

Assignment

WEEK 06: GRAPH DATABASES

WEEK 07: MAPREDUCE

Teacher and teaching assistant:

David Kofoed Wind,

Finn Årup Nielsen

(Rasmus) Malthe Jørgensen

Ulf Aslak Jensen

WEEK 06

Exercise 6.1.	-----	3
Exercise 6.2.	-----	4
Exercise 6.3.	-----	5
Exercise 6.4	-----	6
Exercise 6.5	-----	7

WEEK 07

Exercise 7.1	-----	8
Exercise 7.2	-----	9
Exercise 7.3	-----	11

Exercise 6.1:

Graph Database, Neo4j Web UI, and Cypher query

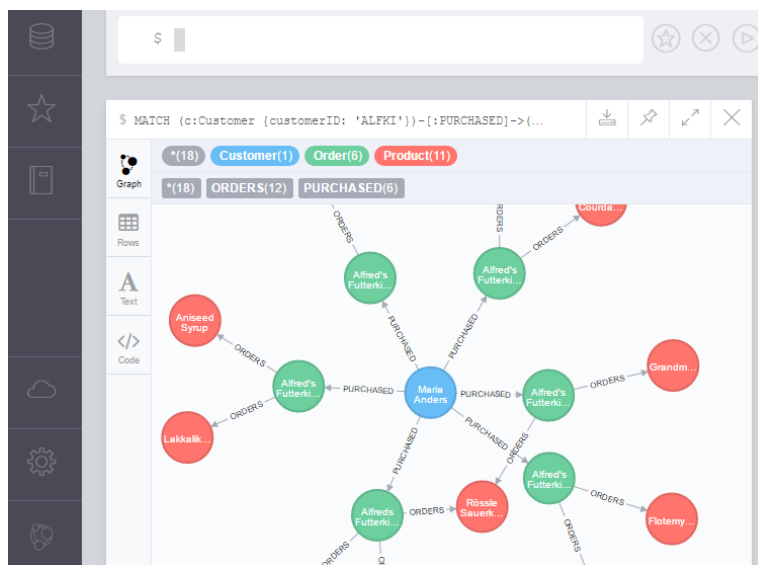
The exercise for the week is made by using the graph database service offered by Graph Story. Graph story would offer an instance which provides us to create database through Neo4j Web UI by inserting Cypher query.

Graphstory

neo-trial-scarlett-medhurst-orange		Actions
URL	https://neo-trial-scarlett-medhurst-orange.azure.graphstory.com:7473/browser/	
Host	neo-trial-scarlett-medhurst-orange.azure.graphstory.com	
HTTPS Port	7473	
BOLT Port	7687	
Username	neo_trial_scarlett_medhurst_orange	
Password	pP4yDK0JxFAwRBD2bl0Ypn818D9Na0bi4102Tuv	
Provider	azure	

The figure above shows a snapshot of the instance properties. By clicking the URL it will direct to Neo4j Web UI. The username and password is for accessing the instance.

Neo4j Web UI



The figure shows the interface of the Neo4j Web UI. User could run Cypher query in the command terminal. In the following exercise 6.2 to 6.5 would show the results of both graph and table to verify the solution.

Exercise 6.2:

The customer with customerID ALFKI has made a number of orders containing some products. Return all the orders made by ALFKI and the products they contain.

