Ludovic Carré, Maxime Deloche, Vincent Lefoulon

Projet neo4j: Social network

I Application domain

Our database contains data on a social network. There are three different kind of entities in our social network: persons, companies and places.

Person nodes have the following properties: firstName, lastName, and born (date of birth). Each Person node also has labels corresponding to its nationality/ies. To have a simpler dataset, we only have 3 different nationalities (French, German and English) with several Persons having multiple (2 or 3) nationalities. A Person can be friend with another one (relation ‘FRIENDS’, N-M), and married (relation ‘MARRIED’, 1-1).

Company nodes have the following properties: a unique ‘name’ and a revenue (‘CA’). A Person can work for a company, corresponding to a ‘WORKS’ relation (1-N, with properties ‘salary’ and ‘position’). The ‘position’ property has 3 possible values : worker, executive or engineer.

Place nodes only have one property, the name of the place they represent. Each Company is located in a Place (relation ‘LOCATED’, 1-N), and each Person lives in a Place (relation ‘LIVES’, 1-N).

We therefore have 6 node types : Person, English, French, German, Company and Place.

Finally, there are uniqueness constraints on the names of a Company and a Place.

II Indices

We created an index on the “firstName” and “lastName” properties of the Person label to speed up the first request used to compute how close are two persons on the social network. This request must do a search on firstName and lastName of the two nodes considered before computing their proximity on the network.
Without the index, the request takes 34ms to complete and when we add the index, it only takes 5ms.

There are two other indices on the “name” property of Company and Place labels that are automatically created when adding the “UNIQUE” constraint on those labels.

III Requests

1. We compute the degree of proximity between two persons in the social network which we define as the length of the shortest path between the two nodes in the graph. A value of 1 means that the two persons are friends, a value of 2 that they are not friends but have a friend in common, a value of 3 that they are not friends and have no friends in common but that they have a friend of a friend in common, … Adding the index on Person(firstName, lastName) speeds up the request because it will be used to search for the two specified nodes in the list.
   Execution time without index: 34ms.
   Execution time with index: 5ms.

```
MATCH (a:Person { firstName: "Marie", lastName: "Lariviere" }),
   (b:Person { firstName: "Daniela", lastName: "Muller" }),
   p = shortestPath((a)-[:FRIEND*]-(b))
RETURN length(p);
```

2. We wish to know the number of persons by nationality that are working and were born after a given year, here 1980.

```
MATCH (n:Person)-[:WORKS]-() WHERE n.born < 1980
RETURN labels(n), count(*);
```

3. We want to know the maximum degree of proximity in our social network. Thus, we want to find the two nodes that are farthest from each other and get the chain of friends that link them together.

```
MATCH (a:Person),(b:Person), p = shortestPath((a)-[:FRIEND*]-(b))
WHERE a <> b
RETURN max(length(p)), p ORDER BY length(p) DESC LIMIT 1;
```

4. Given a company name, we want to know the number of persons of a given nationality that work for this company. This request will utilize the index automatically created on the Company name property when we created the unique constraint on this field.

```
MATCH (p:Person)-[:WORKS]-(c:Company{name:'Google'})
RETURN c.name, filter(x in labels(p) where x <> "Person"), count(*)
ORDER BY c.name;
```

5. For all companies in the social network, we want to get the proportion of persons working in the company that have more than one nationality.

```
MATCH (p:Person)-[:WORKS]-(c:Company)
WHERE length(labels(p)) > 2
WITH c.name as companyName, count(*) as nbMulti
MATCH (pp:Person)-[:WORKS]-(cc:Company)
WHERE cc.name = companyName
RETURN cc.name, (nbMulti*1.0)/count(*) as percentage
ORDER  BY percentage DESC;
```

6. We want to find the mean salary of every kind of position for all the companies in our social network, with the place where the company is located.

```
MATCH (:Person)-[w:WORKS]-(c:Company)-[:LOCATED]-(p:Place)
RETURN w.position, c.name, p.name, avg(w.salary)
ORDER BY c.name, avg(w.salary) desc;
```