Lab Assignment 1: Property Graphs

Presented by:
Pablo Jose Lopez Estigarribia
Gabriela Martínez

Presented to:
Ph.D. Óscar Romero

Universitat Politècnica de Catalunya
Facultat d'Informàtica de Barcelona
March 13th, 2018

Section A: Modeling, Loading, Evolving

Section A.1: Modeling

Visual representation of the graph

According to the information provided by the lab assignment and our common sense regarding both the conceptual model and performance of the queries that will be shown in a later stage, the following graph model is proposed:

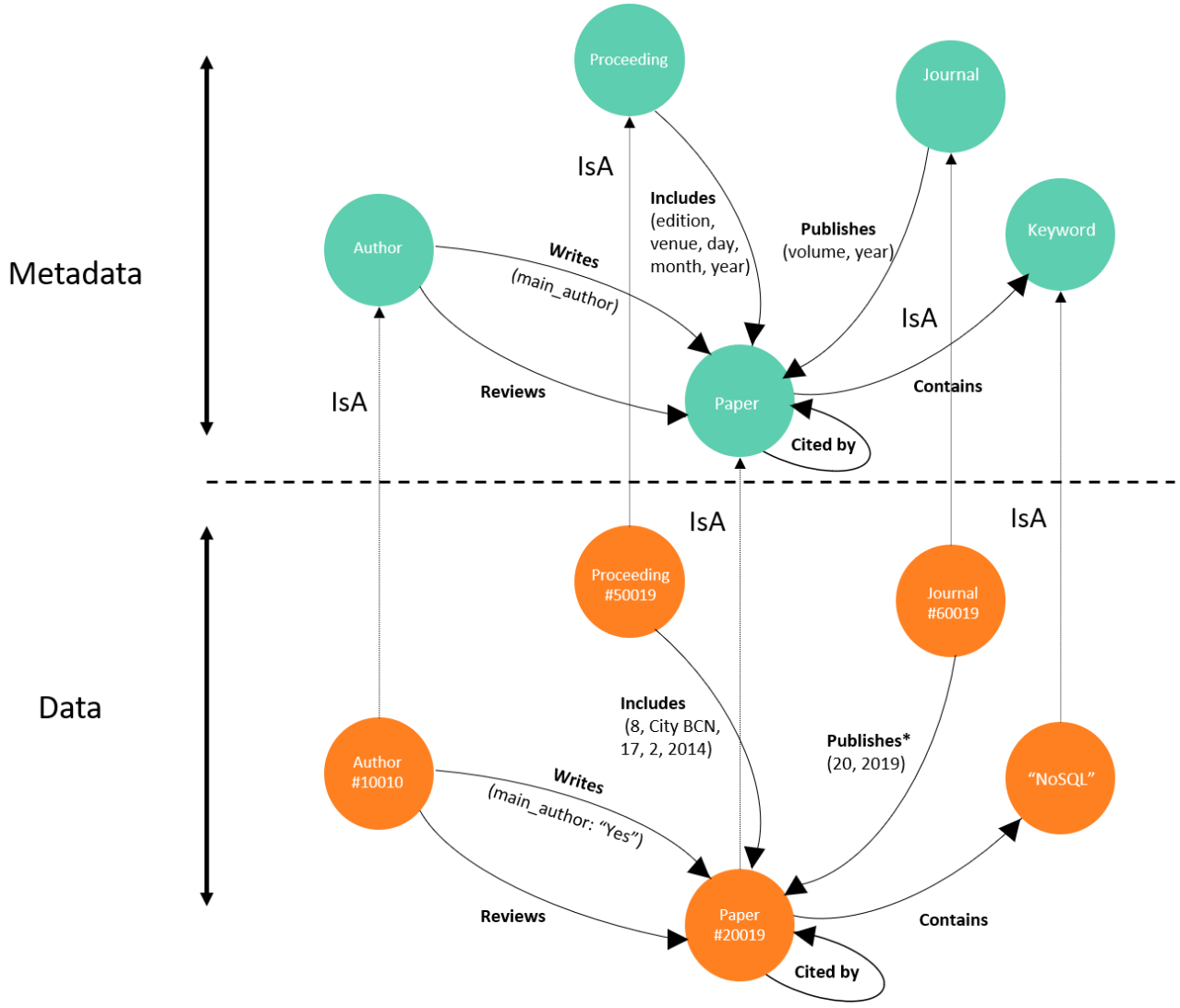

Figure 1. Conceptual graph modeling proposed for modeling the paper authorship.

