Universitat Politècnica de Catalunya

Facultad de Informática de Barcelona

Lab Assignment 1: Property Graph

Semantic Data Management

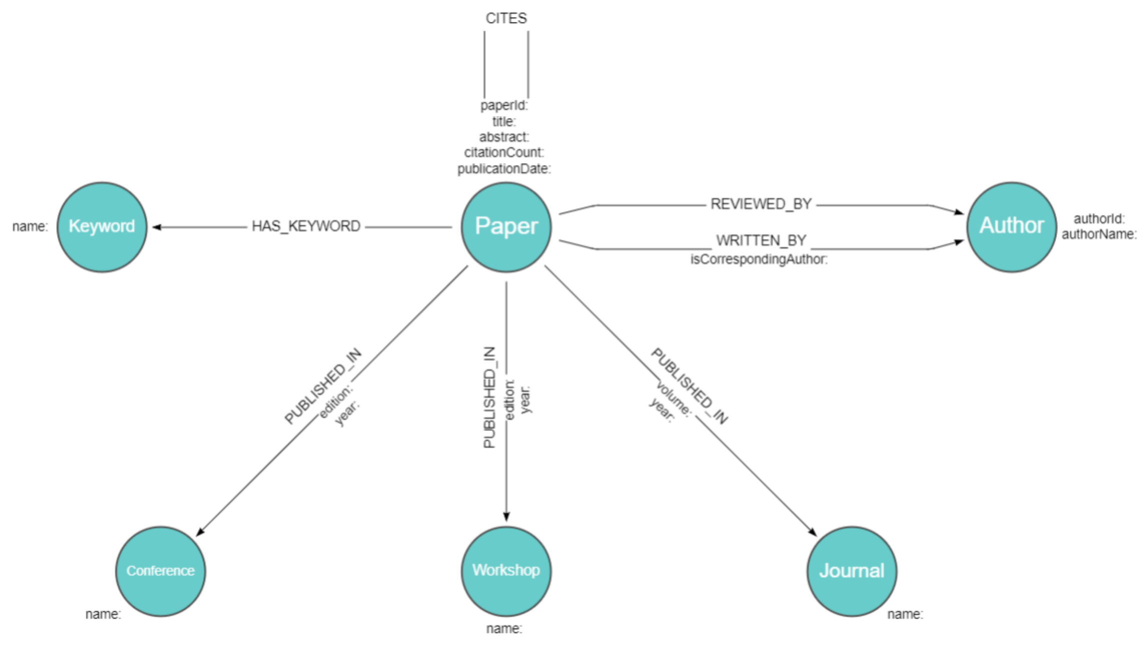
Spring 2024

Authors:
Sony Shrestha
Hareem Raza

Professor:
Anna Queralt

A.1. Modeling

The following diagram depicts the visual representation of the graph designed.



The graph model consists of **six distinct nodes** along with **seven relationships** connecting these nodes. Here is the breakdown of each node, relationship and its associated properties:

Nodes

1. **Paper** represents academic papers that were published in journals, conferences, or workshops. Properties associated with entity Paper includes paperId, title, abstract, citationCount and publicationDate
2. **Keyword** represent the topics or themes that are associated with papers. Properties associated with entity Keyword include keyword.
3. **Author** represents the authors who have written the papers and may also serve as reviewers. Properties associated with entity Author include authorId and authorName.
4. **Conference** represents the academic conferences where papers are presented. Properties associated with entity Conference include its name.
5. **Workshop** represents workshops that are often more specialized or focused than conferences, where participants have the opportunity to present their papers. Properties associated with entity Workshop include its name.
6. **Journal** represents scientific journals where papers are published. Properties associated with Journal include its name.

Relationships

1. **WRITTEN_BY** (Paper → Author) connects a Paper node to an Author node, indicating that the paper was written by the author. It has a property named **isCorrespondingAuthor** which is a boolean flag indicating whether the author is the corresponding author for that paper or not.
2. **HAS_KEYWORD** (Paper → Keyword) connects a Paper node to a Keyword node, indicating that the paper contains certain keywords.
3. **PUBLISHED_IN** (Paper → Journal/Conference/Workshop) connects a Paper node to

a Journal/Conference/Workshop node, indicating the journal or conference or workshop where the paper was published. The properties of this relationship vary slightly depending on the publication type. For **Journals**, the properties are ***volume*** and ***year*** while for **Conference** and **Workshop**, the properties include ***edition*** and ***year***. It represents the volume/edition and year in which the paper was published.

