Semantic Data Management
April 2024

LAB 1
Graph-Based Data Management Report

Berat Furkan Koçak
Rana İşlek

Universitat Politècnica de Catalunya

Big Data Management and Analytics

A. Modeling, Loading, Evolving

A.1 Modeling

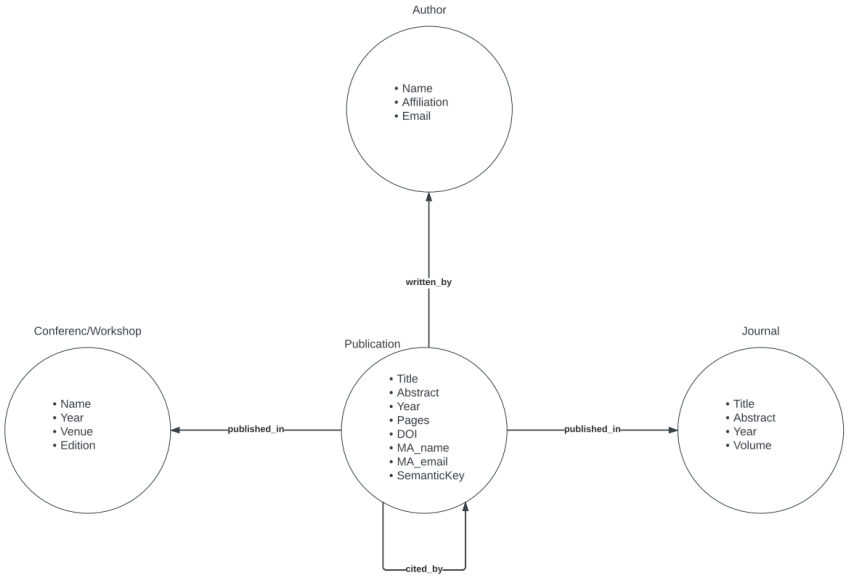


Figure 1: Our Graph Schema Design

First iteration of our graph database considered the most important attributes for each entity as shown above.

In this design, “publication” can be seen as the core node of the graph model since the model is based on research papers and their interactions. “Publication” covers any research paper without looking at where it was published or what kind of paper that is. In this current version, it has eight different attributes which are title of the publication, abstract of the publication, year that it was published in, the unique DOI number, name and email of the corresponding main author of the publication and a key which we called as “semantic_key” for now. One can find our design rationale below:

- We added an additional feature “semantic_key” with respect to our term project’s goal, enabling recommendations between semantically relevant papers, instead of simple keywords. These keys are vector representations provided by SPECTER2 embeddings. These embeddings are trained on vast amounts of text data and are capable of capturing nuanced semantic relationships between words and phrases.
- The design ensures that journals and conferences/workshops are represented as different nodes since they are separated majorly with their attributes.
    - This provides two main advantages: Maintenance and non-redundancy. Maintenance is achieved by having a single responsibility like architecture for each node. Moreover, since the number of shared attributes are decreased, redundancies that can occur by updates are also minimized.