*Note that for the instance shown as data, the relationship "publishes" would not exist, as one paper cannot be presented both in a proceeding and in a journal. We show the information of this relationship and its attributes only for illustration purposes.

Nodes attributes

The nodes considered in our conceptual model have the following attributes:

Author

- authorID
- authorName

Proceeding

- proceedingID
- proceedingName
- totalPapers (number of papers a proceeding has published)

Journal

- journalID
- journalName
- totalPapers (number of papers a journal has published)

Keyword

- keywordID
- keyword

Paper

- paperID
- paperTitle
- citations (number of times a paper has been cited by other)

Up to this point, the neo4j graph schema looks like the following:

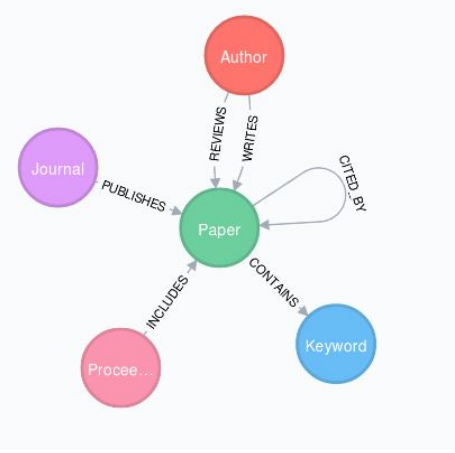

Figure 2. Neo4j schema for the graph modeled for section A1.

Conceptual modeling decisions

To justify our modeling, we took into consideration the following conceptual aspects:

- *Author*, *Paper* and *Keyword* nodes are fundamental to comply with a logic conceptual model that describes how papers are produced and to understand what type of content they offer.

- Two separate nodes were created for *Proceeding* and *Journal* because even though both of them aim to communicate papers, they perform this task in different ways in reality and obey to different types of knowledge generation. This implies that their content in terms of how the papers are published is also different (e.g. while journals are organized in volumes, proceedings summarize conferences or workshops in editions). Therefore, we think it is easier to understand the conceptual model when these two nodes are separated.

- The relationship "Writes" indicates which authors have written which papers. The direction of the relationship starts from the node with fewer records, in this case, *Author,* which will benefit the traversal of this relationship when queried.

- Similarly, the relationship "*Reviews*" was created from the *Author* to the *Paper* node due to the cardinality reason we exposed before. This relationship indicates which authors were asked to review which candidate papers for publication (either in a journal or in a conference/workshop).

- Following the same reasoning, relationships "*Publishes*" and "*Includes*" start from the *Journal* and *Proceeding* nodes and go towards the *Paper* node, which is a node with more cardinality. These relationships aim to represent where the papers are communicated, either in a journal or in a proceeding. Note that from the conceptual perspective, the node *Proceeding* gathers the workshops and conferences, two concepts that are not explicitly separated for the purpose of this graph.

- Additionally, each paper has a relationship called "*Contains*" that points to the keywords that are present in the paper content. In real life, the number of keywords of a specific domain can be higher or lower than the number of papers published. Hence, it is not possible to determine which of the two nodes has fewer cardinality. This motivated us to build the relationship from the *Paper* node, as it was clearer to understand that "a paper contains a keyword" and not the other way around.

Furthermore, with regards to the queries we want to execute on top of this graph, we took into account the following details:

- In order to ease the computation of the h-index of every author in the graph and also the impact factor of the journals, we introduced a calculated field called "citations" in the *Paper* node, as it allowed to compute beforehand the number of times each paper has been cited by others and this benefits queries performance.

- The second and third query, which expects the top 3 most cited papers per conference, also motivates the reason why we kept a node storing the proceedings, which presents the papers published by conferences.

Section A.2: Instantiating/Loading

A previous step before instantiating and loading the graph database consists of the generation of data. For this purpose, we created a python script called `dataGen.py` which creates data related to all the nodes and relationships of our graph database.

The data is generated by specifying the number of records that we would like for each node type (paper, author, journal, etc). Then, the relationships are created by performing random associations between the corresponding nodes. This generation and random association are carried out taking into consideration some logic constraints. For example, an author of a paper should not be the reviewer of the same paper he/she wrote, a paper can not be cited by itself, and if a paper is already associated with a conference it can not be associated again with a different conference or journal.

