Exercise Sheet 3, 2019

6.0 VU Advanced Database Systems

Gent Rexha (11832486), Princ Mullatahiri (11846033)

10.06.2019

Exercise 1 (Distributed Joins) [2 Points]

a)

Relation	Size	#Records	Record size (bytes)
Users	1	40,000	800
Orders	2	900,000	30
Products	3	1,500	2000

Relation	Attribute	Size (bytes)
Orders	price	4
Orders	prod	8
Orders	user	8
Users	uid	8
Users	name	100
Products	pid	8

$$\pi_{name,price}(Users \bowtie_{uid=user} Orders)$$

I.

$$(40000 \times 800) + (900000 \times 30) = 32000000 + 27000000 = 59000000$$

II.

$$(40000 \times 800) + (40000 \times 900000 \times \frac{1}{360000} \times (100 + 4)) = 32000000 + 100000 \times 104 = 42400000$$

III.

$$(900000 \times 30) + (40000 \times 900000 \times \frac{1}{360000} \times (100 + 4)) = 27000000 + 100000 \times 104 = 37400000$$

IV.

Send Users' to site 2 : $40000 \times 8 = 320000$

Semi join with Orders send result to site 1 : $100000 \times 12 = 1200000$

Compute result at site 1 and send result to site 4: $100000 \times 104 = 10400000$

Total Cost: 11920000

V.

Send Orders' to site 1 : $900000 \times 8 = 7200000$

Semi join with Users send result to site 2 : $100000 \times 108 = 10800000$

Compute result at site 2 and send result to site 4: $100000 \times 104 = 10400000$

Total Cost: 28400000

a*)

I.

$$(40000 \times 108) + (900000 \times 12) = 4320000 + 10800000 = 15120000$$

II.

$$(40000 \times 108) + (40000 \times 900000 \times \frac{1}{360000} \times (100 + 4)) = 4320000 + 100000 \times 104 = 14720000$$

III.

$$(900000 \times 12) + (40000 \times 900000 \times \frac{1}{360000} \times (100 + 4)) = 10800000 + 100000 \times 104 = 21200000$$

For IV and V we already make projection as soon as possible so the result would not change.

b) $Products \bowtie \pi_{name,price,prod}(Users \bowtie_{uid=user} Orders)$

For B in the first case we only take the best result that is IV scenario and add products to projection.

Send Users' to site 2 : $40000 \times 8 = 320000$

Semi join with Orders send result to site 1 : $100000 \times 20 = 2000000$

Compute result at site 1 and send result to site 4: $100000 \times 112 = 11200000$

Total Cost: 13520000

And for the join with Products because there is no projection in there and the record cost is too big it is not recommended to make a join and than transform data in site 4 it is much better if we just send all the data at site 4 and then compute join. From the first result we have 100000 rows with size of 112 for record.

$$(100000 \times 112) + (1500 \times 2000) = 11200000 + 3000000 = 14200000$$

So the total cost is : $13520000 + 14200000 = 27720000$

Exercise 2 (Denormalization) [3 Points]

Requirements:

- Two queries should be fast:
  - Listing all people who have borrowed a book for more than two weeks and not returned it yet (not returned means no to value in the borrowed relation).
  - Asking if any edition of a book is currently available to be borrowed. (i.e., owned - borrowed but not yet returned books > 0)
- The borrowed relation is updated frequently (and needs to be efficient). For all other relations, updates are rare and can be more expensive.

To solve the requirements the tables firstly were transformed into separate JSON objects. After analyzing both queries we deciced that the first query can be answerd by the relation `BorrowedBooks` where we save all the books that are borrowed and not returned yet, also we have another relation `ReturnedBooks` where we save the history of returned books, in this way we save time during searching phase because there are less

rows to check for each book that is not returned yet. For the second query we decided to add a new field to the `Edition` collection named `borrowedby` which keeps track of the people who have the current document (Book Edition) borrowed and not yet returned. This allows for a quick calculation of the available books by substracting the length of the `borrowedby` array to the `owned` value.

Although this quick calculation comes with a price. that everytime a book is returned the `Edition` with its `borrowedby` field has to be updated.

Exercise 3 (Graph Databases) [5 Points]

a)

1.)

```
MATCH (n)-[:REGISTERED_ADDRESS]->(a:Address)
    RETURN DISTINCT(a.countries) AS country
    ORDER BY country;
```

2.)

```
MATCH (e:Entity)-[:REGISTERED_ADDRESS]->(a:Address)
    WHERE a.countries CONTAINS 'Albania'
    RETURN e.name, a.countries
    LIMIT 5;
```

b)

1.)

```
MATCH (i:Intermediary)-[:INTERMEDIARY_OF]->(n)
                RETURN i.name AS Name, COUNT(*) AS frequency
                ORDER BY frequency DESC LIMIT 10;
```

2.)

```
MATCH a=(i:Intermediary)-[:INTERMEDIARY_OF |:OFFICER_OF]->(n)
                RETURN i.name AS Name, COUNT(*) AS frequency
                ORDER BY frequency DESC LIMIT 10;
```

