TU WIEN

DATA SCIENCE

184.780 Advanced Database Systems

Exercise 3

Mal Kurteshi 11924480

May 21, 2021

1 Exercise 1: Distributed Joins

(a)

(I) Send both tables to site 4 and join there

- Customers to site 4 : 25000 * 500 = 12500000 bytes
- Contracts to site 4 : 400000 * 80 = 32000000 bytes
- Total Communication cost : 12500000 + 32000000 = 44500000 bytes

(II) s. Customers to site 2, join there and send result to site 4

- s. Customers to site 2: 25000 * 500 = 12500000 bytes
- no. of rec. after join : $\frac{1}{40000}$ * 400000 * 25000 = 250000 bytes
- rec. send result to site 4: 250000 * (78+8) = 21500000 bytes
- Total Communication cost : 21500000 bytes

(III) Symmetrically: Send Contracts to site 1, join there and send the result to site 4.

- s. Customers to site 1: 400000 * 80 = 32000000 bytes
- no. of rec after join: $\frac{1}{40000}$*400000*25000 = 250000 bytes
- rec. Send result to site 4 :250000*(78+8) = 21500000 bytes
- Total Communication cost: 32000000 + 21500000 = 53500000 bytes

(IV) Send only the join attributes of Customers to site 2, semi join with Contracts, send the result back to site 1 to compute the full join. Finally transfer the full join to site 4.

- s. Customers to site 2: 25000 * 16 = 400000 bytes
- s. join to site 1: ($\frac{1}{40000}$*400000*25000)*80 = 20000000 bytes
- s. Result to site 4: ($\frac{1}{40000}$*400000*25000)*(78+8) = 21500000 bytes
- Total Communication Cost: 400000 + 20000000 + 21500000 = 41900000 bytes

(V) The semi-join strategy in the opposite direction.

- s. Contracts to site 1: 400000 * 8 = 3200000
- s. Result of join to site 2: ($\frac{1}{40000}$*400000*25000)*500 = 125000000 bytes
- s. Result to site 4: ($\frac{1}{40000}$*400000*25000)*(78+8) = 21500000 bytes

- Total Communication Cost: 3200000 + 125000000 + 21500000 = 149700000 bytes

(a*)

(I) Send both tables to site 4 and join there

- Customers to site 4: 25000*(16 + 78) = 2350000 bytes
- Contracts to site 4: 400000*(8+8) = 6400000 bytes
- Total Communication cost: 2350000 + 6400000 = 8750000 bytes

(II) Send Customers to site 2, join there and send result to site 4

- s. Customers to site 2: 25000*(16 + 78) = 2350000 bytes
- no. of records after join: $\frac{1}{40000}$*400000*25000 = 250000 records
- res to site 4: 250000*(78+8) = 21500000 bytes
- Total Communication cost: 2350000 + 250000 + 21500000 = 24100000 bytes

(III) Symmetrically: Send Contracts to site 1, join there and send the result to site 4

- s. Customers to site 1: 400000 * (8 + 8) = 6400000 bytes
- no. of records after join: $\frac{1}{40000}$*400000*25000 = 250000 records
- res to site 4 : 250000*(78+8) = 21500000 bytes
- Total Communication cost: 6400000 + 250000 + 21500000 = 28150000 bytes

(IV) Send only the join attributes of Customers to site 2, semi join with Contracts, send the result back to site 1 to compute the full join. Finally transfer the full join to site 4

- s. Customers to site 2: 25000 * 16 = 400000 bytes
- join to site 1: $\frac{1}{40000}$*400000*25000*(7+8) = 3750000 bytes
- res to site 4: $\frac{1}{40000}$*400000*25000*(78+8) = 21500000 bytes
- Total Communication Cost: 400000 + 3750000 + 21500000 = 25650000 bytes

(V) The semi-join strategy in the opposite direction

- s. Contracts to site 1: 400000 * 8 = 3200000
- res of join to site 2: ($\frac{1}{40000}$*400000*25000)*(78+16) = 23500000 bytes

- res to site 4: ($\frac{1}{40000}$*400000*25000)*(78+8) = 21500000
- Total Communication Cost: 3200000 + 23500000 + 21500000 = 48200000 bytes

(b) For this case we will take the best result achieved in the a* which is the result IV.