The output of the `dataGen.py` script is a set of CSV files that contain the data to instantiate the nodes as well as the records indicating the existence of relationships between them. The CSV files are generated under the subdirectory `./data/ .`

In order to load the generated data from the CSV files into the graph database, we have created a script called `A1A2_graphCreation.py`. This script takes the CSV file from the subfolder `./data/` and loads it into the neo4j graph performing bulk loads with the "LOAD CSV" tool.

The scripts for generating data and perform the bulk load are available in the following GitHub repository: https://github.com/mgmartinezl/Lab1_Semantics_2019

The order of execution is:

- Use the script `dataGen.py` to generate the CSV files into the `/data/ subdirectory.`
- Use the script `A1A2_graphCreation.py` to perform the bulk load from the CSV files into the neo4j graph database.

Section A.3: Evolving the graph

The following graph represents an updated version of the first conceptual model considered in section A1 and will implement additional features:

- The relationship "*reviews*" will be altered so that the graph can store the actual review that each author sends with regards to a paper being evaluated. For this purpose, the relationship will now hold two attributes: one called "review" to store the content of the review description, and other called "decision", which stores the suggestion of the author aiming the acceptance of a paper.

- An additional node will be created, called *Organization*. This node will contain information about the organization (university or company) that is affiliating each author. Specifically, this node will contain the attributes "orgID" and "orgName".

- The relationship between organizations and authors is also created in this updated version under the name “hosts”. The direction of this relationship is pointing to the *Author* node because the node that stores the organizations has fewer cardinality.

