SPARK

**SolutionExamexample1**

**Exercise2:**

inputPathPOIs = "exam\_ex2\_data/POIs.txt"

outputPathPartA = "out\_PartA"

outputPathPartB = "out\_PartB"

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**# Exercise 2 - Part A**

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Read the content of POIs.txt

# POI\_ID,latitude,longitude,city,country,category,subcategory

poisRDD = sc.textFile(inputPathPOIs)

# Select only the Italian cities

poisItalyRDD = poisRDD.filter(lambda line: line.split(",")[4]=="Italy").cache()

# Select only the POIs with subcategory="Taxi" or subcategory="Busstop"

def filterSubcategories(line):

    fields = line.split(",")

    subcategory = fields[6]

    if (subcategory=="taxi" or subcategory=="busstop"):

        return True

    else:

        return False

poisItalySelectedSubcategoriesRDD = poisItalyRDD.filter(filterSubcategories)

# Generate for each of the selected input lines a pair with

# key = city

# value = (taxi=0/1,busstop=0/1)

def extractPairsCitySubcategoriesCounters(line):

    fields = line.split(",")

    city = fields[3]

    subcategory = fields[6]

    if (subcategory=="taxi"):

        return (city, (1, 0))

    else:

        return (city, (0, 1))

cityTaxiBusstopRDD = poisItalySelectedSubcategoriesRDD.map(extractPairsCitySubcategoriesCounters)

# For each Italian city, compute

# - number of taxi POIs

# - number of Busstop POIs

cityNumTaxiNumBusstopRDD = cityTaxiBusstopRDD.reduceByKey(lambda v1,v2: (v1[0]+v2[0], v1[1]+v2[1]) )

# Select only the element with

# - number of taxi POIs >=1

# - number of Busstop =0

selectedCityNumTaxiNumBusstopRDD = cityNumTaxiNumBusstopRDD\

.filter(lambda pair: pair[1][0]>=1 and pair[1][1]==0)

# Save the selected cities = the keys of selectedCityNumTaxiNumBusstop

selectedCityNumTaxiNumBusstopRDD.keys()\

.saveAsTextFile(outputPathPartA)

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**# Exercise 2 - Part B**

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Count the number of museum POIs for each Italian city

# Generate for each input line a pair with

# key = city

# value = 1 if subcategory="museum"

# value = 0 if subcategory!="museum"

def extractCityMuseum(line):

    fields = line.split(",")

    city = fields[3]

    subcategory = fields[6]

    if (subcategory=="museum"):

        return (city, 1)

    else:

        return (city, 0)

cityMuseumPOIsRDD = poisItalyRDD.map(extractCityMuseum)

# Count the number of museum POIs for each Italian city

cityNumMuseumPOIsRDD = cityMuseumPOIsRDD.reduceByKey(lambda v1,v2: v1+v2).cache()

# Each input element represents

# - an italian city

# - the number of museum POIs for that city

# Emit one tuple for each input pair. Each tuple represents

# - numOfCities (set to 1 for each input pair)

# - numMuseumPOIs (for that city) = value of the input pair

numCitiesNumMuseumPOIsRDD = cityNumMuseumPOIsRDD.map(lambda pair: (1, pair[1]))

# Compute total number of Italian cities and total number of "museum" POIs the

# Italy (i.e., in the Italian cities)

# Store it in a local variable of the driver

totalNumCitiesNumMuseumPOIs = numCitiesNumMuseumPOIsRDD\

.reduce(lambda tuple1,tuple2: (tuple1[0]+tuple2[0], tuple1[1]+tuple2[1]) )

# Compute average number of "museum" POIs per city in Italy

# Store it in a local variable of the driver

average = totalNumCitiesNumMuseumPOIs[1]/totalNumCitiesNumMuseumPOIs[0]

# Select only the Italian cities with a number of "museum" POIs

# (subcategory="museum") greater than the average number of "museum" POIs per

# city in Italy

selectedCityNumMuseumPOIsRDD = cityNumMuseumPOIsRDD.\

filter(lambda inputPair: inputPair[1]> average)

# Store the selected cities

selectedCityNumMuseumPOIsRDD.keys()\

.saveAsTextFile(outputPathPartB)

**ExamExample1Exercise2SQL**

inputPathPOIs = "exam\_ex2\_data/POIs.txt"

outputPathPartA = "out\_PartASQL"

outputPathPartB = "out\_PartBSQL"

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Exercise 2 - Part A

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Read the content of POIs.txt

# POI\_ID,latitude,longitude,city,country,category,subcategory

poisDF = spark.read.load(inputPathPOIs,\

                            format="csv",\

                            header=False,\

                            inferSchema=True)\

.withColumnRenamed("\_c0", "poiId")\

.withColumnRenamed("\_c3", "city")\

.withColumnRenamed("\_c4", "state")\

.withColumnRenamed("\_c5", "category")\

.withColumnRenamed("\_c6", "subcategory")

#poisDF.printSchema()

#poisDF.show()

# "register" poisDF

poisDF.createOrReplaceTempView("poisTable")

# Define some UDFs

def taxi(subcategory):

    if subcategory=='taxi':

        return 1

    else:

        return 0

def busstop(subcategory):

    if subcategory=='busstop':

        return 1

    else:

        return 0

spark\

.udf\

.register("taxi", lambda subcategory: taxi(subcategory))

spark\

.udf\

.register("busstop", lambda subcategory: busstop(subcategory))

resADF = spark.sql("""SELECT city

FROM poisTable

WHERE state='Italy' AND

(subcategory='taxi' OR subcategory='busstop')

GROUP BY city

HAVING SUM(taxi(subcategory))>=1 AND

SUM(busstop(subcategory))=0

""")

#resADF.show()

resADF.write.csv(outputPathPartA, header=False)

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**# Exercise 2 - Part B**

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Define a UDF

def museum(subcategory):

    if subcategory=='museum':

        return 1

    else:

        return 0

spark\

.udf\

.register("museum", lambda subcategory: museum(subcategory))

# Num. Museums for each Italian city

numMuseumsCityDF = \

spark.\

sql("""SELECT city,

SUM(museum(subcategory)) AS numMuseums

FROM poisTable

WHERE state='Italy'

GROUP BY city

""")

numMuseumsCityDF\

.createOrReplaceTempView("numMuseumsCityTable")

# Compute the average number of museum per Italian city

avgMuseumsDF = \

spark\

.sql("""SELECT avg(numMuseums) AS Threshold

FROM numMuseumsCityTable""")

avgMuseumsDF\

.createOrReplaceTempView("avgMuseumsTable")

# Compute the average number of museum per Italian city

resBDF = spark.sql("""SELECT city

FROM numMuseumsCityTable, avgMuseumsTable

WHERE numMuseumsCityTable.numMuseums>avgMuseumsTable.Threshold""")

# Solution with table function

resBTDF = spark.sql("""SELECT city

FROM numMuseumsCityTable,

(SELECT avg(numMuseums) AS

Threshold FROM numMuseumsCityTable) avgMuseumsTable

WHERE numMuseumsCityTable.numMuseums>avgMuseumsTable.Threshold""")

#resBTDF.show()

resBDF.write.csv(outputPathPartB, header=False)