TPCH-Example-Solution-Notebook-RDD

October 5, 2020

[2]:

```
import sys
from operator import add
from itertools import combinations

import requests

from pyspark.sql.types import *

from pyspark.sql import functions as f
from pyspark.sql.functions import udf

from pyspark.sql.functions import lit
from pyspark import sql

sqlContext = sql.SparkSession.builder \
        .master("local") \
        .appName("Word Count") \
        .getOrCreate()
```

[3]:

```
# lineitems = sqlContext.read.format('csv').options(header='true',␣
 ↪inferSchema='true',  sep ="|").load(sys.arg[1])
path="./tpch_tables_scale_0.1/"
# path is where you have the folder. It can be a distributed path like S3, gc␣
 ↪or hdfs

customer = sqlContext.read.format('csv').options(header='true',␣
 ↪inferSchema='true',  sep ="|").load(path+"customer.tbl")
order = sqlContext.read.format('csv').options(header='true',␣
 ↪inferSchema='true',  sep ="|").load(path+"orders.tbl")
lineitems = sqlContext.read.format('csv').options(header='true',␣
 ↪inferSchema='true',  sep ="|").load(path+"lineitem.tbl")
part = sqlContext.read.format('csv').options(header='true', inferSchema='true',␣
 ↪ sep ="|").load(path+"part.tbl")
supplier = sqlContext.read.format('csv').options(header='true',␣
 ↪inferSchema='true',  sep ="|").load(path+"supplier.tbl")
partsupp = sqlContext.read.format('csv').options(header='true',␣
 ↪inferSchema='true',  sep ="|").load(path+"partsupp.tbl")
```

```
region = sqlContext.read.format('csv').options(header='true',␣
 ↪inferSchema='true',  sep ="|").load(path+"region.tbl")
nation = sqlContext.read.format('csv').options(header='true',␣
 ↪inferSchema='true',  sep ="|").load(path+"nation.tbl")
```

[4]:

```
customerRDD=customer.rdd
orderRDD=order.rdd
lineitemsRDD=lineitems.rdd
partRDD=part.rdd
supplierRDD=supplier.rdd
partsuppRDD=partsupp.rdd
regionRDD=region.rdd
nationRDD=nation.rdd
```

[11]:

```
# Question 1
# Implement a pyspark code that can find out the top-10 sold products.

l = lineitemsRDD\
    .map(lambda x: (x[1], 1))

result = l\
    .reduceByKey(add)\
    .top(10, lambda x: x[1])

for idx, i in enumerate(result, start=1):
    print("{}: {}".format(idx,i))
```

```
1: (10620, 56)
2: (6140, 54)
3: (15584, 52)
4: (8051, 52)
5: (2292, 51)
6: (10597, 51)
7: (10715, 51)
8: (19444, 50)
9: (3225, 50)
10: (14422, 50)
```

[12]:

```
# ---------------------------------------------------------------------------
# Question 2

# Find the top-10 customers based on the number of products ordered.

l = lineitemsRDD.map(lambda x: (x[0], x[4]))
o = orderRDD.map(lambda x: (x[0], x[1]))

result = o.join(l)\
    .map(lambda x: (x[1][0], x[1][1]) )\
    .reduceByKey(add)\
    .top(10, lambda x: x[1])

for idx, i in enumerate(result, start=1):
    print("{}: {}".format(idx,i))
```

```
1: (8362, 4082)
2: (9454, 3870)
3: (346, 3817)
4: (6958, 3760)
5: (1105, 3737)
6: (14707, 3710)
7: (11998, 3709)
8: (14398, 3670)
9: (8542, 3660)
10: (8761, 3658)
```

[13]:

```
# ---------------------------------------------------------------------------
# Question 3
# Find the top-10 customers that have ordered products from the same supplier.

l = lineitemsRDD.map(lambda x: (x[0], x[1]))
o = orderRDD.map(lambda x: (x[0], x[1]))
p = partsuppRDD.map(lambda x: (x[0], x[1]))

ol = o.fullOuterJoin(l)\
    .map(lambda x: (x[1][1], x[1][0]) )

result = ol.fullOuterJoin(p)\
    .map(lambda x: ((x[1][0],x[1][1]), 1) )\
    .reduceByKey(add)\
    .top(10, lambda x: x[1])

for idx, i in enumerate(result, start=1):
    print("{}: {}".format(idx, i))
```

```
1: ((4567, 844), 7)
2: ((11809, 17), 6)
3: ((4792, 592), 6)
4: ((874, 430), 6)
5: ((14767, 8), 6)
6: ((2173, 572), 6)
7: ((6139, 233), 6)
8: ((2603, 288), 5)
9: ((5110, 9), 5)
10: ((14551, 942), 5)
```