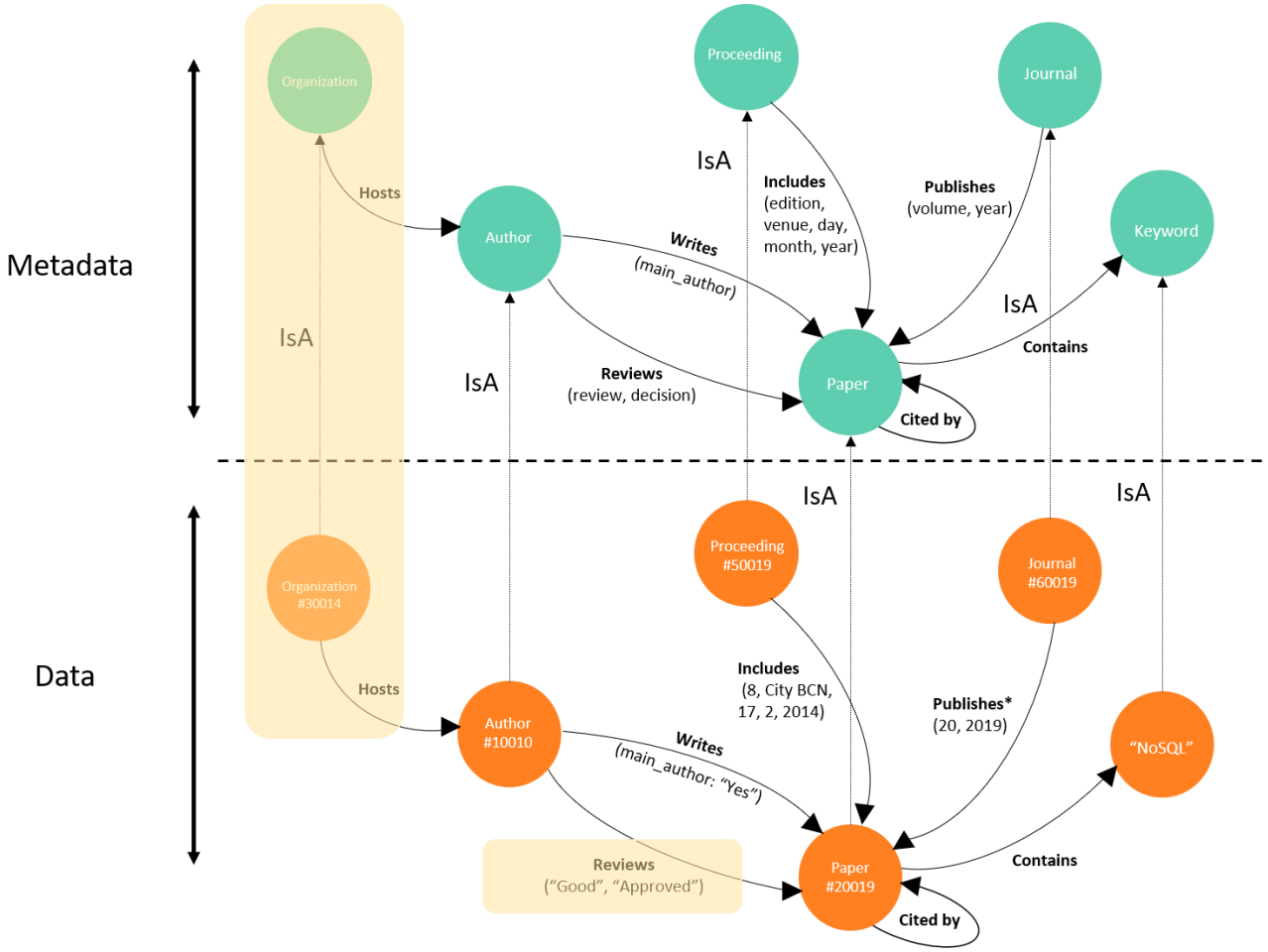

Figure 3. An updated version of the conceptual graph modeling for the lab assignment.

We have created the script `A3_evolveGraph.py` to create the additional nodes/relationships described above and also to instantiate them. This script can also be found in the same repository as the initial script https://github.com/mgmartinezl/Lab1_Semantics_2019.

After this change, our neo4j graph schema looks like this:

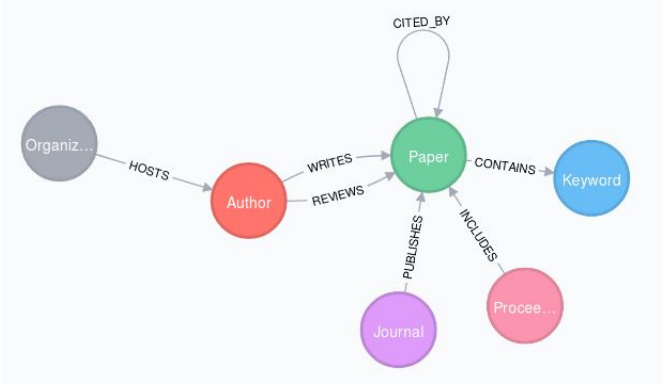

Figure 4. Neo4j schema for the graph modeled for section A3.

Section B: Querying

The following section will present some queries used to extract insights out of the modeled graph.

Query 1: find the h-indexes of the authors in your graph

```
MATCH (a:Author)-[w:WRITES]->(p:Paper)
WITH a, p ORDER BY p.citations DESC
WITH a, collect(p.citations) as citations
WITH a, citations, range(1,size(citations)) as numberPapers
// let us take advantage of the numberPapers list, since it already goes from 1 to N
unwind(numberPapers) as colIndex
//we are presenting each list in a different column, specifying the order in which
each element of the list will be shown
WITH a, numberPapers[colIndex-1] as numOrder , citations[colIndex-1] as citeNumber
where citeNumber >= numOrder
RETURN a.authorID, max(numOrder) as hindex ORDER BY hindex DESC, a.authorID ASC
```

Query 2: find the top 3 most cited papers of each conference

```
MATCH (p:Paper)<-[:INCLUDES]-(a:Proceeding)
WHERE (p)-[:CITED_BY]->()
WITH p, a, size((p)-[:CITED_BY]->()) as numCitations
ORDER BY numCitations desc
RETURN a.proceedingName, collect({paper:p.paperTitle, citations: numCitations})[0..3]
```

Query 3: for each conference find its community

```
MATCH (c:Proceeding)-[st:INCLUDES]->(p:Paper)<-[:WRITES]-(a:Author)
WITH c.proceedingID as conference, a.authorID as author, collect(distinct st.edition)
as editions
WHERE size(editions) >=4
RETURN conference, collect(author) ORDER BY conference ASC
```