Solution

Cypher query for graph

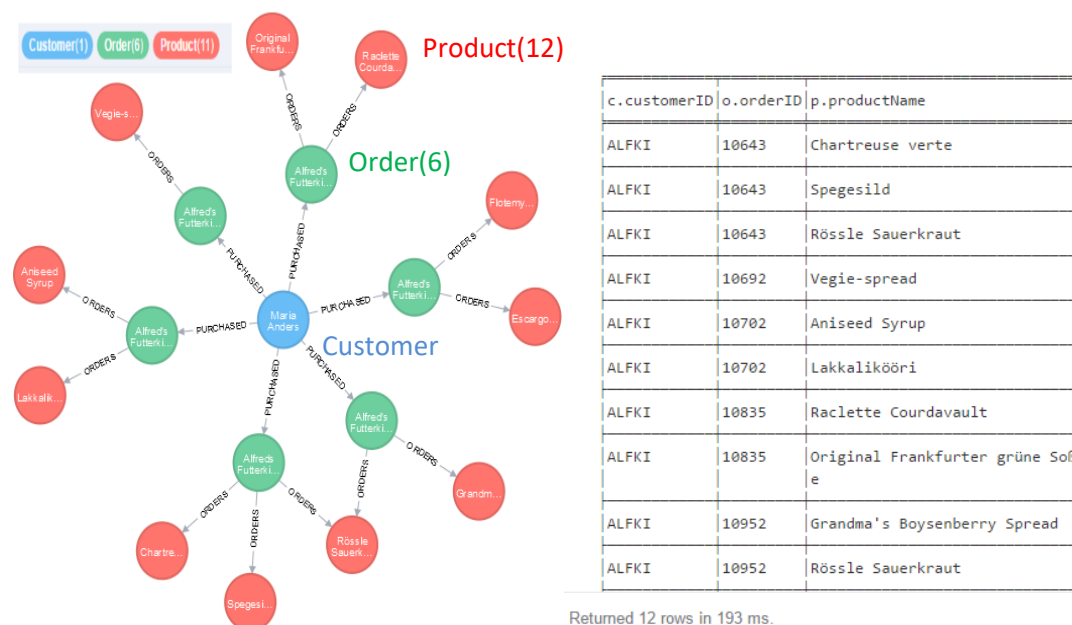
```
MATCH (c:Customer {customerID: 'ALFKI'})-[:PURCHASED]->(o:Order)-[:ORDERS]->(p:Product)
RETURN c, o, p
```

The property key customerID of the node c in table Customer is set to ALFKI. Then it uses the relationship PURCHASED to match the orderID which the customer has purchased. In the next step, it uses the relationship ORDERS to match all products has been ordered by the given orderID. Finally it returns the nodes of Customer, Order and Product.

Cypher query for table

```
MATCH (c:Customer {customerID: 'ALFKI'})-[:PURCHASED]->(o:Order)-[:ORDERS]->(p:Product)
RETURN c.customerID, o.orderID, p.productName
ORDER BY o.orderID
```

Result



The graph shows 12 products in total which are ordered from 6 different orders that purchased by ALFKI. The table shows 12 products information which 6 different orderIDs .

Exercise 6.3:

The customer with customerID ALFKI has made a number of orders containing some products. Return orders made by ALFKI that contain at least 2 products. Also return the products.

Solution

Cypher query for graph

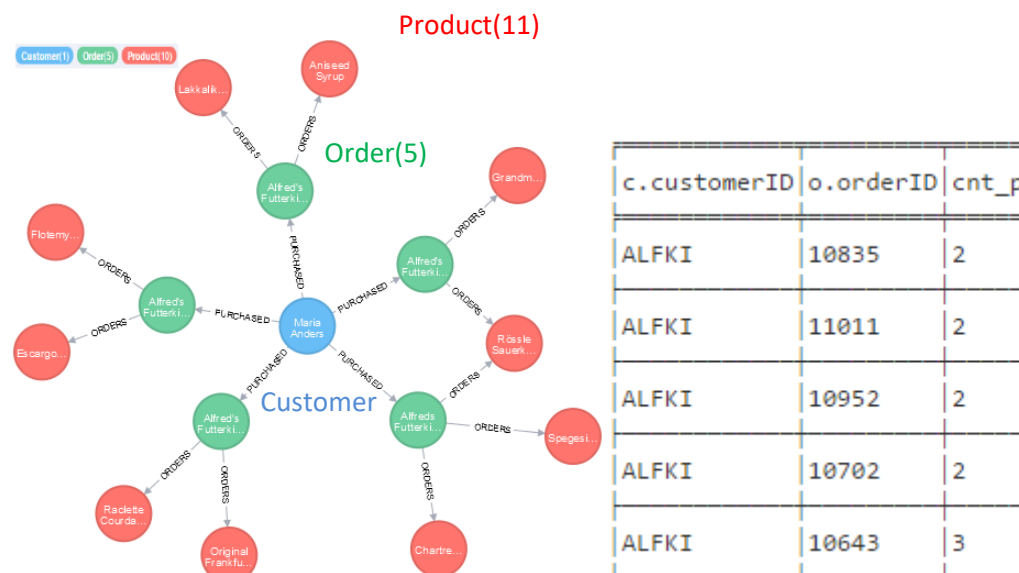
```
MATCH (c:Customer {customerID: 'ALFKI'})-[:PURCHASED]->(o:Order)-[:ORDERS]->(p:Product)
WITH o, COUNT(p) AS cnt_p, COLLECT(p) AS pd, c
WHERE cnt_p > 1
RETURN o, pd, c
```

