Lab Document I (Semantic Data Management)

Thomas Schmidt, Eemi Vihavainen

A Modeling, Loading, Evolving

A 1 - Modeling

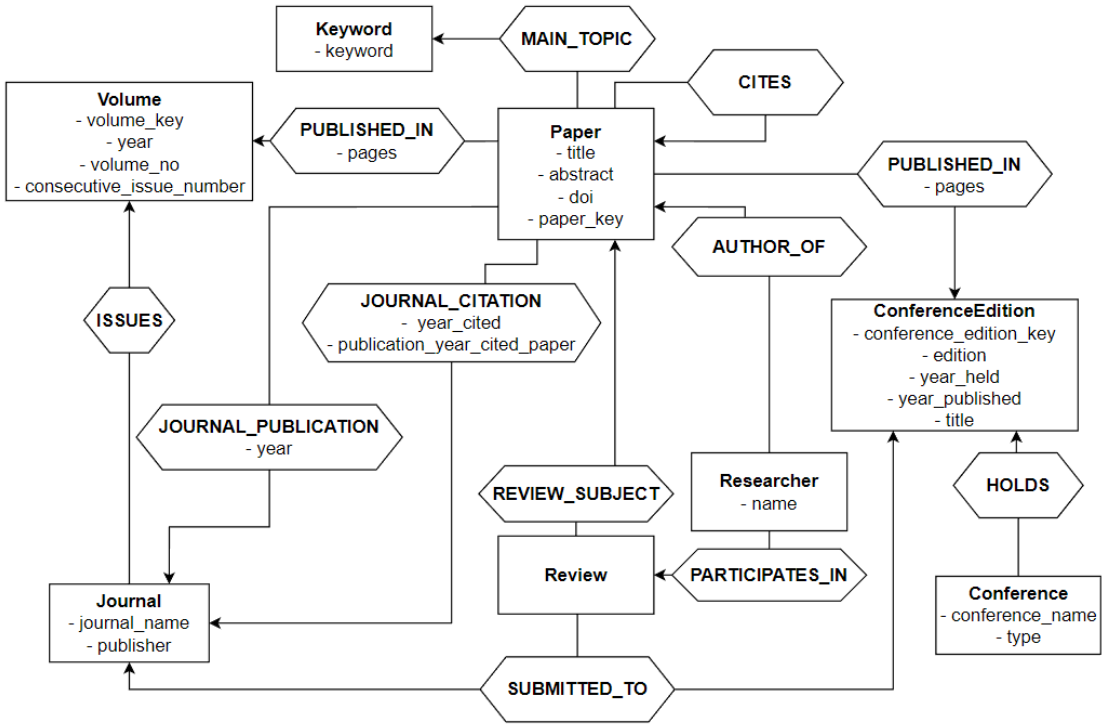

Figure 1: Schematic representation of the graph (Rectangles: nodes, Hexagon(+Line): Relationships, Bold Font: Node/Relationship Label, Normal Font: properties expected on the instances).

Volume:volume_key	key identifying the volume for loading. Consists of journal name, year and volume no	*ConferenceEdition:title*	title of the conference, based on input data often includes the location
Published_in:pages	page number of the paper in the volume or proceeding	*ConferenceEdition:year_published*	year the edition was published
Volume:volume_no	number of the volume in the year	*Conference:conference_name*	name of the conference
Volume:consecutive_issue_number	issue number since inception of the journal	*Journal:journal_name*	name of the journal
Keyword:keyword	keyword string	*Conference:type*	Workshop or Conference
Paper:title	title of the paper	*Journal:publisher*	publisher of the journal
Paper:abstract	abstract of the paper	*Researcher:name*	name of the researcher

Paper:doi	doi of the paper	*Volume:year*	year of the volume
Paper:paper_key	key of the paper from the DBLP dataset	*Journal_Publication:year*	Year the paper was published in the journal
ConferenceEdition:edition	edition number of the conference	*Journal_Citation:year_cited*	year the paper cited a paper from that journal
ConferenceEdition:conference_edition_key	key of the conference edition / proceeding from the DBLP dataset	*Journal_Citation:publication_year_cited_paper*	year the paper that is cited by this paper was published
ConferenceEdition:year_held	year the edition was held		