Query 4: find the impact factors of the journals in your graph

```
//Obtains the published papers for a journal in the previous 2 years.
MATCH (j:Journal)-[pu:PUBLISHES]->(:Paper), (p:Paper)<-[pu2:PUBLISHES]-(j)
WHERE toInt(pu2.year) = toInt(pu.year) - 1  or toInt(pu2.year) = toInt(pu.year) -2
//Calculate the number of papers published for the journal in the previous 2 years,
as well as the number of citations that those papers received
WITH j.journalID as journalID, pu.year as year, count(*) as numPapers,
sum(p.citations) as sumcitations
//Returns the impact of journalID by year
RETURN journalID, year, toInt(sumcitations) / toInt(numPapers) as impact ORDER BY
journalID, year DESC
```

The definition of impact factor of a Journal in a year "y" is the sum of the citations of its papers from the previous 2 years (y-1 and y-2), divided by the total number of published articles in the journal, also in the previous 2 years.

$$\mathrm{IF}_y = \frac{\mathrm{Citations}_{y-1} + \mathrm{Citations}_{y-2}}{\mathrm{Publications}_{y-1} + \mathrm{Publications}_{y-2}}$$

Impact Factor formula. Taken from: https://en.wikipedia.org/wiki/Impact_factor

For this query, we considered that if a journal in the year "y" have had publications only the previous year (y-1), then the values for $Citations_{y-2}$ and $Publications_{y-2}$ will be equal to zero.

Section C: Graph Algorithms

This section aims to present the implementation of two graph algorithms available in neo4j. Specifically, we used the `graph-algorithms-algo-3.3.2.0.jar` file to able these algorithms, which is compatible with the neo4j database version we are using in the assigned virtual machine.

PageRank algorithm

This algorithm computes the influence of a node based on the number of links it has to other nodes in the network. Furthermore, PageRank not only considers how well connected a node is but also considers the number of links their connections have. One key feature of this algorithm is that it takes into consideration the direction and weight of the connections throughout the network. This means that links can transmit different degrees of influence in a specified direction (*Disney, 2014*).

For this particular exercise, we applied this algorithm to compute the importance or influence of each paper within the graph in terms of their citations. Thus, the pageRank procedure is called on the node *Paper* and on the relationship "CITED_BY" as follows:

```
CALL algo.pageRank.stream('Paper', 'CITED_BY', {iterations:20, dampingFactor:0.85})
YIELD nodeId, score
MATCH (p:Paper) WHERE id(p) = nodeId
RETURN p.paperTitle AS Paper, score as Score
ORDER BY score DESC
```

Note that we used the default option offered by neo4j for the *dampingFactor* parameter, which is usually set to 0.85 as we have found that *"the PageRank theory holds that an imaginary surfer who is randomly clicking on links will eventually stop clicking. The probability, at any step, that the person will continue is a damping factor d"* (*Ledford, 2008, p.13*) and many academic studies agree on assigning this probability to the factor d (*Brin & Page, 1998*). Additionally, even though the initial Google pageRank algorithm took 45 iterations to converge, for the scope of this exercise we decided to consider the default option offered by neo4j for the number of iterations, which is 20 (*Lawrence et al, 1999*). A preview of the results obtained from the previous query is the following:

Paper	Score
"Paper #21519"	3.5477900000000004
"Paper #20932"	2.527994
"Paper #20904"	2.51164
"Paper #20240"	2.499043
"Paper #20603"	2.266415
"Paper #20239"	2.2624115000000002
"Paper #20414"	2.182061000000001

Figure 5. Preview of the results gotten by the PageRank algorithm applied to the papers and citations in our database.

Per the previous results, we have obtained a descendant list of papers according to their pageRank score. This means that the paper appearing in the first position (Paper #21519) is the most influential one, as it has the greatest number of citations in the network. It seems that this paper is highly reviewed by others and plays an important role in the rest of the papers' contribution.

Triangle counting algorithm

This algorithm aims to detect communities within a network by counting the number of triangles passing through each node. A triangle includes three nodes that have a relationship (e.g. three people that know each other). Additionally, triangle counting is useful to measure the cohesiveness of detected communities and stability of graphs by the means of the computation of network indices (*Neo4j Graph Database Platform, n.d.*).