3.)

```
MATCH a=(i:Intermediary)-[e]->(n)
                WHERE type(e) <> "OFFICER_OF" AND type(e) <> "INTERMEDIARY_OF"
                RETURN i.name AS Name, COUNT(*) AS frequency
                ORDER BY frequency DESC LIMIT 10;
```

c)

```
Match path = shortestpath( (f:Address{countries:'Luxembourg'})-[*]-(p:Address{countries:'Cyprus'}))
with path,f,p
WHERE length(path)>15 AND length(path)<31
return f.node_id,p.node_id, length(path)
LIMIT 1
```

d)

```
MATCH (i:Intermediary)-[:INTERMEDIARY_OF]->(e:Entity)<-[:OFFICER_OF]-(o:Officer)
    with i, e, o
    MATCH (i)-[:INTERMEDIARY_OF]->(e2:Entity)<-[:OFFICER_OF]-(o)
    WHERE e.name <> e2.name AND e.countries <> e2.countries AND e.country_codes <> e.jurisdiction
    return i, e, o, e2
    LIMIT 1;
```

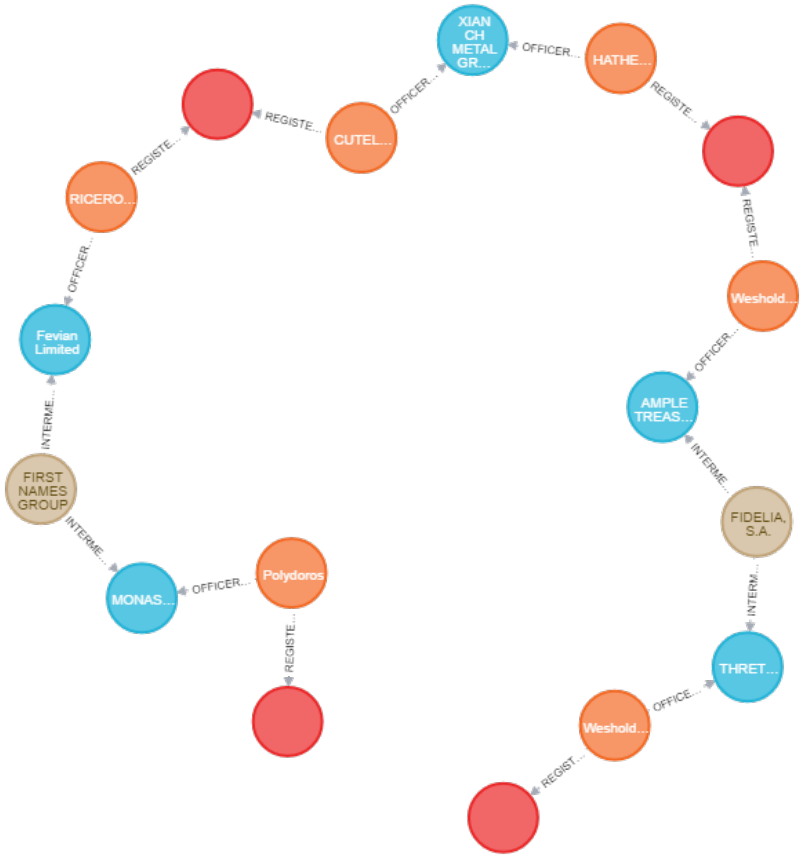

Figure 1: Graph of query C

e)

Creating the node Countries

```
MATCH (o:Other) WHERE NOT o.countries contains ';' AND EXISTS(o.countries)
WITH o
MERGE (c:Country {name:o.countries})
MERGE (o)-[:IN_COUNTRY]->(c)
ON CREATE SET c.name = o.countries
RETURN DISTINCT(o.countries)
```

Show the new created node with its relationship

```
match p= (:Other)-[:IN_COUNTRY]->(c:Country) where c.name = 'Ireland'  return p
```

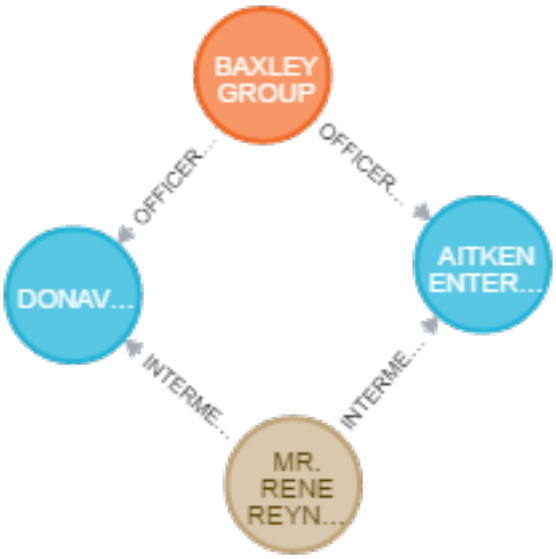

Figure 2: Graph of query d

Figure 3: Graph of query E