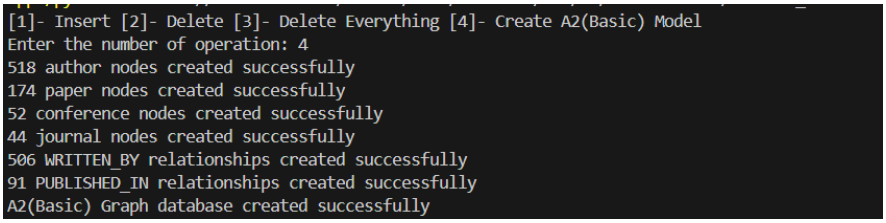
- Edition attribute creates uniqueness for same conferences held in the same year and volume attribute does the same thing for journals.
- Citations are very important in terms of h-index and impact factor calculations. However, we find it not worthy of creation of a separate node for it and instead made it a relation between publication nodes.
- Authors are designated as nodes instead of a property of a publication. This design choice enhanced the reusability of authors participating in different relations as well as possible future inclusion in bigger communities. Moreover, authors are referenced by the written_by relationship from publications, rather than duplicating author details in each publication node, reducing the number of redundant duplications.
- The design ensures that each entity (Author, Publication, Conference/Workshop Edition, Journal) is represented as a separate node, making it easier to maintain and update specific properties or relationships without affecting others.
- The graph design allows for the reusability of nodes and relationships. For example, an author can write multiple publications, and a publication can be presented in multiple conferences/workshops or published in multiple journals. This gives us flexibility to especially adapt this considering the design decisions of our term project.
- The design minimizes redundancy by ensuring that each piece of information is stored only once. For instance, author information is stored in the Author node and referenced by the Writes relationship, rather than duplicating author details in each publication node. Similarly, review information is stored in the Review node and associated with publications through the Reviewed by relationship, avoiding duplication of review data in each publication node.

A.2 Instantiating/Loading

We used *Semantic Scholar* as our data source. *Semantic Scholar* provides a powerful API enabling us to extract various information for different entities necessary for this project such as authors, papers and venues. We created specific pre-process scripts to map nodes and data, transform it and adapt it to the graph we designed and use the bulk load functionality (LOAD_CSV) to load the graph.

Each preprocessing script can be found in our *GitHub repository* under the preprocess folder. Also, each CSV file generated is placed under the data folder. This programmatic choice enabled us to quickly make changes on small parts of the entities and add mock data whenever necessary without generating huge CSV files.

Our program file PartA.2_KocakIslek.py incorporates a very useful app, enabling insertion, deletion of each node as well as deleting and instantiating the designed model in A1. All the Cypher queries for data loading can be found in this python script. To create the first version of our model, please *select 4.*



A.3 Evolving the Graph

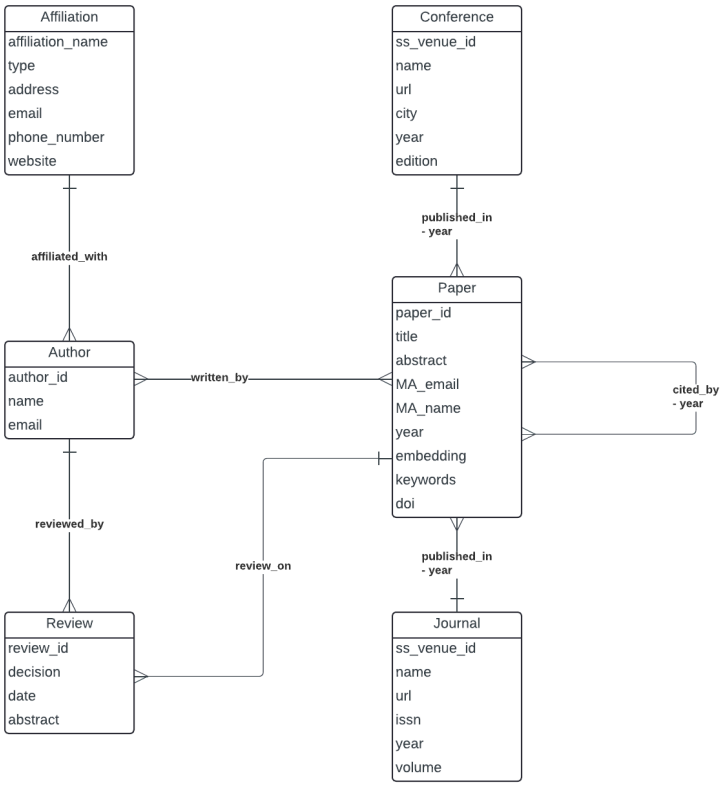


Figure 2: Our model after evolving the graph

With respect to the requirement in A.3, to evolve our graph database, we introduced several modifications to accommodate the changes in data structure and requirements.

