OD First LAB Assignment

Marcel Porta Vallés, Mehmet Fatih Cagil

March 2019

Abstract

This document explains how has been developed a property graph to store information related with scientific papers. All the code are available at `https://github.com/Marcelpv96/PapersDB`

A Modeling, Loading, Evolving

A.1 Modeling

Visual representation of the graph

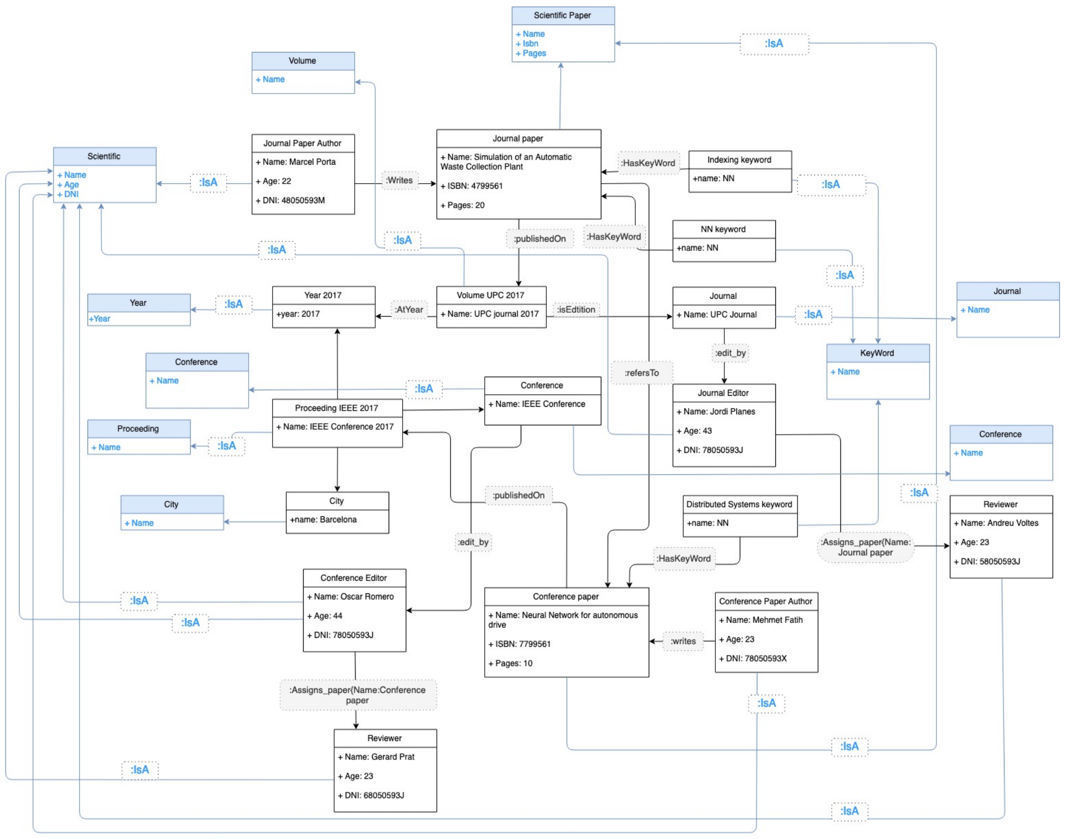

Figure 1: Visual representation of the PapersDB Graph

In summary in this section we are asked to generate a property graph which will store some information, related with scientific papers, This information is: Store Scientific papers that are written by one or several authors, that can be published in a journal volume or in a proceeding of a conference/workshop, this papers can have references to others, finally the editor of the conference or journal will assign a number of scientific to review this paper, and they will accept/denied the paper, in this part it will be important to be sure that the scientific that plays the role of reviewer is different from the author of the paper.