- s. Customers to site 2: 25000 * 16 = 400000 bytes
- s. join to site 1: $\frac{1}{40000}$*400000*25000*(7+8) = 3750000 bytes
- s. Result to site 4: $\frac{1}{40000}$*400000*25000*(78+8) = 21500000 bytes
- s. Services to site 4: 1400*5000 = 7000000 bytes
- Total Communication Cost: 3200000 + 23500000 + 21500000 + 7000000 = 55200000 bytes

2 Exercise 2 : Denormalization

To solve the requirements we have firstly transformed tables into separate JSON objects. From the definition of the JSONs we can see that the first query can be answered by the Rented relation taking to consideration that it contains all the information needed. As for the second query i have added an additional field to the Rented collection name owned, which tracks the case of rented bicycles and not returned ones. This will give the situation to quickly calculate the available bicycles by simply subtracting the length of the rented array to the value owned.

3 Exercise 3: Graph Databases

(a) (I) List all the distinct universities where parlament members have studied.

```
MATCH (p:ParlamentMember) -[:STUDIED_AT]->
(u:University)
Return DISTINCT(u.name) AS ‘University‘
```

(II) Extend this list by also outputting how often the university occurs. Your output should thus consist of pairs of a university's name and the number of parlament members that have studied at the university.

```
MATCH (p:ParlamentMember) -[:STUDIED_AT]->
(u:University)
Return DISTINCT(u.name) AS ‘University‘,
count(p) as ‘No of Parlament members‘
```

(b) (I) First, find the 20 top places by the number of parlament members coming from them, i.e. those 20 places that have the most outgoing BORN_IN edges. Output the name of the place as well as the number of relevant edges. Order the output in descending order by the number of outgoing BORN_IN edges

```
MATCH (:ParlamentMember)-[BORN_IN]->(p: Place)
WITH p,count(*) AS edgecount
RETURN p.name, edgecount ORDER BY edgecount DESC LIMIT 20
```

(II) Find the top 20 districts by the number of parlament members. Note that places are contained in districts.