Firstly, we expanded the model to store reviews alongside reviewers. Each review now includes both textual content and a suggested decision. This change enables us to track reviewer feedback more comprehensively. Also each review has a final decision assigned.

Moreover, we extended the model to include the affiliation of authors, allowing them to be associated with organizations such as universities or companies.

In addition, we incorporated year information for cited_by and published_in relations. These were necessary in order to answer h-Index and Impact Factor queries in part B.

We used Faker library to create mock data for attributes such as name, country and email for various entities. We also used the Natural Language Toolkit (nltk) library in order to extract keywords from papers' abstracts. These adjustments enhance the database's capability to handle diverse data types and support more nuanced analyses.

PartA.3_KocakIslek.py provides you a very useful application to deploy the evolved database into Neo4j using option 2.

```
[1]- Delete Everything [2]- Create A3(Evolved) Model [3]- Exit
Enter the number of operation: 2
1566 paper nodes created successfully
518 author nodes created successfully
35 conference nodes created successfully
28 journal nodes created successfully
4554 WRITTEN_BY relationships created successfully
475 PUBLISHED_IN relationships created successfully
35 affiliation nodes created successfully
518 AFFILIATED_WITH relationships created successfully
180 review nodes created successfully
717 REVIEWED_BY relationships created successfully
174 REVIEW_ON relationships created successfully
1395 CITED_BY relationships created successfully
******A3(Evolved) Graph database created successfully******
```

B. Querying

PartB_KocakIslek.py provides you a very useful application that answers each of the required queries interactively.

```
[1] - Run query 1 - Find the top 3 papers with the most citations for each conference
[2] - Run query 2 - Find the community of authors who have published in at least 4 editions of the same conference
[3] - Run query 3 - Find Impact Factor of Journals
[4] - Run query 4 - Find the most influential authors
[5] - Exit
Please enter the query number you want to run:
```

Query 1: Identifying Top Cited Papers per Conference

Cypher Statement:

```
MATCH (c:Conference)<-[:PUBLISHED_IN]-(p:Paper)
OPTIONAL MATCH (p)<-[:CITED_BY]-(citation)
WITH c, p, COUNT(citation) AS citations
ORDER BY c.name, citations DESC
WITH c, COLLECT(DISTINCT {paper: p, citations: citations}) AS papers
RETURN c.name AS Conference,
    [paper in papers | paper.paper.title][..3] AS TopPapers,
    [paper in papers | paper.citations][..3] AS Citations
```

This query matches papers published in conferences, optionally matches citations for each paper, counts citations, orders by conference name and citation count, and collects the top 3 cited papers for each conference.

Query 2: Community of Authors with Repeated Conference Editions

Cypher Statement:

```
MATCH (c:Conference)-[:PUBLISHED_IN]-(p:Paper)-[:WRITTEN_BY]-(a:Author)
WITH a.name AS author_name, c.name AS conference_name, collect(c.edition) AS editions
WHERE size(editions) >= 4
WITH collect(author_name) as community,conference_name
RETURN conference_name,community
```

In the second query, we identified communities of authors who have published in at least 4 editions of the same conference.

Query 3: Impact Factor of Journals

```
MATCH (journal:Journal )<-[published:PUBLISHED_IN]-(paper:Paper)
WHERE published.year >= {year} - 2
AND published.year <= {year} - 1
WITH journal, paper
MATCH (paper)<-[citation:CITED_BY]-(citing_paper:Paper)
WHERE citation.year = {year}
WITH journal, COUNT(DISTINCT citing_paper) AS citations_received
MATCH (journal)<-[published_in: PUBLISHED_IN]-(paper)
WHERE published_in.year >= {year} - 2
AND published_in.year <= {year} - 1
WITH journal, COUNT(DISTINCT paper) AS citable_items, citations_received
RETURN journal.name AS journal_name,
    SUM(citations_received) AS citations_received_current_year,
    SUM(citable_items) AS citable_items_current_year,
    SUM(citations_received) / SUM(citable_items) AS impact_factor
```

This query matches journals and papers published within the last two years, counts citations received in the current year, and calculates the impact factor.

