OPEN DATA
Neo4j Laboratory Assignment 1

Authors
Goktug Cengiz (Y6545603R)
Jorge Lopez Alonso (71741290N)

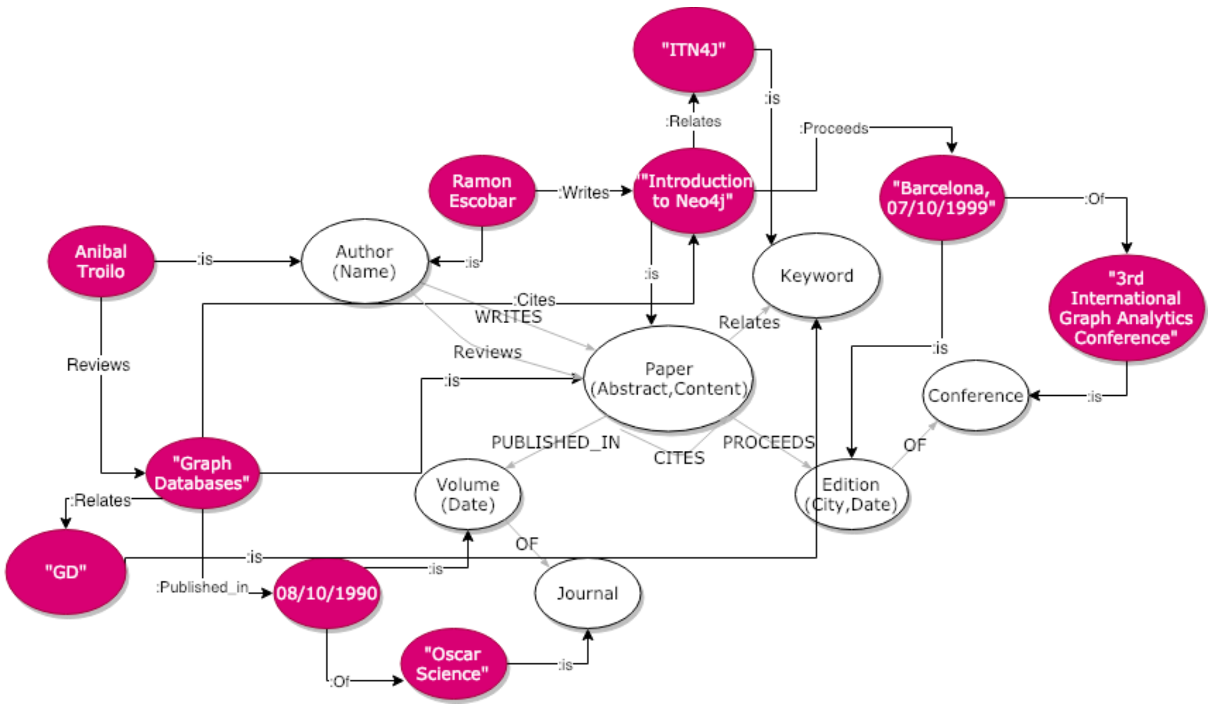
Date
14/03/2019

Table of Contents

A. Modeling, Loading, Evolving

A.1 Modeling

This is the graph that we created, it contains the metadata of the graph that we will import in part A.2:



For these queries, we have restricted ourselves to add any attributes to the edges, this is because they are less efficient than creating nodes. For example, instead of storing the volume of the edition of a conference in the proceeds relationship between a paper and a conference, we have created a distinct node. This decision is due to two reasons, first, we will need to query editions for the impact factor query, and second, if we later need to store data about the edition (such as the city) we would need to do this for every edge, resulting in data redundancy and maintainability costs.

Another decision taken is including keywords as nodes instead of attributes of the papers. This seems less efficient because we need an extra step when querying the papers. However, it is very likely that we want to query what papers contain a certain keyword in the future, and thus its separation in a different node.