Table 1: Attribute Specification

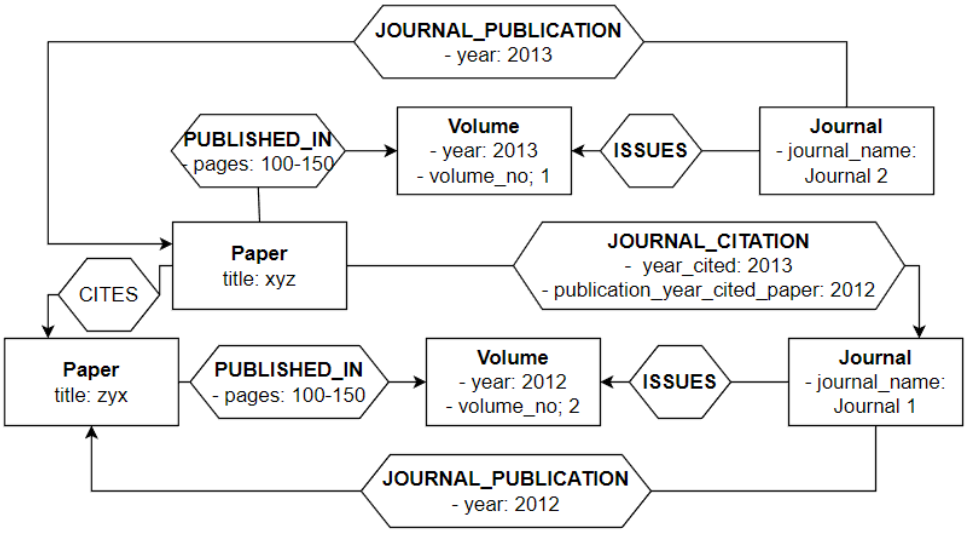

Figure 2: Example of instantiated JOURNAL_CITATION / JOURNAL_PUBLICATION relationships (some properties have been omitted, please refer to Figure 1 for a complete set).

While property graphs formally do not have a schema, it is useful to clearly state the interplay between nodes and relationships with different labels. As such, as a first step we performed some schematic modeling that our graph should follow for this Lab assignment. A visual representation of the “schema” can be found in **FIGURE 1**. We choose this abstracted representation over a representation of instances due to the fact that for the latter the multitude of different node and relationship types would make an instance representation overwhelming. Where appropriate we show excerpts of the instantiated graph (then, some attributes might be omitted to provide a clearer picture. All properties can be found in the schematic representation).
We identified several Concepts that we wanted to represent as nodes: *Paper, Keyword, Review, Researcher, Volume, Journal, ConferenceEdition, and Conference.*

- Paper is the central concept of a graph about publications and is thus modeled as node. A *Paper* can be related to another *Paper* via the CITES relationship, which indicates that one paper is citing the other. While in the schematic representation the CITES relationship points to *Paper* again a paper can obviously not cite itself and the relationship has to refer to another instance (Please refer to **Figure 2** for an example). Since citations have a central role in the scientific community and can be interpreted in different ways

(citing a paper, author or journal), we were also considering modeling it as node. However, this usually would not have had a positive effect on query performance because queries involving citations would need more hops. Instead, for improving the performance of the impact factor-query we introduced two additional relationships from the *Paper* to the *Journal*, as described in the following paragraph.

- To speed up query performance when computing the impact factor, two additional relationships are added (please refer to **Figure 2** for an instantiated representation). JOURNAL_PUBLICATION captures the notion of a paper being published in a journal. It allows you to calculate the number of publications for a given year in one hop, instead of two. JOURNAL_CITATION captures the notion that a paper is citing a paper from that journal. It has the relevant properties to get the right citation count for the numerator of the impact factor in one hop (year of citation, and year the cited paper has been published). The relationship points from the citing paper to the relevant journal.
  While both introduce some data redundancy, the performance gain justifies the creation of these relationships. Also the property values on the relationships don't have to be changed after the relationship has been created because the values are static.
- *Keywords* might be used in several different papers and act as grouping nodes of similar papers, and are therefore modeled as a separate node rather than as property on paper. This also improves query performance for queries based on the keyword, since modeling the keywords as list property would require looping through each keyword list to find relevant papers. Modeling them as node makes such queries a simple lookup.
- *Review* is modeled as a node. This is because it has several relationships to multiple different node types (*Researcher*, *Paper*, and *Journal*/*ConferenceEditon* the paper is submitted to). This complexity can best be represented and collected by modeling it as a node. Researchers are related to Review via the PARTICIPATES_IN relationship, indicating that they take part in the review. The Review is related to the respective paper via the REVIEW_SUBJECT relationship. Finally, the SUBMITTED_TO relationship indicates whether the Paper is submitted to a Journal or a ConferenceEdition. While the schematic representation (Figure 1) shows two outgoing SUBMITTED_TO relationships, obviously one Review instance can only have one such relationship to either a journal or a conference edition. While in the schematic representation there is the relation AUTHOR_OF and PARTICPATES_IN going out of the Researcher node, in the generated data we made sure that an author cannot review her own paper.