The property graph developed is the figure 1, and the most relevant design decisions that we have taken are:

- **Year** node: Maybe year could be an attribute of proceedings and journal volumes, but we think that will be more interesting generate a node for each year, because it will be an easy way to obtain all the proceedings or volumes published at this year.
- Paper author: We will consider at the same level all the authors, a scientific paper can be written by multiple authors.
- **City** node: Like year, we will create a node for cities, due to the same reason.
- **KeyWord**: We have chosen that for each keyword will be a node, because it will represent an easy way to obtain all the papers related with this keyword.
- **Assigns_paper** Edge, between conference editor or journal editor and reviewer, it will contain as an attribute the information related of which paper is assigned, because we think that if the name of the paper appears on the edge is enough, and it will not be necessary to put a link between reviewer and paper.

A.2 Instantiating/Loading

The solution of this section is at *papers_graph.cypher* file. The source data is generated with a generator that we have coded, and it is available at: `https://github.com/Marcelpv96/PapersDB/blob/master/source_data/Data%20generator.ipynb`

A.3 Evolving the graph

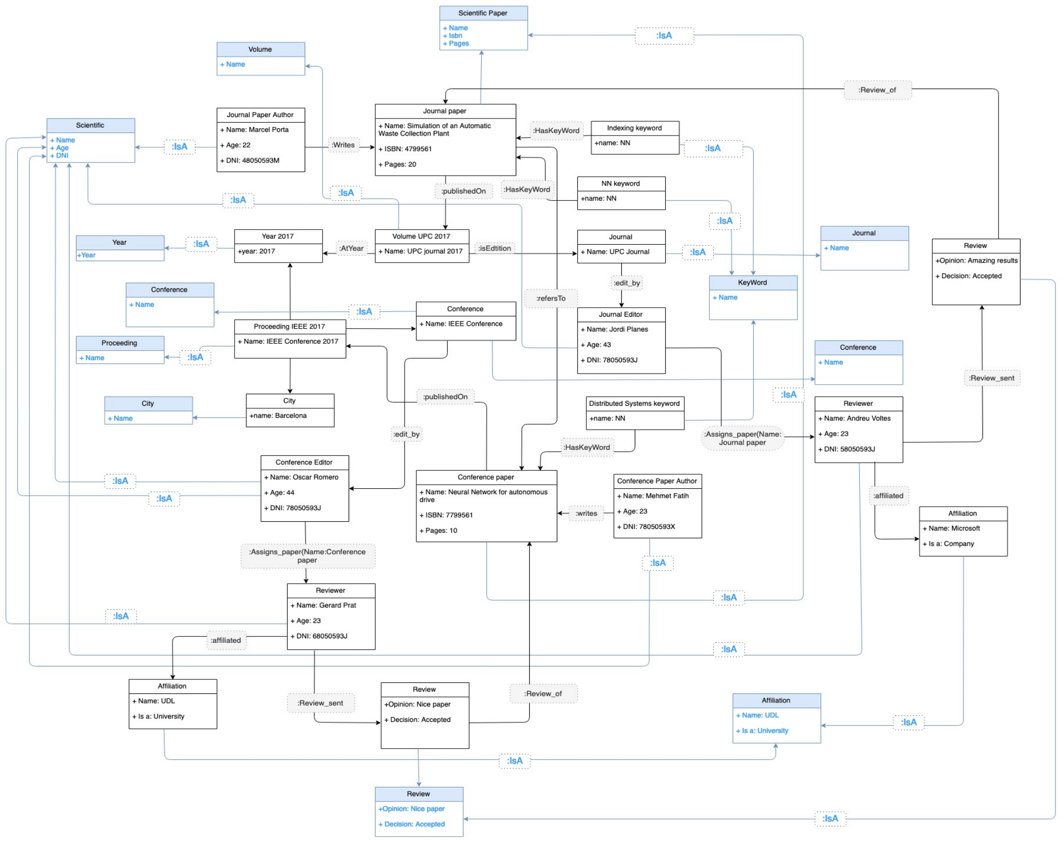

Figure 2: Visual representation of the evolution of PapersDB Graph

Now, we have evolved the graph adding more information related with the review of papers and information about reviewers. We are asked to store the affiliation of reviewer that can be two types, company or university, then store the review that he/she send, this review will have a textual description and a sugged decision.

