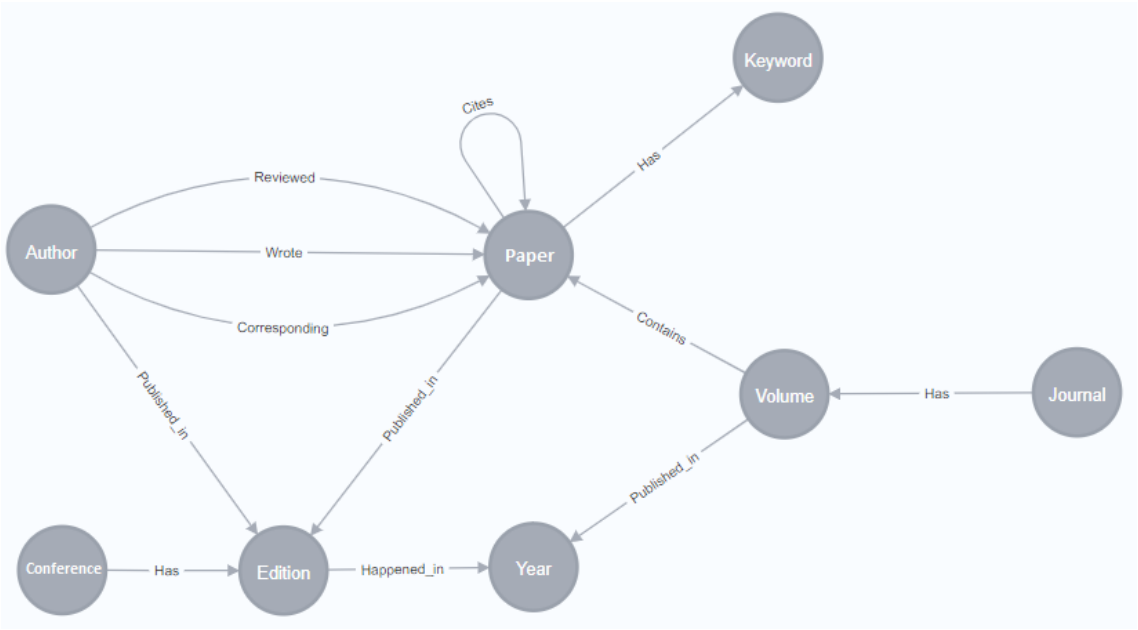
SDM: Property Graphs

Team: Diogo Repas & Luiz Alberto Fonseca

A - Modeling, Loading, Evolving

A.1 Modeling

Below, you'll find the graph schema we designed for the problem.



Design Justifications

Some nodes and edges are obvious and need no explanation. We will justify our design based on what it's useful to solve the questions in section B.

For Section B

Find the top 3 most cited papers of each conference

To answer this (efficiently), we would need a citation edge (we called it "Cites") whenever a paper cites other paper. We also need a conference/proceeding node (we called it "Conference") to summarize the information per conference / proceeding.

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

To answer this (efficiently), we decided to create an authorship node. Semantically, one conference will be linked to one or more editions and the papers are connected to the edition in which they were published. Then we decided to create an edge from Author to Edition because even though we could reach an Author from an Edition by going **(Author)-[Wrote]->(Paper)-[Published_in]->(Edition)**, if we had an edge directly connecting the author and the edition, we would avoid having to search intermediary connections to a paper.

Find the impact factors of the journals in your graph.

Source code: https://github.com/bdma2021/SDM-property-graphs

To calculate the impact factor, we need to consider the number of publications and citations in the last two years for each journal. We decided to add the node Year, because it is more efficient to filter the node Year and then get the volumes connected to that year, than to have a property year in the node Volume and search through all the volumes where YEAR = <some year>.

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

To compute the H-index, we just need the publications of an author and the number of citations of each publication. We have that information in the nodes Author and Paper and relationships Wrote and Cites.