The query for the MATCH part, it is exactly the same as 6.2. After that in the WITH clause it counts the products by given orderID and also collect the productName. In the WHERE clause it set the condition which the number of product should be larger than 1. Finally it returns the Order, Product and the Customer

Cypher query for table

```
MATCH (c:Customer {customerID: 'ALFKI'})-[:PURCHASED]->(o:Order)-[:ORDERS]->(p:Product)
WITH o, COUNT(p) AS cnt_p, COLLECT(p) AS pd, c
WHERE cnt_p > 1
RETURN c.customerID, o.orderID, cnt_p
```

Result



The graph shows 11 products ordered from 5 different orderIDs which at least has two products. The table shows there are 5 different orderIDs which have at least 2 products.

Exercise 6.4:

Determine how many and who has ordered “Uncle Bob’s Organic Dried Pears” (productID 7).

Solution

Cypher query for graph

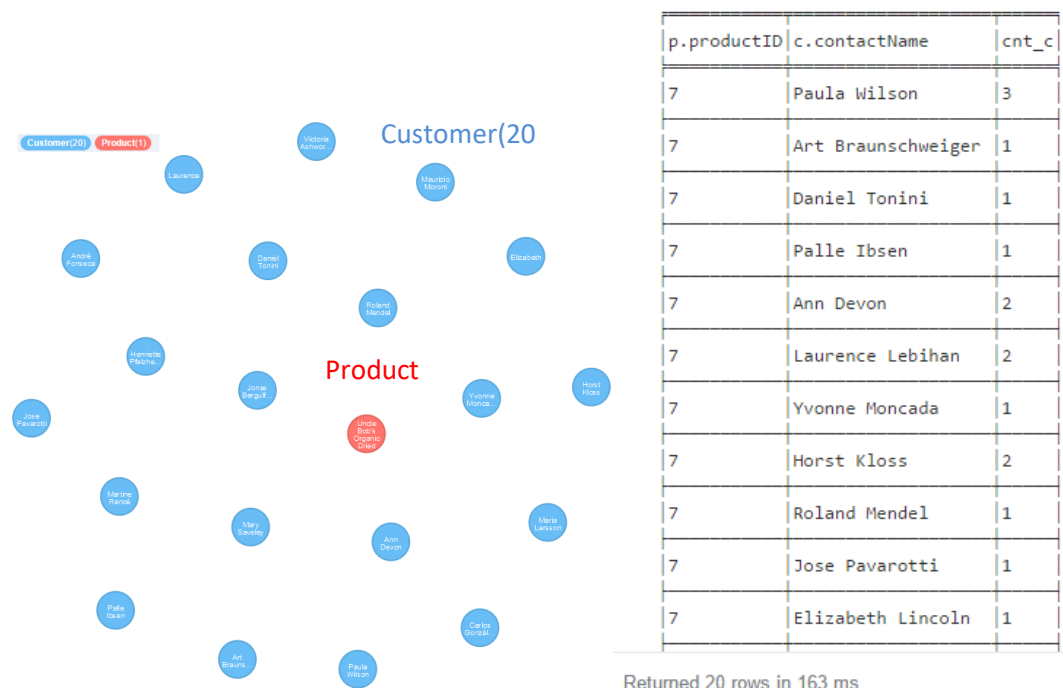
```
MATCH (p:Product {productID: '7'})<-[:ORDERS]-(o:Order)<-[:PURCHASED]-(c:Customer)
RETURN c
```

The property key productID is set to 7. The query used the relationship ORDERS to match the orderID which contains the productID. Then it finds out the customers who has ordered the products by the relationship of PURCHASED. Finally it returns to the Customer.