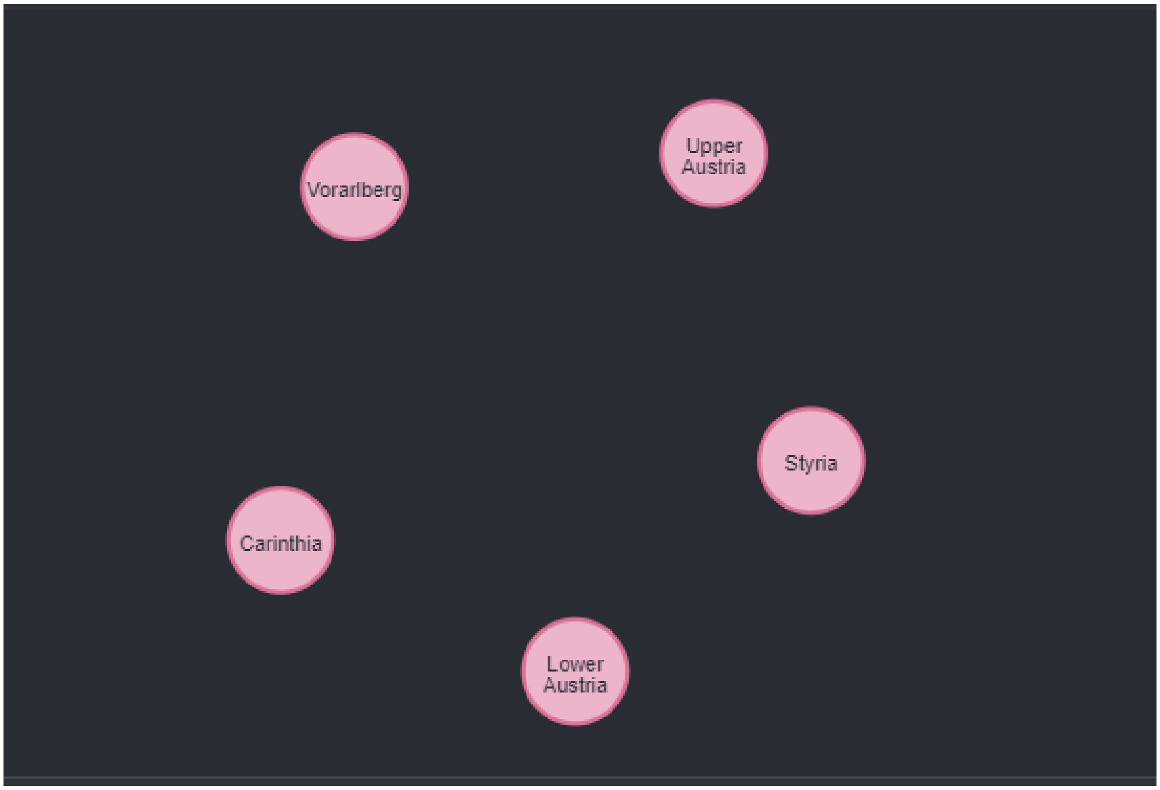
```
MATCH (:ParlamentMember)-[BORN_IN]->(p: Place),
(p)-[LOCATED_IN]->(d:District)
WITH d,count(*) AS edgecount
RETURN d.name, edgecount ORDER BY edgecount DESC LIMIT 20
```

(III) Find the top 10 federal states by the number of Persons (not Parlament-Members) who are a member of the largest two political parties.

```
MATCH (p:PoliticalParty) <-[MEMBER_OF]-() WITH p, count(*) AS partyCount
ORDER BY partyCount DESC LIMIT 2
 MATCH (f:FederalState)<-[:LOCATED_IN*2]-()<-[BORN_IN]-
(n:Person)-[MEMBER_OF]->(:PoliticalParty {name:p.name})
WHERE NOT n:ParlamentMember
WITH f, count(*) AS stateCount
RETURN f ORDER BY stateCount DESC LIMIT 5
```

(c) Find one (use LIMIT 1) subgraph of the form specified in Figure 4 in the database. Make sure to match one person who was a parlament member and one who was not, but is from the same place, a member of the same party and studied at the same university.

```
MATCH
p=((pl:Place)<-[:BORN_IN]-(p1:Person)-[:MEMBER_OF]->(:PoliticalParty)
<-[:MEMBER_OF]-(p2:Person)-[:BORN_IN]->(pl))
RETURN p
LIMIT 1
```