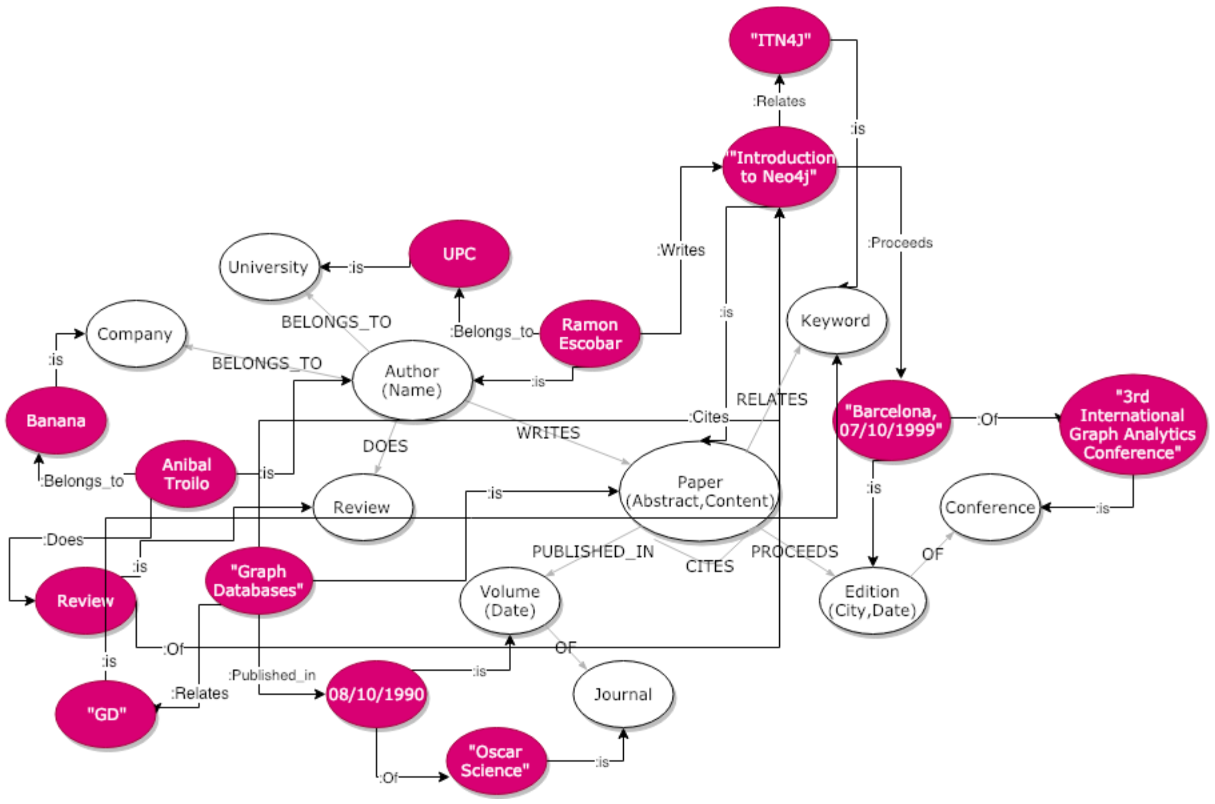
A.2 Instantiating/Loading

In order to create the data, we have made an script that generates 1000 papers, 100 authors, 10 conferences and 10 journals. The links between these data are created randomly in most cases, or following a simple rule in other cases. For example, each author takes 10 papers based on their creation order.

More details can be found in the comments of the script file (PartA.2_LopezCengiz.cyp).

A.3 Evolving the graph

For this problem we have to add three new types of nodes to the graph. The script PartA.3_LopezCengiz.cyp contains the data for the transformation into new nodes. The new data is once again generated randomly. The resulting new graph can be seen on this image:



As we can see, both universities and companies are stored as nodes instead of attributes, the reason for this is the fact that we will probably want to do searches by university or company in the future. Also, reviews have now their own node in order to prevent us to add attributes to the old ‘reviews’ edge.

B. Querying

1. Find the h-indexes of the authors in your graph.

```
MATCH (a:author)-[:writes]->(p:paper)<-[:cites]-
      (paper_that_cites:paper)
WITH a, p, count(paper_that_cites) as number_of_cites
ORDER BY a, p, number_of_cites
WITH a, collect(number_of_cites) as list_of_cites
WITH a, [i in range(0, size(list_of_cites) - 1) |
      i + 1 <= list_of_cites[i]] as cites_boolean
RETURN a, size(filter(b in cites_boolean WHERE b));
```

2. Find the top 3 most cited papers of each conference.

```
MATCH (citator:paper) -[:cites]->(cited:paper)
      -[:published_in]->(:edition)-[:of]->(conf:conference)
WITH conf, cited, count(citator) as cites
ORDER BY cites DESC
WITH conf, collect(cited) as cited_papers
RETURN conf, cited_papers[..3];
```

3. For each conference find its community: i.e., those authors that have published papers on that conference in, at least, 4 different editions.

```
MATCH (au:author)-[:writes]->(pa:paper)-[e1:published_in]->
      (ed:edition)-[:of]->(co:conference)
WITH au, collect(ed) as editions, co
WHERE size(editions) >= 4
RETURN au, co;
```

4. Find the impact factors of the journals in your graph

```
MATCH (p:paper {year:2019})
WITH collect(p) as current_papers
MATCH (p:paper) -[:published_in]-> (:volume) -[:of]->
      (j:journal)
WHERE p.year=2017 or p.year=2018
```

```
MATCH (citer:paper) -[:cites]-> (p)
WHERE citer in current_papers
WITH count(citer) as n_cites, p, j
RETURN j, sum(n_cites) / count(p);
```

C. Graph algorithms

Two algorithms which are called as “PageRank” and “Betweenness Centrality” were used so as to analyze our graph. The first algorithm that was implemented is PageRank, is an algorithm that measures the transitive influence or connectivity of nodes. In our study, PageRank was computed with the node **paper** and the relation were taken into as **cites**. This is the query:

```
CALL algo.pageRank.stream('paper', 'cites', {iterations:20,
     dampingFactor:0.85})
YIELD nodeId, score
MATCH (p:paper) WHERE id(p) = nodeId
RETURN p.title AS title, score
ORDER BY score DESC;
```

The reason why PageRank Algorithm was used for our graph is to find most valuable paper which has most citations. Furthermore, outdated and trending subjects can be detected with this algorithm. As a result, PageRank algorithm were used to measure the importance of the papers with respect to citations.

The second algorithm that was implemented is Betweenness Centrality. This is an algorithm that detects the amount of influence a node has over the flow of information in the graph. In our study, Betweenness Centrality was computed with the **keyword** nodes taking into account the **related_to** relationship. This is the query:

```
CALL algo.betweenness.stream(NULL, 'related_to',
     {direction:'both'})
YIELD nodeId, centrality
MATCH (k:keyword) WHERE id(k) = nodeId
RETURN k.name AS k, centrality
ORDER BY centrality DESC;
```

The reason why Betweenness Centrality Algorithm was to check what keywords were the most used in our articles, and therefore what topics seem to be more important.

D. Recommender

The scripts are stored in the ‘PartD_LopezCengiz.cyp’ file. We add data to the database in the following way:

- First, we add the node (c:community {name:'database'}). This node is linked to the seven keywords through the [:composes] edge.
- Second, we find the papers/journals that are in this community and link them to the community through a [:related_to] link.
- Third, we find the top 100 papers through page rank and link them to the community through the [:is_top_100_of] link.
- Finally, we link the authors of the papers as potential reviewers and/or gurus of the community.