SOLUTION WITH RDDs

EXERCISE 1

Some cells may look like they haven't been run, in those cases their output is reported (copied and pasted) in a markdown cell because of very long warnings that would make the report longer than 10 pages.

In [ ]:
```
from datetime import datetime

RDD_reg = sc.textFile('/data/students/bigdata_internet/lab3/register.csv')
RDD_st = sc.textFile('/data/students/bigdata_internet/lab3/stations.csv')
header = RDD_reg.first()
new_rdd_r = RDD_reg.filter(lambda x: x!= header)
header2 = RDD_st.first()
new_rdd_s = RDD_st.filter(lambda x: x!= header2)
new_rdd_r2 = new_rdd_r.map(lambda l : l.split('\t'))
new_rdd_r2.take(2)
```

[['1', '2008-05-15 12:01:00', '0', '18'], ['1', '2008-05-15 12:02:00', '0', '18']]

In [3]:
```
filtered_RDD = new_rdd_r2.filter(lambda x : int(x[2])>0 or int(x[3])>0)
before = new_rdd_r2.count()
after = filtered_RDD.count()
```

1.1.1 How many rows of data we obtain before and after the data cleaning above?

In [4]:
```
print("Number of rows before cleaning:",before)
print("Number of rows after cleaning:",after)
```

```
Number of rows before cleaning: 25319028
Number of rows after cleaning: 25104121
```

EXERCISE 2

In [ ]:
```
RDD=filtered_RDD.map(lambda line:(datetime.strptime(line[1],\
                                                    "%Y-%m-%d %H:%M:%S").\
                                  strftime("%A %-H"),line[0],line[2],line[3]))
pair_rdd=RDD.map(lambda line: \
                 ((line[1]+' '+str(line[0])),int(line[2]),int(line[3])))
pair_rdd.take(2)
```

[('1 Thursday 12', 0, 18), ('1 Thursday 12', 0, 18)]

In [6]:
```
def zero_or_one(line):
 critical_v=0
 readings_tot=0
 free_slots=int(line[2])
 key=line[0]
 if free_slots==0:
     critical_v=1
 else:
     critical_v=0
```

```
 return key,(critical_v,1)
```

The zero_or_one function is used to signal a critical case (0 free slots) with a 1, another 1 is used to signal the presence of a case (critcal or not) so that, in the following cell, we can obtain the number of total cases (sum of all the ones in line[1][1]) and the number of critical cases (sum of the values in line[1][0]) by reducung the RDD with respect to the key.

In [7]:
```
pair_rdd2=pair_rdd.map(lambda line: zero_or_one(line))
reduced_pair=pair_rdd2.reduceByKey(lambda v1,v2: (v1[0]+v2[0],v1[1]+v2[1]))
```

In [ ]:
```
pair_rdd2.take(1)
```

[('1 Thursday 12', (0, 1))]

In [9]:
```
reduced_pair.take(1)
```

Out[9]:
```
[('1 Thursday 17', (8, 577))]
```

2.1) Compute criticality values C(Si,Tj) for each pair (Si,Tj)

In [30]:
```
#the criticality for a certain key(station day hour) is equal to the ratio
#between readings with critical state and total readings
critical_pairs=reduced_pair.map(lambda line:(line[0],line[1][0]/line[1][1]))
critical_pairs.take(5)
```

Out[30]:
```
[('1 Thursday 17', 0.01386481802426343),
 ('1 Thursday 21', 0.1258741258741259),
 ('1 Friday 16', 0.036020583190394515),
 ('1 Friday 18', 0.008620689655172414),
 ('1 Friday 22', 0.09137931034482759)]
```

2.2. Selects only the critical pairs having a criticality value greater than a minimum threshold. The minimum criticality threshold is a float between 0 and 1 passed as an argument of the application.

In [11]:
```
threshold=0.6
selected=critical_pairs.filter(lambda line: float(line[1]>threshold))
```

Out[11]:
```
[('9 Friday 22', 0.6258389261744967),
 ('58 Monday 1', 0.6239554317548747),
 ('9 Friday 10', 0.6129032258064516),
 ('10 Saturday 0', 0.622107969151671),
 ('58 Monday 0', 0.6323119777158774)]
```

2.3. Order the results by increasing criticality. If there are two or more records characterized by the same criticality value, consider the station id value (in ascending order). If also the station is the same, consider the day of the week (ascending from Monday to Sunday) and finally the hour (ascending from 0 to 23).