[19]:

```
# ---------------------------------------------------------------------------
# Question 4 and 5
# Find the customers who have not ordered products from their own country and␣
 ↪have ordered only foreign products.

o = orderRDD.map(lambda x: (x[0], x[1]))
l = lineitemsRDD.map(lambda x: (x[0], x[2]))
c = customerRDD.map(lambda x: (x[0], x[3]))
s = supplierRDD.map(lambda x: (x[0], x[3]))

ol = o.fullOuterJoin(l)\
    .map(lambda x: (x[1][0], x[1][1]))
olc = ol.fullOuterJoin(c)\
    .map(lambda x: (x[1][0], (x[0], x[1][1])))
olcs = olc.fullOuterJoin(s)\
    .map(lambda x: ((x[1][0][0],x[1][0][1]), x[1][1]))\
    .filter(lambda x: x[1] is not None)

def rdd_set(x,y):
    x = [x,] if isinstance(x,int) else x
    y = [y,] if isinstance(y,int) else y
    result = set(x)
    result.update(y)
    return list(result)

result = olcs.reduceByKey(lambda x,y: rdd_set(x,y))


# Q4 Answer
result1 = result\
    .map(lambda x: (x[0][0], all(i is x[0][1] for i in x[1])))\
    .filter(lambda x: x[1] is True)

print(result1.collect())
```

```
[]
```

[20]:

```
# Q5 Answer
result2 = result\
    .map(lambda x: (x[0][0], x[0][1] not in x[1]))\
    .filter(lambda x: x[1] is True).collect()

for idx, i in enumerate(result2[:20], start=1):
    print("{}: {}".format(idx,i[0]))
```

```
1: 5701
2: 11338
```

```
3: 9359
4: 4781
5: 6905
6: 3953
7: 14888
8: 13117
9: 11474
10: 1586
11: 2957
12: 12797
13: 6652
14: 12070
15: 85
16: 14423
17: 3137
18: 12128
19: 2411
20: 13910
```