To deal with this task, we have added new nodes and edges, the more relevant design decisions are:

- **Review** node: We have chosen to generate a node with the attributes *Opinion* that will be the text description and *Decision* that will contains if the paper is accepted or denied. Finally, we will create an edge between Review and Paper, named *Review_of* without any attribute. We think that will be useful to store all the nodes of different reviews and link them to the corresponding paper, it will create an easy way to access fast to all the reviews of a paper, for example to measure the number of acceptance that it has from the different reviewers.
- **Afifiliation** node: We will store as a node the affiliation of each reviewer, then link it to reviewer with an edge named *affiliated*. In this node as, an attribute will appear if it is Company or University. We think that it is a good approach, because if there is different reviewers from some affiliation you can go to their reviews easily.

B Querying

Find the h-indexes of the authors in your graph

```
MATCH(person:Scientific)-[:writes]->(paper:Scientific_Paper)-[:refersTo]->
(citations:Scientific_Paper)
WITH person, paper, count(citations) AS citations_no
ORDER BY person, citations_no DESC
WITH person, collect(citations_no) as citations_list
WITH person,  citations_list, range(0, size(citations_list)-1) AS index
UNWIND index AS i
WITH person, citations_list, index, collect(citations_list[i]-(i+1)) as rest
WITH person, citations_list, [x IN range(0, size(citations_list)-1)
WHERE rest[x] < 0 ] AS result
RETURN person, citations_list, CASE WHEN size(result) = 0
THEN size(citations_list) ELSE head(result) END as h_index
ORDER BY h_index DESC
```

Find the top 3 most cited papers of each conference.

```
MATCH (citing_paper:Scientific_Paper)-[:refersTo]->
(cited_paper:Scientific_Paper)-[:publishedON]->(proceeding:Proceeding)-
[:isEdition]->(conference:Conference)
WITH conference, cited_paper, count(citing_paper) as references
ORDER BY references DESC
WITH conference, collect(cited_paper) as cited_papers
RETURN conference, cited_papers[..3]
```

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

```
MATCH (person:Scientific)-[:writes]->(paper:Scientific_Paper)-[:publishedON]->
(edition:Proceeding)-[:isEdition]->(conference:Conference)
WITH person, collect(edition) as editions, conference
WHERE size(editions) >= 4
RETURN person,conference
```

Find the impact factors of the journals in your graph

```
MATCH (paper:Scientific_Paper)-[:publishedON]->(edition:Volume)-[:AtYear]->
(year:Year)
WHERE year.year = "2019"
WITH collect(paper) as papers_2019
MATCH (paper:Scientific_Paper)-[:publishedON]->(edition:Volume)-[:AtYear]->
(year:Year)
WHERE year.year = "2018" OR year.year = "2017"
MATCH (cited_paper:Scientific_Paper)-[:refersTo]->(paper)
```

```
WHERE cited_paper IN papers_2019
MATCH (edition)-[:isEdition]->(journal:Journal)
WITH count(cited_paper) as no_citings, paper, journal
RETURN journal, tofloat(sum(no_citings))/tofloat(count(paper))
```

C Graph algorithms

C.1 Betweenness Centrality Algorithm

First algorithm that we used is Betweenness Centrality Algorithm. It is a way of detecting the influence level of a node over the flow of the graph. In this case, node ***Scientific Paper*** and relationship ***refersTo*** are used in the algorithm.