In [31]:
```
divided_pairs=selected.map(lambda line:(int(line[0].split(' ')[0]),\
                                        line[0].split(' ')[1],\
                                        int(line[0].split(' ')[2]),line[1]))
ordered_pairs=divided_pairs.sortBy(lambda line:(line[3],line[0],\
                                                (datetime.strptime(line[1],'%A')\
```

```
                                                .strftime('%w')),line[2]))
#in ordered_pairs when I order with respect to week days,
#week days are associated to a number (sunday = 0, monday=1....)
```

Out[31]:
```
[(9, 'Friday', 10, 0.6129032258064516),
 (10, 'Saturday', 0, 0.622107969151671),
 (58, 'Monday', 1, 0.6239554317548747),
 (9, 'Friday', 22, 0.6258389261744967),
 (58, 'Monday', 0, 0.6323119777158774)]
```

2.4. Store the sorted critical pairs in the output folder (also an argument of the application), by using a csv files (with header), where columns are separated by "tab". Store exactly the following attributes separated by a "tab":

- station #l[0]
- station longitude #l[1][1][0]
- station latitude #l[1][1][1]
- day of week #l[1][0][0]
- hour #l[1][0][1]
- criticality value #l[1][0][2]

In [32]:
```
new_rdd_s2 = new_rdd_s.map(lambda l: (int(l.split('\t')[0]),\
                                      [float(l.split('\t')[1]),\
                                    float(l.split('\t')[2]),\
                                    l.split('\t')[3]]))
ordered_s=ordered_pairs.map(lambda l: (l[0],[l[1],l[2],l[3]]))
joinRDD=ordered_s.join(new_rdd_s2)
joinRDD.take(1)
```

Out[32]:
```
[(9,
  (['Friday', 10, 0.6129032258064516],
   [2.185294, 41.385006, "MarquÃ¨s de l\\'Argentera"]))]
```

In [14]:
```
joinRDD2=joinRDD.map(lambda l:[str(l[0])+'\t'+str(l[1][1][0])\
                               +'\t'+str(l[1][1][1])+'\t'+\
                               str(l[1][0][0])+'\t'+str(l[1][0][1])+\
                               '\t'+str(l[1][0][2])])
#Create header
labels=['station\tstation longitude\tstation latitude\tday \
of week\thour\tcriticality value']
new_header=sc.parallelize([labels])
finalRDD=new_header.union(joinRDD2)
finalRDD.take(5)
```

Out[14]:
```
[['station\tstation longitude\tstation latitude\tday of week\thour\tcriticality va
lue'],
 ['9\t2.185294\t41.385006\tFriday\t10\t0.6129032258064516'],
 ['9\t2.185294\t41.385006\tFriday\t22\t0.6258389261744967'],
 ['10\t2.185206\t41.384875\tSaturday\t0\t0.622107969151671'],
 ['58\t2.170736\t41.377536\tMonday\t1\t0.6239554317548747']]
```

2.5. How many critical pairs do you obtain? Report also the complete output result of the applications.

In [15]:
```
joinRDD2.count()
```

Out[15]:
```
5
```

SOLUTION WITH SQL

EXERCISE 1

In [16]:
```
spark = SparkSession.builder.getOrCreate()
DF =spark.read.load('/data/students/bigdata_internet/lab3/register.csv',\
                    sep ='\t',
                   format = 'csv',
                   header=True,
                   inferSchema=True)
```

In [17]:
```
rows=DF.count()
DF_filtered=DF.filter("used_slots>0 or free_slots>0")
new_rows=DF_filtered.count()
```

1.1.1 How many rows of data we obtain before and after the data cleaning above?

In [18]:
```
print("Number of rows before cleaning:",rows)
print("Number of rows after cleaning:",new_rows)
```

```
Number of rows before cleaning: 25319028
Number of rows after cleaning: 25104121
```

EXERCISE 2

In [19]:
```
DF_filtered.createOrReplaceTempView("Register")
new_DF=spark.sql("SELECT station,date_format(timestamp,'EEEE') as day,\
 hour(timestamp) as hour,used_slots,free_slots\
 FROM Register\
 ")
new_DF.createOrReplaceTempView("Table")
```

In [20]:
```
DF_filtered.show(4)
```

```
+-------+-------------------+----------+----------+
|station|          timestamp|used_slots|free_slots|
+-------+-------------------+----------+----------+
|      1|2008-05-15 12:01:00|         0|        18|
|      1|2008-05-15 12:02:00|         0|        18|
|      1|2008-05-15 12:04:00|         0|        18|
|      1|2008-05-15 12:06:00|         0|        18|
+-------+-------------------+----------+----------+
only showing top 4 rows
```

In [21]:
```
new_DF.show(4)
```

```
+-------+--------+----+----------+----------+
|station|     day|hour|used_slots|free_slots|
+-------+--------+----+----------+----------+
|      1|Thursday|  12|         0|        18|
|      1|Thursday|  12|         0|        18|
|      1|Thursday|  12|         0|        18|
|      1|Thursday|  12|         0|        18|
+-------+--------+----+----------+----------+
only showing top 4 rows
```