[21]:

```
# Q6 Answer

def jaccard_similarity(list1, list2):
    s1 = set(list1)
    s2 = set(list2)
    return len(s1.intersection(s2)) / len(s1.union(s2))

# order: (order, cust)
# line: (order, part)
o = orderRDD.map(lambda x: (x[0],x[1]))
l = lineitemsRDD.map(lambda x: (x[0], x[1]))

# ol: ("cust", "part")
ol = o.fullOuterJoin(l).map(lambda x: (x[1][0],x[1][1]))\
    .reduceByKey(lambda x,y: rdd_set(x,y))

result = ol.cartesian(ol)\
    .filter(lambda x: x[0][0] != x[1][0])\
    .map(lambda x: (x[0][0], x[0][1], x[1][0], x[1][1],␣
 ↪jaccard_similarity(x[0][1],x[1][1])) )\
    .top(10, lambda x: x[4])
```

```
1: (8456, [15747, 3143, 18343, 14515, 4126], 10376, [15395, 2979, 18343, 13032,
3307, 16495, 17401], 0.09090909090909091)
2: (10376, [15395, 2979, 18343, 13032, 3307, 16495, 17401], 8456, [15747, 3143,
18343, 14515, 4126], 0.09090909090909091)
3: (4808, [1295, 17169, 17813, 8856, 155, 17566, 11683, 15524, 15395, 682,
19122, 8374, 17976, 6460, 5450, 4184, 16601, 5479, 7020, 2940, 3327], 10901,
[12473, 15428, 11462, 15629, 12206, 1295, 2039, 4184, 9529, 10142, 13687],
0.06666666666666667)
4: (10901, [12473, 15428, 11462, 15629, 12206, 1295, 2039, 4184, 9529, 10142,
13687], 4808, [1295, 17169, 17813, 8856, 155, 17566, 11683, 15524, 15395, 682,
19122, 8374, 17976, 6460, 5450, 4184, 16601, 5479, 7020, 2940, 3327],
0.06666666666666667)
5: (7532, [7680, 259, 1285, 17416, 10391, 18718, 6434, 17071, 13236, 2105,
16448, 4551, 846, 16338, 15572, 2652, 11741, 2151, 13424, 18163, 14710, 377],
5390, [7555, 4612, 13349, 14791, 16969, 5452, 846, 17233, 9073, 14710, 1533],
0.06451612903225806)
6: (5390, [7555, 4612, 13349, 14791, 16969, 5452, 846, 17233, 9073, 14710,
1533], 7532, [7680, 259, 1285, 17416, 10391, 18718, 6434, 17071, 13236, 2105,
16448, 4551, 846, 16338, 15572, 2652, 11741, 2151, 13424, 18163, 14710, 377],
0.06451612903225806)
7: (2489, [774, 6535, 16529, 6418, 8087, 6812, 18589, 13216, 10656, 5154, 8104,
11946, 12718, 15026, 2741, 8764, 7101, 7102, 10431, 18112, 4035, 3911, 3656,
15438, 4560, 16979, 15323, 1756, 1245, 1128, 1901, 8173, 241, 19442, 16120,
1658, 7806, 6399], 4283, [13060, 774, 12044, 18704, 11547, 11040, 297, 8764,
13506, 3656, 8651, 3405, 14545, 16340, 6485, 472, 5723, 16094, 13791, 1635,
12524, 1901, 6765, 2672, 11122, 13810, 1655, 5626, 4221], 0.06349206349206349)
8: (4283, [13060, 774, 12044, 18704, 11547, 11040, 297, 8764, 13506, 3656, 8651,
3405, 14545, 16340, 6485, 472, 5723, 16094, 13791, 1635, 12524, 1901, 6765,
2672, 11122, 13810, 1655, 5626, 4221], 2489, [774, 6535, 16529, 6418, 8087,
6812, 18589, 13216, 10656, 5154, 8104, 11946, 12718, 15026, 2741, 8764, 7101,
7102, 10431, 18112, 4035, 3911, 3656, 15438, 4560, 16979, 15323, 1756, 1245,
1128, 1901, 8173, 241, 19442, 16120, 1658, 7806, 6399], 0.06349206349206349)
9: (2768, [6112, 128, 300, 12367, 1648, 13813, 14935, 824, 19866, 14779, 19644,
1887], 4385, [13056, 10758, 10504, 14216, 19982, 13839, 14611, 17571, 10793,
19644, 7100, 19390, 15426, 11350, 11224, 2907, 8544, 15976, 1648, 14965, 7419,
3967], 0.0625)
10: (4385, [13056, 10758, 10504, 14216, 19982, 13839, 14611, 17571, 10793,
19644, 7100, 19390, 15426, 11350, 11224, 2907, 8544, 15976, 1648, 14965, 7419,
3967], 2768, [6112, 128, 300, 12367, 1648, 13813, 14935, 824, 19866, 14779,
19644, 1887], 0.0625)
```

[34]:

```python
for idx, i in enumerate(result, start=1):
    print("{:3}:  {:5}  {:5}".format(idx,i[0], i[4]))
```

```
  1:   8456  0.09090909090909091
  2:  10376  0.09090909090909091
  3:   4808  0.06666666666666667
  4:  10901  0.06666666666666667
  5:   7532  0.06451612903225806
  6:   5390  0.06451612903225806
  7:   2489  0.06349206349206349
  8:   4283  0.06349206349206349
  9:   2768  0.0625
 10:   4385  0.0625
```

[39]:

```python
# Q7 Answer

def combin(x):
    return [ '+'.join([str(i),str(j)]) for i,j in combinations(sorted(x),2) ]

l = lineitemsRDD.map(lambda x: (x[0], x[1]))

result = l\
    .combineByKey(lambda x:[x], lambda i,j:i+[j], lambda u,w:u+w)\
    .map(lambda x: (x[0], list(x[1])))\
    .filter(lambda x: len(x[1]) > 1)\
    .flatMap(lambda x: combin(x[1]))\
    .map(lambda x: (x, 1))\
    .reduceByKey(add)\
    .top(10, lambda x: x[1])

for idx, i in enumerate(result, start=1):
    print("{}: {}".format(idx, i))
```

```
1: ('6031+15277', 3)
2: ('14405+17144', 3)
3: ('11630+14244', 3)
4: ('11004+15109', 3)
5: ('364+3823', 3)
6: ('250+7045', 3)
7: ('12966+16068', 3)
8: ('5850+11561', 3)
9: ('595+11837', 3)
10: ('5085+10907', 3)
```

[ ]: