartition(1) \
                .sortBy(ascending = False, keyfunc = lambda \
                    (year, station, minTemp): minTemp)

minTemp.saveAsTextFile(oFile3)

"""
Another Solution
Q2.2 year, station with the min, minValue ORDER BY minValue DESC
"""
minTemp2 = sqlContext.sql(
        """
        SELECT station, year, MIN(temp) AS temp
        FROM TempSchema
        GROUP BY station, year
        ORDER BY temp DESC
        """
    )

minTemp2.rdd.repartition(1) \
             .sortBy(ascending=False, keyfunc=lambda (year, station, temp): temp)

minTemp2.saveAsTextFile(oFile4)
```

**Note:**

- Using the the *FIRST* function as we need a value from the first row of a sorted group, but the needed value is not the sort key.

**Max Temp:**

```
print maxTemp.take(10)
[Row(year=u'1975', station=u'86200', temp=36.1),
 Row(year=u'1992', station=u'63600', temp=35.4),
 Row(year=u'1994', station=u'117160', temp=34.7),
 Row(year=u'2014', station=u'96560', temp=34.4),
 Row(year=u'2010', station=u'75250', temp=34.4)]
```

**Min Temp:**

```
print minTemp.take(10)
[Row(year=2010, station=u'95530', minTemp=15.2),
Row(year=1979, station=u'99090', minTemp=13.1),
Row(year=1984, station=u'53220', minTemp=12.0),
Row(year=2001, station=u'117160', minTemp=8.0),
Row(year=2010, station=u'89560', minTemp=7.9),
Row(year=1998, station=u'104390', minTemp=7.5),
Row(year=1986, station=u'84390', minTemp=5.3),
Row(year=2009, station=u'71140', minTemp=4.9),
Row(year=1970, station=u'107530', minTemp=4.1),
Row(year=1951, station=u'149160', minTemp=3.4)]
```

## Question 2:

- `year, month, value ORDER BY value DESC`
- `year, month, value ORDER BY value DESC`

**Code:**

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
"""
Created on Mon Dec 12 17:06:24 2016

@author: ashraf
"""
from pyspark import SparkContext
from pyspark.sql import SQLContext, Row
from pyspark.sql import functions as F

iFile = 'data/temperature-readings.csv'
oFile = 'data/sql_over_ten_temp_distinct_counts'


sc = SparkContext(appName = "TempCounterSparkSQLJob")

sqlContext = SQLContext(sc)

inFile = sc.textFile(iFile) \
            .map(lambda line: line.split(";")) \
            .map(lambda obs: \
                Row(station = obs[0], date = obs[1],  \
                    year = obs[1].split("-")[0], \
                    month = obs[1].split("-")[1], time = obs[2], \
                    yymm = obs[1][:7], \
                    temp = float(obs[3]), quality = obs[4]))
```

```
tempSchema = sqlContext.createDataFrame(inFile)

tempSchema.registerTempTable("TempSchema")

"""
Q1. Temperatures readings higher than 10 degrees
"""
overTenTemp = sqlContext.sql(" \
                        SELECT FIRST(year), FIRST(month), COUNT(temp) AS counts\
                        FROM TempSchema \
                        WHERE temp >= 10 AND year >= 1950 AND year <= 2014\
                        GROUP BY year, month \
                        ORDER BY counts DESC")

"""
Q2. Distinct Temperatures readings higher than 10 degrees
"""
overTenTempDistinct = tempSchema.filter(tempSchema["temp"] > 10) \
                                .groupBy("yymm") \
                                .agg(F.countDistinct("station").alias("count"))


overTenTempDistinct = overTenTempDistinct.rdd.repartition(1) \
                            .sortBy(ascending = False, keyfunc = lambda \
                                    (yymm, counts): counts)

overTenTempDistinct.saveAsTextFile(oFile)
```

**Distinct Temperatures readings counts:**

```
print overTenTempDistinct.take(10)
[Row(yymm=u'1972-10', count=378),
 Row(yymm=u'1973-05', count=377),
 Row(yymm=u'1973-06', count=377),
 Row(yymm=u'1973-09', count=376),
 Row(yymm=u'1972-08', count=376),
 Row(yymm=u'1972-05', count=375),
 Row(yymm=u'1972-06', count=375),
 Row(yymm=u'1972-09', count=375),
 Row(yymm=u'1971-08', count=375),
 Row(yymm=u'1972-07', count=374)]
```

## Question 3:

- year, month, station, avgMonthlyTemperature ORDER BY avgMonthlyTemperature DESC

**Code:**

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
"""
Created on Mon Dec 12 17:06:24 2016

@author: ashraf
"""
from pyspark import SparkContext
from pyspark.sql import SQLContext, Row

iFile = 'data/temperature-readings.csv'
oFile = 'data/sql_station_avg_mth_temp'

sc = SparkContext(appName="AvgTempSparkSQLJob")

sqlContext = SQLContext(sc)

inFile = sc.textFile(iFile) \
            .map(lambda line: line.split(";")) \
                       .filter(lambda obs:
                               (int(obs[1][:4]) >= 1960 and
                                int(obs[1][:4]) <= 2014)) \
                       .map(lambda obs: Row(station=int(obs[0]),
                                            day=obs[1],
                                            month=obs[1][:7],
                                            temp=float(obs[3])))

tempSchema = sqlContext.createDataFrame(inFile)

tempSchema.registerTempTable("TempSchema")

avgMonthTemp = sqlContext.sql(
        """
        SELECT mytbl.month, mytbl.station, AVG(mytbl.max_temp + mytbl.min_temp) / 2 AS avg_temp
        FROM
        (
        SELECT month, station, MIN(temp) AS min_temp, MAX(temp) AS max_temp
        FROM TempSchema
        GROUP BY day, month, station
        ) AS mytbl
        GROUP BY mytbl.month, mytbl.station
        ORDER BY AVG(mytbl.max_temp + mytbl.min_temp) / 2 DESC
        """
    )

avgMonthTemp.rdd.repartition(1).sortBy(ascending=False,
                                  keyfunc=lambda (month, station, temp): temp)

avgMonthTemp.saveAsTextFile(oFile)
```

**Average monthly temperatures:**

```
print avgMonthTemp.take(10)
[Row(month=u'2014-07', station=96000, avg_temp=26.3),
Row(month=u'1994-07', station=96550, avg_temp=23.07105263157895),
Row(month=u'1983-08', station=54550, avg_temp=23.0),
Row(month=u'1994-07', station=78140, avg_temp=22.970967741935485),
Row(month=u'1994-07', station=85280, avg_temp=22.872580645161293),
Row(month=u'1994-07', station=75120, avg_temp=22.858064516129033),
Row(month=u'1994-07', station=65450, avg_temp=22.856451612903232),
Row(month=u'1994-07', station=96000, avg_temp=22.808064516129033),
Row(month=u'1994-07', station=95160, avg_temp=22.764516129032256),
Row(month=u'1994-07', station=86200, avg_temp=22.711290322580645)]
```

## Question 4:

- `station, maxTemp, maxDailyPrecipitation ORDER BY station DESC`

**Code:**

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
"""
Created on Mon Dec 12 17:06:24 2016

@author: ashraf
"""
from pyspark import SparkContext
from pyspark.sql import SQLContext, Row

iFile = 'data/temperature-readings.csv'
iFile2 = 'data/precipitation-readings.csv'
oFile = 'data/sql_max_temperature_precipitation'

sc = SparkContext(appName="MaxTempPrecExtractorSparkSQLJob")

sqlContext = SQLContext(sc)

# Temperatures
inFile = sc.textFile(iFile).map(lambda line: line.split(";")) \
                                      .map(lambda obs: Row(station=int(obs[0]),
                                                           temp=float(obs[3])))

tempSchema = sqlContext.createDataFrame(inFile)
```

```
tempSchema.registerTempTable("TempSchema")

# precipitation
inFile2 = sc.textFile(iFile2).map(lambda line: line.split(";")) \
                                          .map(lambda obs: Row(station=int(obs[0]),
                                                                day=obs[1],
                                                                precip=float(obs[3])))

precSchema = sqlContext.createDataFrame(inFile2)
precSchema.registerTempTable("PrecSchema")

combinedTempPrec = sqlContext.sql(
        """
        SELECT tr.station, MAX(temp) AS max_temp, MAX(precip) AS max_precip
        FROM
        TempSchema AS tr
        INNER JOIN
        (
        SELECT station, SUM(precip) AS precip
        FROM PrecSchema
        GROUP BY day, station
        ) AS pr
        ON tr.station = pr.station
        GROUP BY tr.station
        HAVING max_temp >= 25 AND
               max_temp <= 30 AND
               max_precip >= 100 AND
               max_precip <= 200
        ORDER BY tr.station DESC
        """
    )

tempPrec.saveAsTextFile(oFile)
```

**Note:**

- As we have a condition on the aggregation function *Max(temp)* and *Max(precip)*, it should be under *HAVING* clause and not *WHERE* clause.

**Max daily temperatures/precipitation:**

```
tempPrec.take(5)
[]
```

## Question 5:

- station, maxTemp, maxDailyPrecipitation ORDER BY station DESC

**Code:**

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
"""
Created on Mon Dec 12 17:06:24 2016

@author: ashraf
"""
from pyspark import SparkContext
from pyspark.sql import SQLContext, Row

iFile = 'data/stations-Ostergotland.csv'
iFile2 = 'data/precipitation-readings.csv'
oFile = 'data/OstergotlandAveMonthlyPrec'

sc = SparkContext(appName="OstergotlandAvgMonthlyPrecSparkSQLJob")

sqlContext = SQLContext(sc)

# Ostergotland Stations
ostergotlandStations = sc.textFile(iFile).map(lambda line: line.split(";")) \
                           .map(lambda obs: int(obs[0])) \
                           .distinct().collect()

ostergotlandStations = sc.broadcast(ostergotlandStations)

ostergotlandStations = {station: True for station in ostergotlandStations.value}

precipitations = sc.textFile(iFile2).map(lambda line: line.split(";")) \
                                          .filter(lambda obs:
                                                    ostergotlandStations.get(int(obs[0]), False)) \
                                          .map(lambda obs: Row(day=obs[1],
                                                              month=obs[1][:7],
                                                              station=int(obs[0]),
                                                              precip=float(obs[3])))

precSchema = sqlContext.createDataFrame(precipitations)
precSchema.registerTempTable("PrecSchema")

avgMthPrec = sqlContext.sql(
        """
        SELECT mytbl2.month, AVG(mytbl2.precip) AS avg_precip
        FROM
        (
        SELECT mytbl1.month, mytbl1.station, SUM(mytbl1.precip) AS precip
        FROM
        (
        SELECT month, station, SUM(precip) AS precip
        FROM PrecSchema
        GROUP BY day, month, station
        ) AS mytbl1
        GROUP BY mytbl1.month, mytbl1.station
        ) AS mytbl2
        GROUP BY mytbl2.month
        ORDER BY mytbl2.month DESC
        """
    )

avgMthPrec.rdd.repartition(1).sortBy(ascending=False, keyfunc=lambda (month, precip): month)

avgMthPrec.saveAsTextFile(oFile)
```

**Östergotland average monthly precipitation:**