A 2 - Instantiating/Loading

The DBLP dataset is used as basis for the lab assignment. To be able to comply with the tasks, some additional data is generated.

- 10k articles and inproceedings are loaded and subsequently filtered to records containing complete information (year, journal name, pages, authors,...). Only publications from 1986 onwards are included. A random publisher is generated (Publisher {1,25}).
- To map articles to volumes and volumes to the issuing journal, a volume key is introduced that comprises the journal name, the year published, and the volume number of that year.
- Keywords are created by tokenizing the title and filtering out stop words. This only leads to one-word keywords. For part C thus, several Journals and Conferences were selected that should later be contained in the database community. For those, database community keywords were added (which are usually 2 words long).

- After scanning some titles of proceedings it became clear that the location information is included in the title string. As described also in the previous part, we thus did not create random locations, because they are not relevant in the remainder of the lab exercise and just kept the location information as part of the title.

→ ***Created files:*** **{articles, inproceedings, proceedings}_preprocessed.csv**

Citations are generated by considering that a paper can only cite a paper that is *published before* itself. Between 5 and 15 citations are created for every paper. To ensure that there are enough citations for journals to compute the impact factor, another 5 to 15 citations citing specifically articles are generated for each paper.

→ ***Created files:*** **citations.csv**

Editor and chairperson relationships were created by considering the 3 most published authors for that conference or journal. In the cases this didn't yield any data a random author was assigned as editor/chaiperson.

→ ***Created files:*** **editors.csv, chairpersons.csv**

Review information is generated by assigning a journal a number of reviewers. Usually these are three but we included some variation. Reviewers are selected by taking random researchers that have published in that journal or conference. *Then it is ensured that the author is not included in the reviewers.* Where it was not possible to sample affiliated reviewers, a random researcher was selected. Also a review text (lorem ipsum), and a suggested decision is added. The suggested decision evolves around standards in the scientific community and can be {"acceptance", "conditional acceptance", "conditional rejection", "outright rejection"}, where the first two are considered as supporting the publication.

→ ***Created files:*** **reviewers.csv**

An affiliation is generated for loading after evolving the graph to include such information.

→ ***Created files:*** **affiliations.csv**

Other assumptions for this part before running the import queries:

1. The DB was created via the UI. Name:publication-graph, PW: publication-graph.
2. The DB was started via the UI.
3. The preprocessed files for import are placed in the respective input folder of the DB.

A 3 - Evolving the Graph

Proposed solution for Review: Review content and proposed decision will be added as properties on the PARTICIPATES_IN relationship. Since this relationship connects a reviewer to the review anyway, it is the logical place to store that information. Since number of reviewers for a journal can vary it is also *not* desirable to store this information as properties on the review (i.e. review_content_reviewer1, suggested_decision_reviewer1,...).
However, the *Review* will get a new property that reflects the overall decision of the review. The overall decision is computed via a query. If a review has a negative decision, i.e. the publication is declined, the previously created PUBLISHED_IN, JOURNAL_CITATION and JOURNAL_PUBLICATION relationships are removed because an unpublished paper should not count for h-index and other citation related queries. Also, for the purpose of this

assignment, the incoming CITES relationships to that paper are removed, following the intuition that an unpublished paper can't be cited.

Proposed solution for affiliation: Since several researchers might be affiliated with the same institution, the institutions are modeled as nodes. They also have a type property indicating if they are a company or university. Researchers are related to them via an AFFILIATED_WITH relationship.

B Querying

1. *Most cited papers*

The query returns the conference name and the corresponding 3 most cited paper titles as list. Please note that if a paper has several publications but only less then 3 of them are cited, also less then 3 papers will show up in the result for that conference. Most conferences will return 3 papers.

```
MATCH (paper:Paper)<-[:CITES]-(citing_paper:Paper),
(paper)-[:PUBLISHED_IN]->(:ConferenceEdition)<-[:HOLDS]-(conference:Conference)
WITH conference, paper, count(citing_paper) AS no_citations
ORDER BY conference.conference_name, count(citing_paper) desc
WITH conference, collect(paper.title) AS papers
WITH conference, papers[0..3] AS most_cited
RETURN conference.conference_name AS conference, most_cited
```