For the scope of this exercise, we will compute for each paper the number of triangles in which it participates. Every triangle that each paper forms with other two papers, means that those three papers are cited reciprocally (they know each other in terms of citations!) and this would suggest the existence of communities of papers that are related to a specific topic within the databases domain. Thus, the higher the number of triangles a paper participates in, the higher the number of topic communities of papers it is related to. Additionally, we will compute the *clustering coefficient* of each paper, which "*is a measure of the degree to which nodes in a graph tend to cluster together*" (*Neo4j Graph Database Platform, n.d.*). The higher this likelihood, the tighter the relationship that exists within each paper and the communities which it is part of (a measure of cohesiveness). The query that allowed us computing these two variables for each paper is as follows:

```
CALL algo.triangleCount.stream('Paper','CITED_BY')
YIELD nodeId, triangles, coefficient
MATCH (p:Paper) WHERE id(p) = nodeId
RETURN p.paperTitle as Paper, triangles, coefficient
ORDER BY triangles DESC
```

A preview of the results obtained from the previous query is the following:

Paper	triangles	coefficient
"Paper #21070"	7	0.03333333333333333
"Paper #20020"	6	0.031578947368421054
"Paper #20031"	6	0.03508771929824561
"Paper #20433"	5	0.023809523809523808
"Paper #20441"	5	0.02631578947368421
"Paper #20598"	5	0.047619047619047616
"Paper #20609"	5	0.032679738562091505

Figure 6. Preview of the results gotten by the Triangle Counting algorithm applied to the papers and citations in our database.

According to the previous figure, the paper with the title "Paper #21070" is the most linked one with other pairs of papers in terms of citations. This paper can be identified to be part of four different communities in which it is being cited by others that it also cites as well, probably because all those papers are extremely relevant to the topic they are presenting.

Furthermore, if we execute the same query again but order by the coefficient in a descendant way, we get these top papers:

Paper	triangles	coefficient
"Paper #20595"	2	0.13333333333333333
"Paper #20629"	1	0.06666666666666667
"Paper #20793"	1	0.06666666666666667
"Paper #21091"	1	0.06666666666666667
"Paper #21677"	1	0.06666666666666667
"Paper #21851"	1	0.06666666666666667
"Paper #20891"	2	0.05555555555555555

Figure 7. Preview of the results gotten by the Triangle Counting algorithm applied to the papers and citations in our database.

Note that the paper with the title "Paper #20595" has the first position with a coefficient of 13%, which indicates the probability of this paper to cite the papers of its communities (given by the triangles in which this paper participates in) in a consistent and continuous way. As per the number, which is low for indicating strong "community" relationships, we can infer that papers in this graph (which have even lower coefficients) simply tend to cite the papers that are relevant for the topic they are related to and there is no a specific group of papers that are prone to cite themselves very often, something expected to happen from the academic perspective. However, in a social network, for instance, it is very common to observe large clustering coefficients that reflect the deep relationships (e.g. friendship or family ties) that people involved maintain (*Neo4j Graph Database Platform*, n.d.).

Section D: Recommender

In order to build a simple recommender that allows editors and chairs to choose the right reviewers for the papers, we must conduct several queries to the graph created, as follows:

Find the research communities

The following query will create a node called *researchCommunity* within the built graph. This node will contain at first, the authors that have written papers that contain at least one of the following keywords: `'big data'`, `'data management'`, `'indexing'`, `'data modeling'`, `'data processing'`, `'data storage'`, `'data querying'.` Note that these authors will be considered if and only if they are the main author of the paper at the matter and will be connected to the research community through the "BELONGS" relationship.

```
MATCH (a:Author)-[:WRITES {main_author: 'Yes'}]->(p:Paper)-[:CONTAINS]->(k:Keyword)
WHERE k.keyword IN  ['big data', 'data management', 'indexing', 'data modeling',
'data processing', 'data storage', 'data querying']
MERGE (c:Community {name : 'researchCommunity'})
MERGE (a)-[:BELONGS]->(c)
```

Find the associated journals and conferences

The authors identified in the previous query, publish their papers in different journals or conferences. However, we only want to identify those journals and proceedings which have at least 90% of their papers related to at least one of the keywords mentioned. We will associate these journals and conferences to the research community by the means of the "JournalPartOf" and "ProceedingPartOf" relationships.