Cypher query for table

```
MATCH (p:Product {productID: '7'})<-[:ORDERS]-(o:Order)<-[:PURCHASED]-(c:Customer)
WITH c, COUNT(c) as cnt_c, p
RETURN p.productID, c.contactName, cnt_c
```

Result



The graph shows 20 customers have order the product. The table returns 20 rows information or the customers .

Exercise 6.5:

How many different and which products have been ordered by customers who have also ordered “Uncle Bob’s Organic Dried Pears”?

Solution

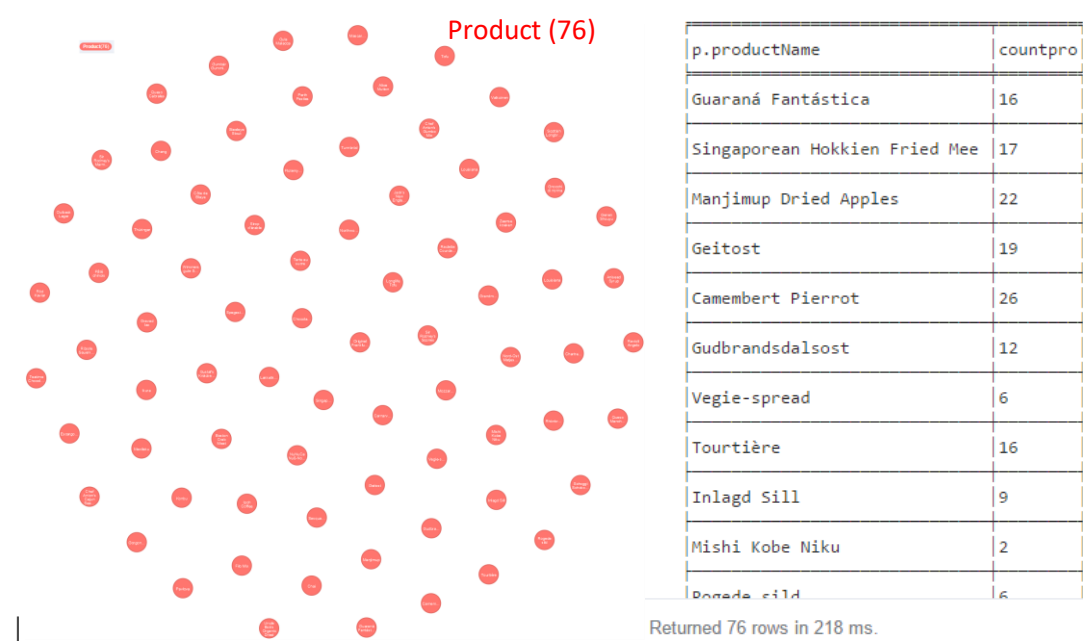
Cypher query for graph

```
MATCH (n:Product{productID:'7'})<-[:ORDERS]-(o:Order)-[:PURCHASED]-(c:Customer)-[:PURCHASED]-(orders2)-[:ORDERS]-(p:Product)
WITH p, count(p) as countpro
RETURN p
```

Cypher query for table

```
MATCH (n:Product{productID:'7'})<-[:ORDERS]-(o:Order)-[:PURCHASED]-(c:Customer)-[:PURCHASED]-(orders2)-[:ORDERS]-(p:Product)
WITH p, count(p) as countpro
RETURN p.productID, p.productName, countpro
```

Result



The graph shows there are 76 products has been ordered by the customers who has ordered the same product(productID 7). The table gives the same result which returns 76 rows of the products information.

Exercise 7.1:

Define and implement a MapReduce job to count the occurrences of each word in a text file. Document that it works with a small example.