2. *Community*

This query returns a list of author names that participate in the community for each conference. Also the number of authors in that community are returned for convenience.

```
MATCH (c:Conference)-[:HOLDS]->(ce:ConferenceEdition),
(ce)<-[:PUBLISHED_IN]-(p:Paper)<-[:AUTHOR_OF]-(r:Researcher)
WITH c, r, count( DISTINCT ce) AS no_editions
WHERE no_editions >= 4 WITH c, collect(r.name) AS community
RETURN c.conference_name, community, size(community) AS community_size
```

3. *Impact factor*

The query returns all impact factors for journals, for which an impact factor can be computed in the given year. The parameterization is done in the program. When passing a year, year-1 and year-2 are computed and inserted into the query placeholders. The number of citations and publications is also returned.

```
MATCH (:Paper)-[jc:JOURNAL_CITATION]->(journal:Journal)
WHERE jc.year_cited = <YEAR> AND (jc.publication_year_cited_paper=<YEAR-2> OR
jc.publication_year_cited_paper=<YEAR-1>)
WITH journal, count(jc) AS no_citations
MATCH (:Paper)<-[jp:JOURNAL_PUBLICATION]-(journal)
WHERE jp.year = <YEAR-2> OR jp.year = <YEAR-1>
WITH journal, no_citations, count(jp) AS no_publications
```

```
RETURN journal.journal_name, no_citations, no_publications, (tofloat(no_citations) / no_publications) AS impact_factor
ORDER BY impact_factor desc
```

4. H-index

The query returns h-index for all Researchers in the db. That's why the name is included in the output. The optional match is included so that also a h-index of 0 can be computed.

```
MATCH (a:Researcher)-[ao:AUTHOR_OF]->(paper:Paper)
OPTIONAL MATCH (paper)<-[ci:CITES]-(citing_paper:Paper)
WITH a, paper, count(citing_paper) AS no_cit
WITH a, collect([a, paper, no_cit]) AS papers
UNWIND range(0, size(papers)-1) AS ind
WITH a, papers[ind][0] AS auth, papers[ind][1] AS pap, ind, papers[ind][2] AS no_cit,
CASE papers[ind][2] >= ind +1 WHEN true then 1 else 0 end AS in_count
RETURN a.name, sum( in_count) AS h_index ORDER BY h_index desc
```

C Recommender

Step 1

Created a constraint for a new label "Community". The created the community and related it to the relevant keywords by using a DEFINED_BY relationship.

```
MERGE (com:Community {community_name: "Database Community"})
WITH com, ["data management", "indexing", "data modeling", "big data", "data processing", "data storage", "data querying"] AS db_com_keywords
UNWIND db_com_keywords AS kw
MATCH (k:Keyword {keyword: kw}) MERGE (com)-[:DEFINED_BY]->(k)
```

Step 2

Identify number of publications for each journal, and then the number of papers that relate to the database community. Then the threshold of 90% of publications is checked.

```
MATCH (paper:Paper)-[:PUBLISHED_IN]->()<-[:HOLDS|ISSUES]-(pub)
WITH pub, count(paper) AS no_publications
MATCH (com:Community {community_name: "Database Community"})-[:DEFINED_BY]-(kw:Keyword) MATCH (kw)<-[:MAIN_TOPIC]-(paper:Paper)-[:PUBLISHED_IN]->()<-[:HOLDS|ISSUES]-(pub)
WITH pub, count( DISTINCT paper) AS no_in_community, no_publications, com
WITH pub, no_in_community, no_publications, toFloat(no_in_community) / no_publications AS community_participation, com WHERE community_participation >= 0.9
MERGE (com)<-[rt:RELATED_TO]-(pub)
SET rt.participation_rate = community_participation
```

Step 3

We broke this step down into two queries. The first query is doing the heavy lifting of identifying papers published as part of Journals/Conferences of the database community which is resource intensive. This information is persisted in the COMMUNITY_PAPER

relationship along with the number of citations from the community as property. This relationship can be used as the basis for several other queries. In particular, the second query builds on this relationship to quickly identify the top100 papers.