Query 4: Most Influential Authors by H-Index

```
MATCH (a:Author)-[:WRITTEN_BY]-(p:Paper)-[:CITED_BY]-(c:Paper)
WITH a, p, COUNT(c) AS citations
ORDER BY citations DESC      You, 1 second ago • Uncommitted changes
WITH a, COLLECT(citations) AS citationList
WITH a, [i IN RANGE(1, SIZE(citationList)) | REDUCE(s = 0, x IN citationList[..i] | s + x)] AS cumulativeCitations
WITH a, [i IN RANGE(1, SIZE(cumulativeCitations)) | CASE WHEN cumulativeCitations[i - 1] >= i THEN i ELSE 0 END] AS hIndexes
WITH a, MAX(hIndexes) AS hIndex
RETURN a.name, hIndex
```

This query calculates the H-Index for each author by counting citations for their papers, generating cumulative citation counts, and deriving the H-Index based on these counts.

C. Recommender

1. Objective

The objective of *Section C* is to develop a recommender system within the *Neo4j environment*, aimed at identifying potential reviewers for the database community. This process involves several key steps, each leveraging Cypher queries to manipulate and analyze the graph data.

2. Process Overview

The recommender system's development is divided into four main steps, each designed to progressively refine the selection of potential reviewers based on predefined criteria related to the database research community.

Step 1: Identifying the Database Community

In the first step, we defined the database community based on a set of keywords associated with research topics. Keywords are already given in the assignment which are: data management, indexing, data modeling, big data, data processing, data storage and data querying.

In the papers_details.csv, one can find “keywords” extracted from each paper’s abstract using the “nltk” library in Python. We decided to store “keywords” information in string format since neo4j accepts only particular formatted data types to do search with queries.

Moreover, we splitted the given keywords to single words in order to have an ease and efficiency to compare keywords. This way we could match more keywords and catch more relations between papers.

Query:

```
MATCH (p:Paper)
WHERE p.keywords IS NOT NULL AND ANY(keyword IN split(toString(p.keywords), ",")
WHERE toLower(keyword) IN ['data', 'management', 'indexing', 'modeling', 'big data', 'processing', 'storage', 'querying'])
SET p:DatabaseCommunity
RETURN p
LIMIT 25
```

This query labels papers that match the specified keywords, effectively grouping them under the *DatabaseCommunity* label for further analysis.

Step 2: Identifying Related Conferences and Journals

In the second step, we found publication venues that predominantly feature works from the database community.

Query:

```
MATCH (p:DatabaseCommunity)-[:PUBLISHED_IN]->(v)
WITH v, COUNT(p) AS PapersCount
SET v:RelatedToDatabaseCommunity
RETURN v, PapersCount
LIMIT 25
```

This query assesses the concentration of database community papers within venues, we can identify those most relevant to the field, labeling them as *RelatedToDatabaseCommunity*.

Step 3: Identifying Top Papers

In the third step, we highlighted the most cited papers within the database community as primary candidates for review.

Query:

```
MATCH (p:DatabaseCommunity)-[:CITED_BY]->(cited:Paper)
WITH p, COUNT(cited) AS citations
ORDER BY citations DESC
LIMIT 100
SET p:TopPaper
```

This query ranks papers based on their citation count, assuming that higher citations correlate with higher relevance and influence within the community.

Step 4: Identifying Potential Reviewers and Gurus

In the fourth step, we determined *potential reviewers* from the authors of the top papers and identified *gurus* within the field.