Solution

```
#!/usr/bin/python
from mrjob.job import MRJob
from mrjob.step import MRStep
import re
import time
WORD_RE = re.compile(r"[\w']+")
class MRWordCount(MRJob):

    def mapper(self, key, line):
        for word in WORD_RE.findall(line):
            yield word.lower(), 1

    def reducer(self, key, values):
        yield key, sum(values)

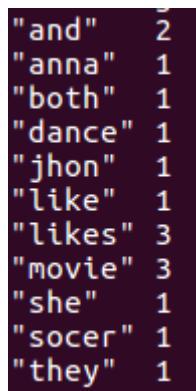
if __name__ == '__main__':
    MRWordCount.run()
```

The mapper set each distinct word from given text input as a key and set each occurrence as one. The reducer sums up all the occurrences for each key.

Result

```
Jhon likes movie and socer.
Anna likes movie and she likes dance.
They both like movie.
```

The context above is the text input to the map reducer.



"and"	2
"anna"	1
"both"	1
"dance"	1
"jhon"	1
"like"	1
"likes"	3
"movie"	3
"she"	1
"socer"	1
"they"	1

The figure above shows each word and the number are the occurrences for each word.

Exercise 7.2:

Define and implement a MapReduce job that determines if a graph has an Euler tour

Solution

```
#!/usr/bin/python
from mrjob.job import MRJob
from mrjob.step import MRStep
import re
import sys
import time
WORD_RE = re.compile(r"[\w']+")

class MR_euler_tour(MRJob):
    #mapper
    def mapper(self, key, line):
        for elem in line.split():
            yield elem, 1
    #reducer
    def reducer(self, key, values):
        v = [v for v in values]
        if sum(v) % 2 != 0:
            print ("Node%s has odd degree of %d" % (key, sum(v)))
            sys.exit()
        yield key, sum(v)
        del v[:]
#Run MRJob
if __name__ == '__main__':
    MR_euler_tour.run()
#If program didn't stop it the graph Euler Tour
print "The graph has Euler Tour"
```

The mapper set each nodes and set each occurrence as one. The reducer counts occurrences of each node and determine if the graph is a Euler Tour. By the definition of the Euler Tour, each nodes has to have even degree. In contrast, the graph does not have a Euler Tour. If a node has odd degree the program would be stopped. Before it stops, it outputs the reason such that which node has odd degree and the numbers of the degree. If all nodes have even degree the program would say the graph has Euler Tour.

Result of 1st graph

```
wen@wen:~/work/repos/bigwork2016/week7/wen/ex7.2$ ./euler_tour.py d1_eulerGraphs.txt
No configs found; falling back on auto-configuration
Creating temp directory /tmp/euler_tour.wen.20161016.142506.791174
Running step 1 of 1...
Streaming final output from /tmp/euler_tour.wen.20161016.142506.791174/output...
"0"      2
"1"      2
"2"      2
"3"      2
Removing temp directory /tmp/euler_tour.wen.20161016.142506.791174...
The graph has Euler Tour
```

The figure shows the first graph has Euler Tour since all nodes have even degree.

Result for 2nd graph

```
wen@wen:~/work/repos/bigwork2016/week7/wen/ex7.2$ ./euler_tour.py d2_eulerGraphs.txt
No configs found; falling back on auto-configuration
Creating temp directory /tmp/euler_tour.wen.20161016.142550.349021
Running step 1 of 1...
Node0 has odd degree of 3
```

The 2nd graph doesn't have Euler Tour since the program finds out the node 0 has odd degrees of 3.

Since the some of the graph has such long data of nodes, the figures would only show the ending part of the program output.

Result for 3rd graph

```
wen@wen:~/work/repos/bigwork2016/week7/wen/ex7.2$ ./euler_tour.py d3_eulerGraphs.txt
"4"      4
"5"      4
"6"      4
"7"      4
"8"      2
"9"      4
Removing temp directory /tmp/euler_tour.wen.20161016.142701.092520...
The graph has Euler Tour
```

The 3rd graph does have Euler Tour since all nodes have even degree.