For the journals, the case is:

```
MATCH(j:Journal)-[r:PUBLISHES]->(p:Paper)-[c:CONTAINS]->(k:Keyword)
WHERE k.keyword IN ['big data', 'data management', 'indexing', 'data modeling', 'data
processing', 'data storage', 'data querying']
WITH j, count(r) as papersCommunity
WITH j, (toFloat(papersCommunity) / j.totalPapers) as calc
WHERE calc > 0.9
MERGE (c:Community {name : 'researchCommunity'})
MERGE (j)-[:JournalPartOf]->(c)
```

Similarly, the conferences are found via the proceedings node:

```
MATCH(pr:Proceeding)-[r:INCLUDES]->(p:Paper)-[c:CONTAINS]->(k:Keyword)
WHERE k.keyword IN ['big data', 'data management', 'indexing', 'data modeling', 'data
processing', 'data storage', 'data querying']
WITH pr, count(r) as papersCommunity
WITH pr, (toFloat(papersCommunity) / pr.totalPapers) as calc
```

```
WHERE calc > 0.9
MERGE (c:Community {name : 'researchCommunity'})
MERGE (pr)-[:ProceedingPartOf]->(c)
```

Find top papers of the research community

Next step to building the recommender consists of extracting the top papers of the journals and conferences (proceedings) that resulted from the previous query, in terms of the page rank considering the number of citations of each paper. The following query will compute the top 100 papers and they will be assigned to the research community by the means of the "TopPaperPartOf" relationship:

```
CALL apoc.cypher.run(
"CALL algo.pageRank.stream('Paper', 'CITED_BY', {iterations:20, dampingFactor:0.85})
YIELD nodeId, score
MATCH(p:Paper)<-[:INCLUDES]-(pr:Proceeding)-[:ProceedingPartOf]->(c:Community)
WHERE id(p) = nodeId
RETURN DISTINCT p.paperID AS Paper, score as Score
UNION
CALL algo.pageRank.stream('Paper', 'CITED_BY', {iterations:20, dampingFactor:0.85})
YIELD nodeId, score
MATCH(p:Paper)<-[:PUBLISHES]-(j:Journal)-[:JournalPartOf]->(c:Community)
WHERE id(p) = nodeId
RETURN DISTINCT p.paperID AS Paper, score as Score", NULL)
YIELD value
WITH value.Paper AS Paper, value.Score as Score
ORDER BY Score desc limit 100
WITH collect(Paper) as paperIDList
MATCH (p:Paper) where p.paperID in paperIDList
MERGE (c:Community {name : 'researchCommunity'})
MERGE (p)-[:TopPaperPartOf]->(c)
```

For the execution of the previous query, we installed the apoc library available in neo4j (`apoc-3.3.0.4-all.jar`) that allowed us called a subroutine to perform operations from a UNION operator.

Find potential reviewers and gurus

The authors of the top 100 papers computed before are considered automatically as potential reviewers for any paper related to the databases domain. Additionally, authors that at least wrote two papers in that top 100 will be considered as gurus in the field. The following query will create relationships within the research community that allow us to identify these authors.

Specifically, we want to identify potential reviewers as:

```
MATCH (a:Author)-[w:WRITES {main_author:
'Yes'}]->(p:Paper)-[:TopPaperPartOf]->(c:Community)
MERGE (a)-[:PotentialReviewerPartOf]->(c)
```

Note that these potential reviewers are linked to the *research community* by the means of the "PotentialReviewerPartOf" relationship.

Similarly, the gurus will also be part of the community by the means of the "GuruPartOf" relationship:

```
MATCH (a:Author)-[w:WRITES {main_author:'Yes'}]->(p:Paper)
-[:TopPaperPartOf]->(c:Community)
WITH a, count(w) as TopPapers, c
WHERE TopPapers >= 2
MERGE (a)-[:GuruPartOf]->(c)
```

References

Disney, A. (2014). Social network analysis: Centrality measures. Retrieved March 11, 2019, from https://cambridge-intelligence.com/keylines-faqs-social-network-analysis/

Brin, S., & Page, L. (1998). The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, *30*(1–7), 107–117. http://doi.org/10.1016/S0169-7552(98)00110-X

Neo4j Graph Database Platform. (n.d.). The PageRank algorithm. Retrieved March 11, 2019, from https://neo4j.com/docs/graph-algorithms/current/algorithms/page-rank/#algorithms-pagerank-syntax

Ledford, J. L. (2008). *SEO : search engine optimization bible*. Wiley.

Page, L., Brin, S., Motwani, R., & Winograd, T. (1999). The PageRank Citation Ranking: Bringing Order to the Web. Retrieved from http://ilpubs.stanford.edu:8090/422/