A.2 Instantiating/Loading

Part of the data used was extracted from Aminer (www.aminer.org/data/#Citation) [DBLP-Citation-network V13: 5,354,309 papers and 48,227,950 citation relationships (2021-05-14)]. This data is collected from DBLP, ACM, and other sources. With the Aminer dataset we were able to get the data to create the following nodes and relationships:

> Nodes: Author, Paper, Keyword, Year
> Relationships: (Author)-[**Wrote**]->(Paper), (Author)-[**Reviewed**]->(Paper), (Author)-[**Corresponding**]->(Paper), (Paper)-[**Has**]->(Keyword), (Paper)-[**Cites**]->(Paper)

For the rest of the entities, fake data was generated. We created fake data for the following entities:

> Nodes: Conference, Edition, Journal, Volume
> Relationships: All relationships related to the nodes above.

Preprocessing

To insert the data in Neo4J, we used the LOAD CSV command. So, first we generated one .csv file for each node and one for each relationship in our schema. First, we inserted the nodes, then we created indexes on the nodes' ids to speed up the insertion of the relationships. After the index creation we insert the relationships from the .csv files. The code for this part is in a python script named PartA.2_FonsecaRepas.py.

Reproducing the code

To execute the code correctly you need to create a Neo4j database manually by yourself. Then you must place the .csv files in the **import** directory of your database, which is the recommended secure folder to put any data that should be imported into a Neo4j instance. After that, on line 362 in the script, you should modify the parameters **bolt-url**, **user** and **password** to comply with the settings of the Neo4j instance you just created. The whole script takes around 2 up to 5 minutes to finish.

You will find the .csv files in this GitHub repository (https://github.com/bdma2021/SDM-property-graphs) in the file data.zip.

A.3 Evolving the graph

To insert the necessary modifications, we propose to add 2 attributes to the relationship Reviewed, which are:

- Content – the content of the review
- Decision – The decision suggested by the reviewer.

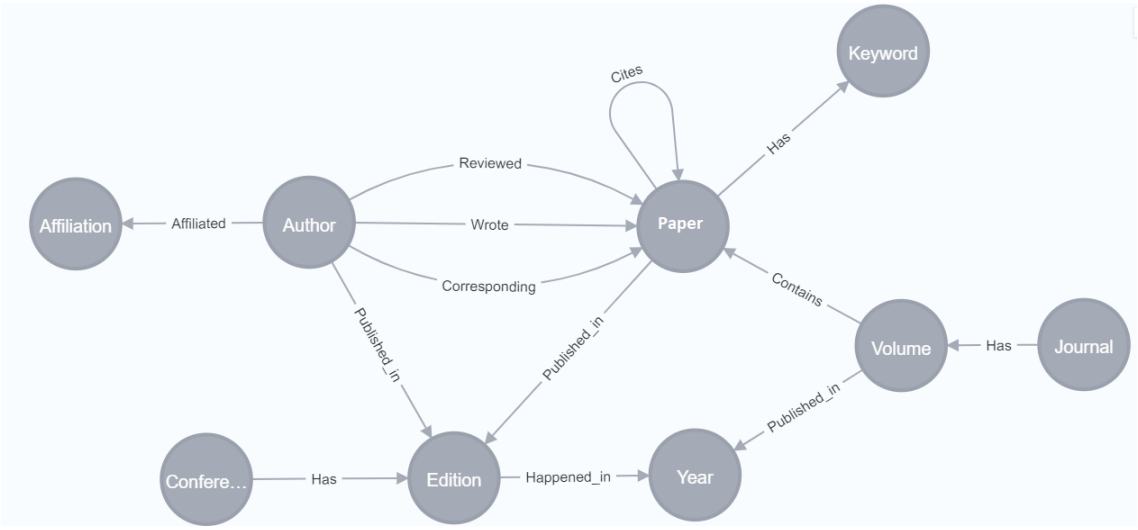
Since these attributes are not related to an Author nor to a Paper, but to a Review of an Author to a Paper, we decided that they should be better placed in the relationship itself.