Result for 4th graph

```
wen@wen:~/work/repos/bigwork2016/week7/wen/ex7.2$ ./euler_tour.py d4_eulerGraphs.txt
"95"     4
"96"     4
"97"     4
"98"     4
"99"     4
Removing temp directory /tmp/euler_tour.wen.20161016.142828.922085...
The graph has Euler Tour
```

The 4th graph does have Euler Tour since all nodes have even degree.

Result for 5th graph

```
wen@wen:~/work/repos/bigwork2016/week7/wen/ex7.2$ ./euler_tour.py d5_eulerGraphs.txt
No configs found; falling back on auto-configuration
Creating temp directory /tmp/euler_tour.wen.20161016.143209.130602
Running step 1 of 1...
Node1 has odd degree of 3
wen@wen:~/work/repos/bigwork2016/week7/wen/ex7.2$
```

The fifth graph does not have a Euler Tour since node 1 has odd degree of 3.

Exercise 7.3:

Make a MapReduce job which counts the number of triangles in a graph.

Solution:

```
#!/usr/bin/python
from mrjob.job import MRJob
import re

WORD_RE = re.compile(r"[\w']+")

class Count_triangles(MRJob):
    def steps(self):
        return [
            self.mr (mapper=self.mapper_1,
                    reducer=self.reducer_1),
            self.mr (mapper= self.mapper_2,
                    reducer=self.reducer_2),
            self.mr (mapper=self.mapper_3,
                    reducer=self.reducer_3),
            self.mr (mapper=self.mapper_div_duplicate),
        ]

    ##convert to undirect graph
    def mapper_1(self, _, line):
        yield line.split()[0],line.split()[1]
        yield line.split()[1],line.split()[0]

    ##reduce to edge list
    def reducer_1(self, key, value):
        yield key, self.concatenate(value)

    ##rind each vertex's edge
    def mapper_2(self, key, value):
        for element in value:
            yield (min(key, element),max(key, element)), value

    def reducer_2(self, key, value):
        yield key, self.concatenate(value)

    def mapper_3(self, key, value):
        yield None,
len(self.concatenate(set(value[0]).intersection(value[1])))

    def recuder_3(self, key, value):
        yield 'Total Number of Tirangle:', sum(value)

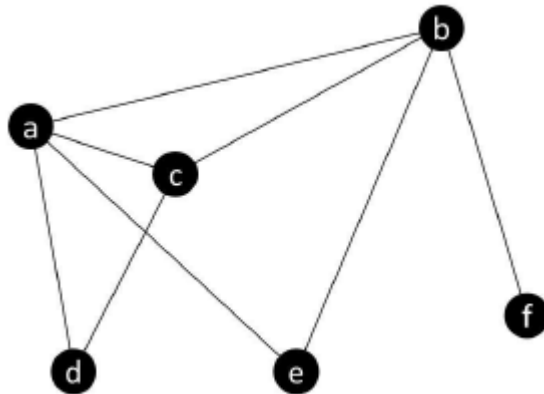
    ##dived 3 to remove duplicated path
    def mapper_div_duplicate(self, key, value):
        yield 'Total Number of Tirangle: ', value/3

    ##concaternate the nodes
    def concatenate(self, input_value):
        concatenate_list = []
        for i in input_value:
            concatenate_list.append(i)
        return concatenate_list

if __name__ == '__main__':
    Count_triangles.run()
```

Result

```
1 0
2 0
3 0
4 0
2 1
4 1
5 1
3 2
```



The data above is an directed adjacency list which contains 3 triangles. The isomorphic graph could be assume as the diagram above

```
"Total Number of Tirangle: " 3
Removing temp directory /tmp/find_triangle.wen.20161016.192819.841377...
```

The result from the program also gives same result.

Result for the roadNet-CA.mtx file.

```
Streaming final output from /tmp/find_triangle.wen.20161016.193502.113825/output...
"Total Number of Tirangle: " 120676
Removing temp directory /tmp/find_triangle.wen.20161016.193502.113825...
```

Compare to the given solution the program gives the same result.