COMMUNITY_PAPER relationship:

```
MATCH (com:Community {community_name: "Database Community"})
MATCH (paper:Paper)-[cite:CITES]->(cited_paper:Paper)
MATCH (paper)-[:PUBLISHED_IN]->()<-[:HOLDS|ISSUES]-(pub)-[:RELATED_TO]->(com)
MATCH
(cited_paper)-[:PUBLISHED_IN]->()<-[:HOLDS|ISSUES]-(pub)-[:RELATED_TO]->(com)
WITH cited_paper, count(cite) AS no_citations, com
MERGE (com)-[cp:COMMUNITY_PAPER]->(cited_paper)
SET cp.community_citations = no_citations
```

Top X (in this case 100) papers:

```
MATCH (com:Community {community_name: "Database
Community"})-[cp:COMMUNITY_PAPER]->(paper:Paper)
WITH cp, cp.community_citations AS no_citations
ORDER BY no_citations desc WITH collect(cp) AS papers
WITH papers[0..100] AS top100papers
UNWIND top100papers AS highlight SET highlight.top100 = true
```

Step 4

```
MATCH (com:Community {community_name: "Database Community"})-
[cp:COMMUNITY_PAPER]->(paper:Paper)<-[:AUTHOR_OF]-(author:Researcher)
WHERE cp.top100 = true MERGE (com)-[:POTENTIAL_REVIEWER]->(author)
WITH author, count( DISTINCT paper) AS no_publications, com
WHERE no_publications >= 2 MERGE (com)-[:GURU]->(author)
```

D Graph Algorithms

The two algorithms chosen were the **PageRank** and **Node similarity**.

PageRank implementation:

```
CALL gds.graph.project(  'paper-rank',  'Paper',  'CITES')

CALL gds.pageRank.stream('paper-rank')
YIELD nodeId, score
RETURN gds.util.asNode(nodeId).title AS title, score
ORDER BY score DESC, title ASC
```

With this query we obtain a list of papers that have an assigned non-negative score. This score is generally speaking unbound and correlates with the size of the graph. In general, PageRank works by putting a score/weight on every node in the graph based on the importance of another node which has a relationship to the examined node. The scoring is done in an iterative process. By not specifying an iteration number, the default of 20 iterations is applied in this specific example. In the context of this application we consider

papers and their citation relationships. Thus, the papers are scored by the number and influence of the citations they have. The score of a paper increases with more citations and is also influenced by the weight/influence of the paper which cites it.
Essentially, this means that the higher the score of the paper, the more influential it is. Unsurprisingly, older papers tend to have higher scores than more recent papers, because they are more likely to have more citations and also more citations from influential papers. One useful application of this could be to use it as relevance score when searching papers by keywords: inputting some keywords in a search engine, the related papers (see for example the next section of node similarity) could be fetched and ordered by relevance using the scoring obtained from the PageRank algorithm. Another application could be to identify influential papers in general, and of certain communities specifically. However, comparing the page rank scores across communities has to be done with caution. Since some communities are more active than other communities, influential papers probably have different absolut scores depending on the community.

Node Similarity implementation:

```
CALL gds.graph.project( "similarity_of_papers", ["Paper", "Keyword"], "MAIN_TOPIC")

CALL gds.nodeSimilarity.stream("similarity_of_papers2")
YIELD node1, node2, similarity
WITH gds.util.asNode(node1) AS paper1, gds.util.asNode(node2) AS paper2, similarity
WHERE paper1 <> paper2
RETURN paper1.title as paper1, paper2.title AS paper2, id(paper1), id(paper2), similarity
ORDER BY similarity DESC, paper1, paper2
```

For the application of the node similarity we identified papers and their keywords as a potential use case. The query returns two papers and a similarity score between 0 and 1, that indicates how similar they are in terms of their keywords. To this end, the node similarity algorithm uses similarity measures like Jaccard Similarity or the Overlap coefficient to assess the similarity of a set of nodes to another set of nodes. The comparable nodes are considered similar by the number of neighboring nodes (in this case keywords) they share. The output score ranges from 0 to 1, where 0 represents no common keywords of the papers considered, and 1 represents two papers having exactly the same keywords. Every value in between 1 and 0 represents the ratio of the intercept. When similarity is of importance, values closer to 1 are of interest, since it suggests the papers having similar keywords. Examining the function call more closely, the "*WHERE*" clause prunes out comparisons of the paper with itself, as this obviously is redundant. Furthermore, it should be noted in the output that there seem to be duplicate rows, and the reason is that the relationship between papers through keywords work both ways, so technically they represent different relationships, even though they depict the same results.
In the context of our graph, checking for similarity between papers by keywords opens up several possible applications. One useful application would be to offer similar papers to a reader, depending on which paper they just read. This stems from the idea that papers on the same topic probably share the same keywords. A possible implementation of this could for example be links in the bottom of a webpage saying something in the lines of "you may be interested…/read more…". In essence the node similarity is thus also a metric that can be used to score the relevance of a paper.