Query for Potential Reviewers:

```
MATCH (a:Author)-[:WRITTEN_BY]->(p:TopPaper)
WITH a
SET a:PotentialReviewer
RETURN a.name AS AuthorName
LIMIT 25
```

Query for Gurus:

```
MATCH (a:Author)-[:WRITTEN_BY]->(p:TopPaper)
WITH a, COUNT(p) AS TopPapers
WHERE TopPapers >= 2
SET a:Guru
RETURN a.name AS GuruName, TopPapers
LIMIT 25
```

These two queries find authors of the top papers which are considered capable reviewers, with those contributing to multiple top papers (>=2) being classified as gurus for their significant contributions to the field.

3. Conclusion

This recommender system provides a structured approach to identifying *potential reviewers in the database community*. Through careful query design and execution, *the system segments the research community*, *highlights influential works*, and *pinpoints key contributors*, offering valuable insights for editors and conference chairs in their *selection of reviewers*.

D. Graph Algorithms

1. Algorithm Selection

For the analysis of our academic graph dataset, we selected two key algorithms from the Neo4J Graph Data Science (GDS) library: PageRank and the Louvain method. We made this decision because of their applicability to our domain of research publication networks and the specific insights we aimed to derive.

PageRank: This centrality algorithm is known for its utility in identifying influential nodes within a network. In the context of research publications, applying PageRank enables us to spotlight the most influential papers or authors, as determined by citation networks. The relevance of PageRank to our domain lies in its ability to identify papers that significantly impact the academic community, either through foundational insights or innovative discoveries.

Louvain Method: This community detection algorithm is proficient at uncovering clusters within networks, which in our case can correspond to groups of papers or authors around specific research topics or methodologies. This algorithm helps identify the structure of academic collaboration and topic specialization, revealing how research communities are interconnected.

2. Implementation and Findings

a. PageRank Algorithm (Centrality)

Cypher Call:

```
CALL gds.pageRank.stream('{graph_name}')
YIELD nodeId, score
RETURN id(gds.util.asNode(nodeId)) AS paperId, gds.util.asNode(nodeId).title AS paper, score
ORDER BY score DESC
LIMIT 10;
```

Parameters Selected: The primary parameters were the nodeProjection set to 'Paper' and the relationshipProjection set to 'CITED_BY' with a 'REVERSE' orientation. These choices aim to model the influence flow from cited papers back to the citing paper, thus accurately representing the citation network's dynamics.

Results Obtained: The results highlighted the top 10 papers by PageRank score, revealing those with the highest influence within the academic graph. Titles such as "Serious Games in Surgical Medical Education" and "Database resources of the National Center for Biotechnology Information" topped the list.

Interpretation: These results underscore the pivotal roles certain papers play within their respective fields. A high PageRank score suggests these works are widely recognized and referenced, serving as key knowledge foundations. This insight is very valuable for editors and conference chairs in identifying trendsetting research and authors whose work commands broad academic attention.

b. Louvain Method (Community Detection)

Cypher Call:

```
CALL gds.louvain.stream('{graph_name}')
YIELD nodeId, communityId
RETURN communityId, collect(gds.util.asNode(nodeId).name) AS authors
ORDER BY size(authors) DESC
LIMIT 5;
```

Parameters Selected: The algorithm was applied with a nodeProjection of 'Author' and a relationshipProjection for 'WRITTEN_BY', reflecting the co-authorship ties without specifying a direction, to analyze the natural clustering of authors.

Results Obtained: Identified several distinct author communities, such as one led by S. Fuentes and another by Wookhee Min. These communities represent clusters of researchers collaborating or sharing thematic interests.

Interpretation: The Louvain method's output shows the collaborative structure within the academic domain, pinpointing groups of researchers that frequently co-author or work within similar research niches. For academia-focused organizations, understanding these clusters aids in fostering collaboration, organizing conferences, and editing special journal issues around these thriving research communities.

GitHub Link: https://github.com/furkanbk/SDM-P1-GRAPH.git