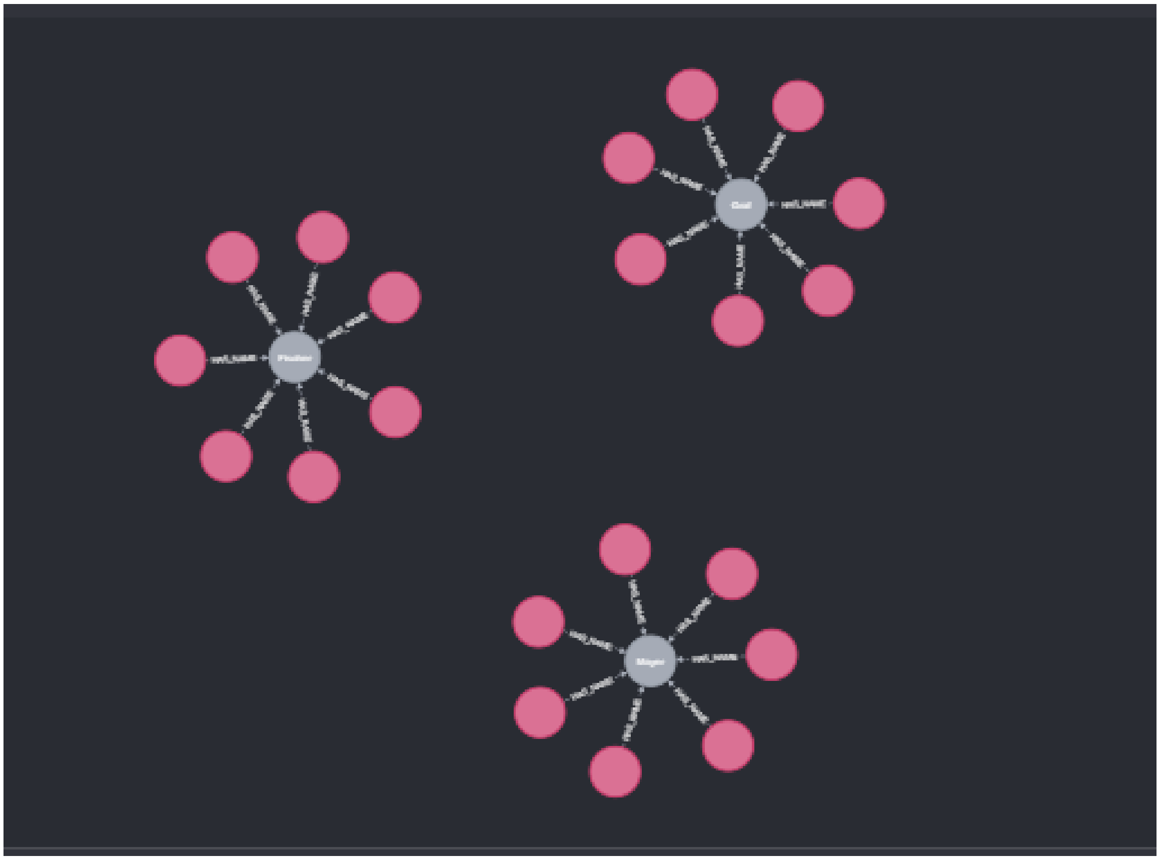


(d) We want to make the lastName attribute a real part of the graph. The best way to do this is in Neo4j is using the MERGE keyword inside of a MATCH. Write a query that for each node p with label ParlamentMember performs the following actions: Add an edge with type HAS_NAME going to a node with label Name that has a name attribute that matches the contents of p.name.Make sure that you don't create duplicate countries or duplicate edges. If the node p has no name attribute, do nothing for this node. When done, write a query to show all the nodes connected to the 3 most common last names of parlament members in the database.

```
MATCH (p:ParlamentMember) WHERE exists(p.last_name)
MERGE (n:Name { name : p.last_name})
MERGE (p)-[r:HAS_NAME]->(n)
RETURN p.last_name, type (r),n.name
```

```
MATCH (n:Name)<-[:HAS_NAME]-(p:ParlamentMember)
WITH n, count(p.last_name) AS c
ORDER BY c DESC
limit 3
MATCH
(p)-[r]->(n)
```