We also need to insert an Author's Affiliation. To achieve that, we decided to create a node Affiliation and a relationship between the authors and their affiliations. This way we can reuse an Affiliation node when two authors have the same Affiliation. The entities created are:

- Affiliation (node) – A node to hold the information about the affiliation (a company or university)
- Affiliated (relationship between Author and Affiliation) – This relationship represents the link between an author and their affiliation.

New schema

The final schema after the modifications is depicted below.



The .csv files containing the data for the required modifications are in the same folder as the others and they're name Affiliation.csv, Affiliated.csv and Reviewed_v2.csv. You must place them in the import folder before running the script.

B – Querying

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

```
MATCH (p:Paper)-[:Published_in]->(:Edition)<-[:Has]-(conference:Conference)
MATCH (:Paper)-[c:Cites]->(p:Paper)
WITH conference, p, COUNT(c) AS citations
ORDER BY citations DESC
RETURN conference, collect(p)[..3] as topThree
```

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

```
MATCH (c:Conference)-[:Has]->(e:Edition)<-[:Published_in]-(a:Author)
WITH c AS conference, a AS author, COUNT(DISTINCT e) AS number_editions
RETURN conference, collect(CASE WHEN number_editions >= 4 THEN author END) AS co
mmunity
ORDER BY size(community) DESC
```


Source code: https://github.com/bdma2021/SDM-property-graphs

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

```
// For year = 2019
MATCH (journal:Journal)-[:Has]->(v:Volume)-[:Contains]->(p:Paper)
MATCH (v)-[:Published_in]->(publication_year:Year)
MATCH (p)-[:Cites]->(reference:Paper)<-[:Contains]-(:Volume)-[:Published_in]-
>(citation_year:Year)
WHERE citation_year.year = "2019" AND publication_year.year IN ["2018", "2017"]
RETURN journal, COUNT(DISTINCT reference) * 1.0 / COUNT(DISTINCT p) AS impact_fa
ctor
ORDER BY impact_factor DESC
```

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

```
MATCH (:Paper)-[c:Cites]->(p:Paper)<-[:Wrote]-(author:Author)
WITH author, p, COUNT(c) AS citations
ORDER BY citations DESC
WITH author, collect({paper:p, citations:citations}) AS paper_citations
UNWIND range(1,size(paper_citations)) AS i
WITH author, paper_citations[i-1].citations AS citations, i
WHERE citations > i
RETURN author, max(i) AS h_index
```

C - Graph algorithms

C.1 Node Similarity

In our context the node similarity algorithm will be used to compute the similarity between two papers based on common keywords. The implementation using the Graph Data Science Library is shown below.

Query 1 – Creating a bipartite graph.

The input of this algorithm is a bipartite, connected graph containing two disjoint node sets. In this case the node sets are for Paper and Keyword.

```
CALL gds.graph.create(
   'paper-similarity',
   ['Paper', 'Keyword'],
   {
      Has: {
         type: 'Has'
      }
   }
);
```

Query 2 – computing similarity.

This query will compute a similarity score for all pairs of papers in the graph. It's a costly algorithm, but once you have the similarities stored you can just check, for example, the papers that are more similar to a certain paper and use that result to generate recommendations of papers to read.

```
CALL gds.nodeSimilarity.stream('paper-similarity')
YIELD node1, node2, similarity
```


Source code: https://github.com/bdma2021/SDM-property-graphs

```
RETURN gds.util.asNode(node1).title AS Paper1, gds.util.asNode(node2).title AS Paper2, simil
arity
ORDER BY similarity DESCENDING, Paper1, Paper2
```

C.2 – Page Rank

Firstly, we need to create a projection of our database and add it to the graph catalog, this can be done with the following cypher query:

```
CALL gds.graph.create('citation_network ', 'Paper', 'Cites');
```

Then, PageRank can be calculated on this projection with the following cypher query:

```
CALL gds.pageRank.stream("citation_network")
YIELD nodeId, score
RETURN gds.util.asNode(nodeId) AS paper, score
ORDER BY score DESC;
```

This query returns all papers sorted by their corresponding PageRank scores. A PageRank score can be interpreted as the relative importance of the paper, if we assume that a paper is as relevant as the papers that cite it.