```
print avgMthPrec.take(10)
[Row(month=u'2016-07', avg_precip=0.0),
 Row(month=u'2016-06', avg_precip=47.6625),
 Row(month=u'2016-05', avg_precip=29.250000000000004),
 Row(month=u'2016-04', avg_precip=26.9),
 Row(month=u'2016-03', avg_precip=19.9625),
 Row(month=u'2016-02', avg_precip=21.5625),
 Row(month=u'2016-01', avg_precip=22.325),
 Row(month=u'2015-12', avg_precip=28.925),
 Row(month=u'2015-11', avg_precip=63.887499999999996),
 Row(month=u'2015-10', avg_precip=2.2625)]
```

## Question 6:

- `Year, month, difference ORDER BY year DESC, month DESC`

**Code:**

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*-
"""
Created on Mon Dec 12 17:06:24 2016

@author: ashraf
"""
from pyspark import SparkContext
from pyspark.sql import SQLContext, Row

iFile = 'data/stations-Ostergotland.csv'
iFile2 = 'data/temperature-readings.csv'
oFile = 'data/OstergotlandAveMonthlyDiffTemp'
```

```
sc = SparkContext(appName="OstergotlandAvgMonthlyTempDiffSparkSQLJob")

sqlContext = SQLContext(sc)

# Ostergotland Stations
ostergotlandStations = sc.textFile(iFile).map(lambda line: line.split(";")) \
                           .map(lambda obs: int(obs[0])) \
                           .distinct().collect()

ostergotlandStations = sc.broadcast(ostergotlandStations)

ostergotlandStations = {station: True for station in ostergotlandStations.value}

temperatures = sc.textFile(iFile2) \
            .map(lambda line: line.split(";")) \
            .filter(lambda obs: ostergotlandStations.get(int(obs[0]), False)) \
            .map(lambda obs: \
                Row(station = obs[0], \
                    date = obs[1],  \
                    year = obs[1].split("-")[0], \
                    month = obs[1].split("-")[1], \
                    day = obs[1].split("-")[2], \
                    yymm = obs[1][:7], \
                    yymmdd = obs[1], \
                    time = obs[2], \
                    temp = float(obs[3]), \
                    quality = obs[4]))

tempSchema = sqlContext.createDataFrame(temperatures)
tempSchema.registerTempTable("TempSchema")

avgMthTemp = sqlContext.sql("""
        SELECT one.yymm,
            AVG(one.minTemp + one.maxTemp) / 2 AS avgTemp
        FROM
        (
        SELECT yymm,
                year,
                yymmdd,
                MIN(temp) AS minTemp,
                MAX(temp) AS maxTemp
        FROM TempSchema
        GROUP BY yymmdd,
                    yymm,
                    year,
                    station
        ) AS one
        WHERE one.year >= 1950 AND one.year <= 2014
        GROUP BY one.yymm
        """)

longTermAvgTemp = avgMthTemp \
                    .filter(F.substring(avgMthTemp["yymm"], 1, 4) <= 1980) \
                    .groupBy(F.substring(avgMthTemp["yymm"], 6, 7).alias("month")) \
                    .agg(F.avg(avgMthTemp["avgTemp"]).alias("longTermAvgTemp"))

diffTemp = avgMthTemp.join(longTermAvgTemp,
                                (F.substring(avgMthTemp["yymm"], 6, 7) ==
                                  longTermAvgTemp["month"]), "inner")

diffTemp = diffTemp.select(diffTemp["yymm"],
                          (F.abs(diffTemp["avgTemp"]) -
                             F.abs(diffTemp["longTermAvgTemp"])).alias("diffTemp"))

diffTemp = diffTemp.rdd.repartition(1).sortBy(ascending = False,
                             keyfunc = lambda (yymm, diff): yymm)

diffTemp.saveAsTextFile(oFile)
```

**Östergotland average monthly precipitation temperature difference:**

```
print diffTemp.take(10)
[Row(yymm=u'2014-12', diffTemp=-0.7938517834097853),
 Row(yymm=u'2014-11', diffTemp=2.0635396726928987),
 Row(yymm=u'2014-10', diffTemp=1.521957490617976),
 Row(yymm=u'2014-09', diffTemp=0.06105818643722749),
 Row(yymm=u'2014-08', diffTemp=-0.6426470719706963),
 Row(yymm=u'2014-07', diffTemp=2.1059218387139893),
 Row(yymm=u'2014-06', diffTemp=-1.8073686197315233),
 Row(yymm=u'2014-05', diffTemp=0.26719065014070154),
 Row(yymm=u'2014-04', diffTemp=2.0661931589915437),
 Row(yymm=u'2014-03', diffTemp=3.176498950234642)]
```