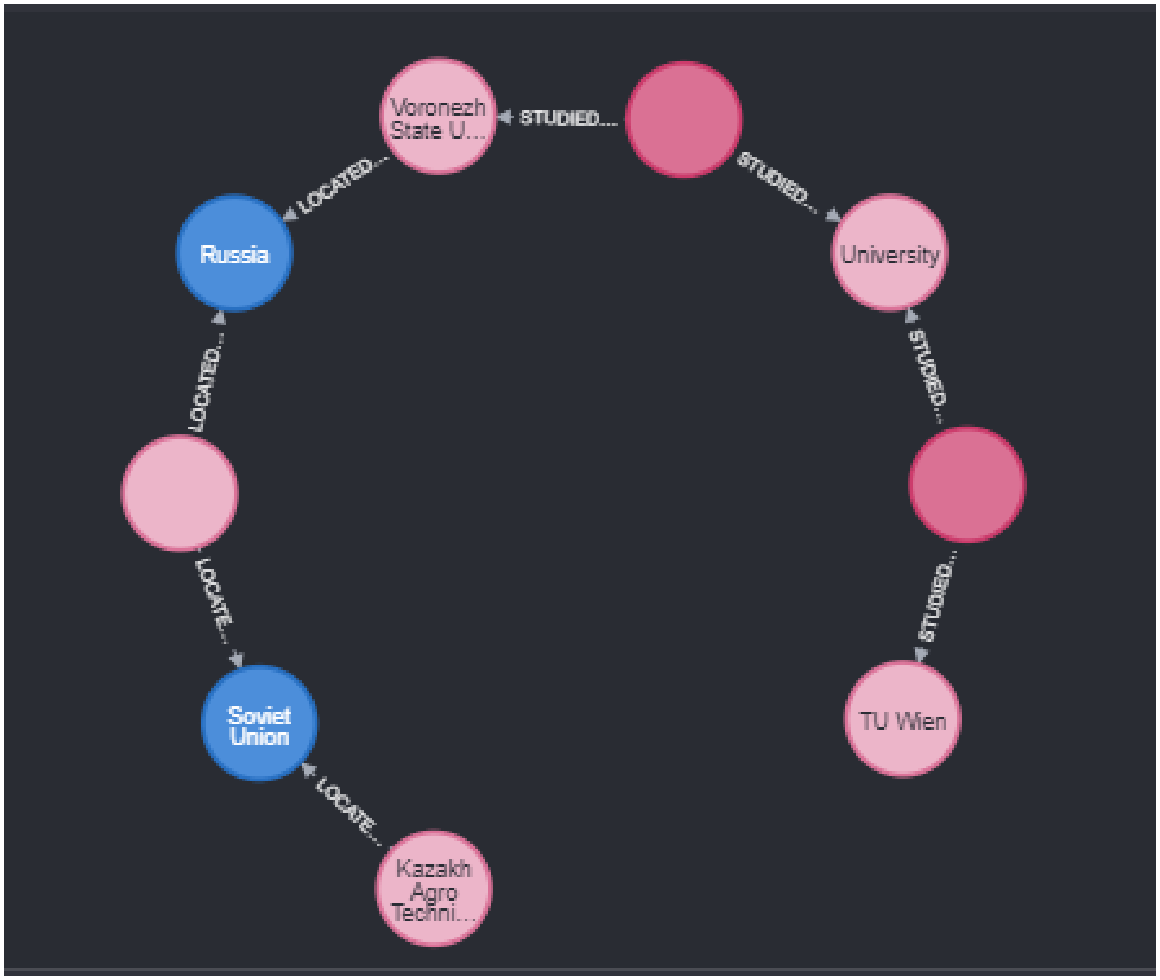
```
RETURN
p,r,n
```



(e) Consider the graph extended with the following additional relationship: any two universities u and v, for which a person exists who studied at both u and v, are connected. On the extended graph, find a shortest path between the "TU Wien" and the "Kazakh Agro Technical University". The shortest path can include relationships of any direction and label. Neo4j provides the shortestPath function for these type of problems.

```
create (u:University)<-[:STUDIED_AT]-(p:Person)-[:STUDIED_AT]->(v:University)
```

```
MATCH (tu:University)
WHERE tu.name CONTAINS "TU Wien"
MATCH (ku:University)
WHERE ku.name CONTAINS "Kazakh Agro Technical University"
MATCH p=shortestPath((tu)-[*0..10]-(ku))
RETURN p
LIMIT 1
```


Voronezh State U…
STUDIED…
STUDIED…
University
LOCATED…
Russia
LOCATED…
STUDIED…
LOCATE…
STUDIED…
Soviet Union
TU Wien
LOCATE…
Kazakh Agro Techni…