Usage of this algorithm in respective node and relationship let us to find *Scientific Paper* nodes with highest amount of influence over the *refersTo* relationship. As the result, we will discover the papers which contributed most in the referencing relationship. It is possible to find out how comprehensive they are in certain communities or subjects.

```
CALL algo.betweenness.stream('Scientific_Paper','refersTo',
{direction:'in'})
YIELD nodeId, centrality
MATCH (key:Scientific_Paper) WHERE id(key) = nodeId
RETURN key.name AS keyword, centrality
ORDER BY centrality DESC;
```

C.2 PageRank Algorithm

Second algorithm that we used is PageRank Algorithm. The algorithm is used for measuring the transitive influence or node connectivity. In this case, node ***KeyWord*** and relationship ***hasKeyWord*** are used in the algorithm.

This algorithm help us to find out most hot topics in the academic fields. As the result, we are obtaining the keywords which have most relationships between papers. In this graph, amount of relationship for a keyword explains its popularity. Keywords help us to find what kind of research topics are trending among academics.

```
CALL algo.pageRank.stream(NULL, 'HasKeyWord',
{iterations:20, dampingFactor:0.85})
YIELD nodeId, score
MATCH (key:KeyWord)
WHERE id(key) = nodeId
RETURN key.name as Keyword, score
ORDER BY score DESC
```

D Recommender

D.1 Defining the research community

We are creating a *Community* node called *DBcommunity* and relating required keywords with community.

```
MERGE(DBcommunity:Community{Community_name:"DB community"})
WITH DBcommunity
MATCH (kw:KeyWord)
WHERE kw.name = "data management" OR kw.name  = "indexing"
```

```
OR kw.name = "data modeling" OR
kw.name = "big data" OR kw.name = "data processing" OR
kw.name = "data storage" OR kw.name = "data querying"
MERGE (DBcommunity)-[:HasKeyWord]->(kw)
```

D.2 Conference-Journal match to the community

Instead of connecting the conference-journal directly to the community, we are linking the proceedings-volumes to the community. This design decision was taken with respect to data we used.

```
MATCH (all_papers:Scientific_Paper)-[:publishedON]->(type)
WHERE type:Volume OR type:Proceeding
MATCH (kw:KeyWord)<-[:HasKeyWord]-(kw_paper:Scientific_Paper)
-[:publishedON]->(type)
WHERE type:Volume OR type:Proceeding AND
kw.name = "data management" OR kw.name  = "indexing" OR
kw.name = "data modeling" OR kw.name = "big data"
OR kw.name = "data processing" OR kw.name = "data storage"
OR kw.name = "data.querying"
WITH type, COUNT(distinct all_papers) as all_number,
COUNT(distinct kw_paper) as kw_papers_number
MATCH (community:Community)
WHERE tofloat(kw_papers_number)/tofloat(all_number) >= 0.9
AND community.Community_name = "DB community"
MERGE (type)-[:relatedTo]->(community)
```

D.3 Identifying the top papers

Using the PageRank algorithm, we did identify the top 100 papers and linked them with the community.

```
CALL algo.pageRank.stream(
    'MATCH (c:Community{Community_name:"DB community"})
    <-[:relatedTo]-(type)
    <-[:publishedON]-(citing_paper)
        WITH citing_paper as citer
        MATCH (c:Community{Community_name:"DB community"})
        <-[:relatedTo]-
        (type)<-[:publishedON]-(paper)
        <-[relation:refersTo]-(citer)
        RETURN distinct paper',
    'refersTo', {iterations:20, dampingFactor:0.85})
YIELD nodeId, score
MATCH (p:Scientific_Paper)
WHERE id(p) = nodeId
WITH p, score LIMIT 100
MATCH (c:Community{Community_name:"DB community"})
RETURN p.name, score
ORDER by score DESC
```

D.4 Identifying the top authors

Authors of top 100 papers are selected as good reviewers. If the author has two papers in the top 100 list, that author marked as guru.

```
MATCH (person:Scientific)-[:writes]->(paper)-[:inTop100]
->(c:Community{Community_name:"DB community"})
WITH person, c, count(paper) as papers
MERGE (person)-[:goodReviewer]->(c)
WITH person, c, papers
WHERE papers >= 2
MERGE (person)-[:isGuru]->(c)
```