D – Recommender

Part 1

For part 1, we first create a node labeled Community with name = 'database'.

```
CREATE (:Community {name: 'database'});
```

Then we find all the keywords in the database community and create a relationship (:Contains) from the community node to those keywords. We slightly changed some keywords, for example, instead of "indexing" we put "database index", etc.

```
MATCH (keyword:Keyword)
WHERE toLower(keyword.keyword) IN ['data management', 'database index', 'data modeling', '
big data', 'data processing', 'data store', 'database querying']
MATCH (community:Community {name: 'database'})
CREATE (community)-[:Contains]->(keyword);
```

Part 2

For part 2, we get all the journals and all the conferences that are above the threshold. We changed the threshold of 90% because none of our journals/conference complied with this value, but the query would not change except for the value of the threshold. We have one query for journals and another one for conferences for purposes of performance. At the end, we create a relationship (Relates) from the journal/conference to the community node.

```
MATCH (journal:Journal)-[:Has]->(:Volume)-[:Contains]->(paper:Paper)
WITH journal, COUNT (paper) AS total_papers
MATCH (journal:Journal)-[:Has]->(:Volume)-[:Contains]->(community_paper:Paper)-[:Has]-
>(:Keyword)<-[:Contains]-(:Community {name: 'database'})
WITH journal, COUNT(community_paper) AS total_com_papers, total_papers
WITH journal, total_com_papers * 1.0 / total_papers AS community_percentual
WHERE community_percentual >= 0.03
```


Source code: https://github.com/bdma2021/SDM-property-graphs

```
MATCH (community:Community {name: 'database'})
CREATE (journal)-[:Relates]->(community);


MATCH (conference:Conference)-[:Has]->(:Edition)<-[:Published_in]-(paper:Paper)
WITH conference, COUNT (paper) AS total_papers
MATCH (conference)-[:Has]->(:Edition)<-[:Published_in]-(community_paper:Paper)-[:Has]-
>(:Keyword)<-[:Contains]-(:Community {name: 'database'})
WITH conference, COUNT(community_paper) AS total_com_papers, total_papers
WITH conference, total_com_papers * 1.0 / total_papers AS community_percentual
WHERE community_percentual >= 0.03
MATCH (community:Community {name: 'database'})
CREATE (conference)-[:Relates]->(community);
```

Part 3

For part 3, we start by creating the database intra-community citation network graph:

```
CALL gds.graph.create.cypher(
        "database_community",
        "MATCH (n:Paper)-[r]-()<--()-[:Relates]->(:Community { name: \"database\"})
        WHERE r:Contains OR r:Published_in RETURN DISTINCT id(n) AS id",
        "MATCH (n)-[r:Cites]->(p) RETURN id(n) AS source, id(p) AS target",
        { validateRelationships: false }
);
```

The top 100 pages of this community can be calculated using the PageRank algorithm over the previously created intra-community citation network. We decided to highlight them using the label “Top100DatabaseCommunity” for reference.

```
CALL gds.pageRank.stream("database_community")
YIELD nodeId, score
WITH gds.util.asNode(nodeId) AS paper
ORDER BY score DESC
LIMIT 100
SET paper :Top100DatabaseCommunity
RETURN paper;
```

Part 4

To get potential reviewers the following cypher query can be used:

```
MATCH (potential_reviewer:Author)-[:Wrote]->(n:Top100DatabaseCommunity)
RETURN DISTINCT potential_reviewer;
```

To identify gurus in the database community the following cypher query can be used:

```
MATCH (guru:Author)-[:Wrote]->(p1:Top100DatabaseCommunity)
MATCH (guru:Author)-[:Wrote]->(p2:Top100DatabaseCommunity)
WHERE p1 <> p2
RETURN DISTINCT guru;
```


Source code: https://github.com/bdma2021/SDM-property-graphs