In [22]:
```
grouped_DF=spark.sql("SELECT station,day,hour,count(free_slots) as free_slots,\
 sum(case when free_slots==0 then 1 else 0 end) as empty_slots \
 FROM Table\
 GROUP BY station,day,hour")
grouped_DF.createOrReplaceTempView("Table2")
grouped_DF.show(4)
```

```
+-------+--------+----+----------+-----------+
|station|     day|hour|free_slots|empty_slots|
+-------+--------+----+----------+-----------+
|      1|  Sunday|  11|       548|          0|
|      4|Thursday|  11|       563|          0|
|      5|  Sunday|  12|       533|         17|
|      7|Thursday|   2|       386|         30|
+-------+--------+----+----------+-----------+
only showing top 4 rows
```

2.1) Compute criticality values C(Si,Tj) for each pair (Si,Tj)

In [23]:
```
criticality_DF=spark.sql("SELECT station,day,hour,\
empty_slots/free_slots as criticality \
 FROM Table2")
criticality_DF.show(4)
```

```
[Stage 61:=============================================================>(71 + 1) / 72]
+-------+--------+----+-------------------+
|station|     day|hour|        criticality|
+-------+--------+----+-------------------+
|      1|  Sunday|  11|                0.0|
|      4|Thursday|  11|                0.0|
|      5|  Sunday|  12|0.03189493433395872|
|      7|Thursday|   2|0.07772020725388601|
+-------+--------+----+-------------------+
only showing top 4 rows
```

2.2) Selects only the critical pairs having a criticality value greater than a minimum threshold. The minimum criticality threshold is a float between 0 and 1 passed as an argument of the application.

In [24]:
```
criticality_DF.createOrReplaceTempView("Table3")
#threshold=0.6
sel_criticalityDF=spark.sql("SELECT *\
 FROM Table3\
```

```
 WHERE criticality>0.6")
criticality_DF.show(4)
```

```
[Stage 63:================>                                  (21 + 51) / 72]
+-------+--------+----+-------------------+
|station|     day|hour|        criticality|
+-------+--------+----+-------------------+
|      1|  Sunday|  11|                0.0|
|      4|Thursday|  11|                0.0|
|      5|  Sunday|  12|0.03189493433395872|
|      7|Thursday|   2|0.07772020725388601|
+-------+--------+----+-------------------+
only showing top 4 rows
```

2.3) Order the results by increasing criticality. If there are two or more records characterized by the same criticality value,consider the station id value(in ascending order). If also the station is the same, consider the day of the week (ascending from Monday to Sunday) and finally the hour (ascending from 0 to23).

In [25]:
```
sorted_DF=spark.sql("SELECT *\
 FROM Table3\
 WHERE criticality>0.6\
 ORDER BY criticality,station,day,hour")
sorted_DF.show()
sorted_DF.createOrReplaceTempView("Table4")
```

```
[Stage 65:============================================================>(71 + 1) / 72]
+-------+--------+----+------------------+
|station|     day|hour|       criticality|
+-------+--------+----+------------------+
|      9|  Friday|  10|0.6129032258064516|
|     10|Saturday|   0| 0.622107969151671|
|     58|  Monday|   1|0.6239554317548747|
|      9|  Friday|  22|0.6258389261744967|
|     58|  Monday|   0|0.6323119777158774|
+-------+--------+----+------------------+
```

2.4. Store the sorted critical pairs in the output folder (also an argument of the application), by using a csv files (with header), where columns are separated by "tab". Store exactly the following attributes separated by a "tab":

station station longitude station latitude day of week hour criticality value

In [26]:
```
station_DF=spark.read.load("/data/students/bigdata_internet/lab3/stations.csv", \
                           sep='\t',format='csv',header=True,inferSchema=True)
station_DF.createOrReplaceTempView("Table5")
```

In [27]:
```
join_DF=spark.sql("SELECT station,longitude,latitude,day,hour,criticality\
 FROM Table4\
 INNER JOIN Table5 ON Table4.station=Table5.id")
join_DF.show()
```

```
+-------+---------+---------+--------+----+------------------+
|station|longitude| latitude|     day|hour|       criticality|
+-------+---------+---------+--------+----+------------------+
|      9| 2.185294|41.385006|  Friday|  10|0.6129032258064516|
|     10| 2.185206|41.384875|Saturday|   0| 0.622107969151671|
|     58| 2.170736|41.377536|  Monday|   1|0.6239554317548747|
|      9| 2.185294|41.385006|  Friday|  22|0.6258389261744967|
|     58| 2.170736|41.377536|  Monday|   0|0.6323119777158774|
+-------+---------+---------+--------+----+------------------+
```

2.5) How many critical pairs do you obtain? Report also the complete output result of the applications.

In [28]:
```
print("Number of critical pairs:",join_DF.count())
```

```
Number of critical pairs: 5
```

In [ ]:
```
outputPath="/user/s307735/lab3SQL"
joinDF.write.csv(outputPath,sep ='\t',header=True)
```