4. **CITES** (Paper $\rightarrow$ Paper) connects a Paper node to another Paper node, representing citations between papers.
5. **REVIEWED_BY** (Paper $\rightarrow$ Author) connects the Paper node to the Author node, indicating that the paper was reviewed by the corresponding reviewer.

Design Decisions and Assumptions

1. In our graph design, we opted to distinguish between journals, conferences, and workshops by representing them as separate nodes. This decision was made to streamline and simplify our queries, enabling **more direct access** to each publication type.
2. We decided to **incorporate properties** such as **year**, **edition**, and **volume** directly into the relationships rather than creating separate nodes for them. This is because properties are relevant only when a paper is published in a conference, journal, or workshop. Moreover, we assume that queries involving aggregation by year would be relatively infrequent in our use case.
3. We integrated the **isCorrespondingAuthor** property into the Author-Paper relationship, specifying the designated corresponding author for a paper. It **eliminates redundant connections**, directly linking the isCorrespondingAuthor property to the specific Author-Paper pair, thereby streamlining the designation process.
4. To maintain data integrity and **minimize redundancy**, our data model **does not differentiate between authors and reviewers** as separate nodes, considering that reviewers are also authors. By utilizing the existing author node to represent both roles and employing the REVIEWED_BY relationship, we capture both authorship and reviewing contributions.

A.2. Instantiating and Loading

The data was systematically compiled and prepared for integration into the Neo4j graph database through a multi-stage process, utilizing both the **Semantic Scholar API** and a series of Python scripts. The sequence of steps that were employed are provided below.

Retrieval of Paper IDs: Utilizing the Semantic Scholar API, we queried for Paper IDs corresponding to 10 different keywords like data management, big data etc.

Fetching of Paper Details: With the obtained Paper IDs, we used the Semantic Scholar's batch API to obtain detailed information for each publication, subsequently storing this information in JSON format. The extracted data included fields such as paperId, authors, title, venue, publicationVenue, year, abstract, citationCount, fieldsOfStudy, publicationTypes, publicationDate, citations, journal, and references.

Data Preprocessing: The preprocessing of raw data obtained from the Semantic Scholar API involved the following actions.

1. **Standardization of Dates:** Publication dates were converted to a consistent format and missing dates were populated with a default value based on available year information or a predefined default year when necessary.

2. **Venue and Publication Type Identification:** Publications were classified as published in either Conferences, Journals, or Workshops by analyzing the venue name and type. Entries that could not be classified were omitted from the dataset to maintain a high data quality standard.
3. **Author Details Extraction:** Essential author information, including IDs and names, was extracted. Missing author IDs were automatically generated.
4. **Extraction of Edition from Conference name:** Edition information which was integrated into the conference name was extracted and stored as a separate field.
5. **Synthetic Data Generation for Citation:** Since citation for most of the papers were not available, this information was synthetically generated to mirror the distribution found in the original Semantic Scholar data. This process involved randomly selecting paper IDs from the pool of available publications.
6. **Synthetic Data Generation for Keywords:** We employed two methods to generate keywords for papers. Firstly, we predefined a set of keywords for each query type and randomly selected four of them to be assigned as keywords for the corresponding paper. Secondly, we utilized the **Spacy** library to extract five keywords from the paper's abstract.
7. **Synthetic Data Generation for Reviewers:** Three authors were randomly selected from the list of available authors, ensuring that an author cannot review their own paper.
8. **Synthetic Data Generation for Corresponding Author:** One of the authors of the paper is selected as corresponding author for that paper.

Splitting data by Node and Relationship Types: To efficiently load the data into Neo4j, we splitted the preprocessed data into separate files, each corresponding to a specific type of node or relationship within the graph (e.g. years, authors, keywords, journals, workshops, conferences, etc.). This segregation was important for the import process and ensured that each element of the dataset could be accurately represented within the graph structure.

File Management for Neo4j Import: Finally, the prepared files were then transferred to Neo4j's import directory and then loaded using a load script (`load_data.cypher`).

A.3. Graph Evolution

Incorporating Reviews and Decisions

In the first part of graph evolution, we extended our model to store the review sent by each reviewer along with their suggested decisions. Randomized review content and associated decisions for each paper review were generated, which were then allocated to respective reviewers, matching the total count of reviewers per paper. The modifications done to accommodate this change are explained below.

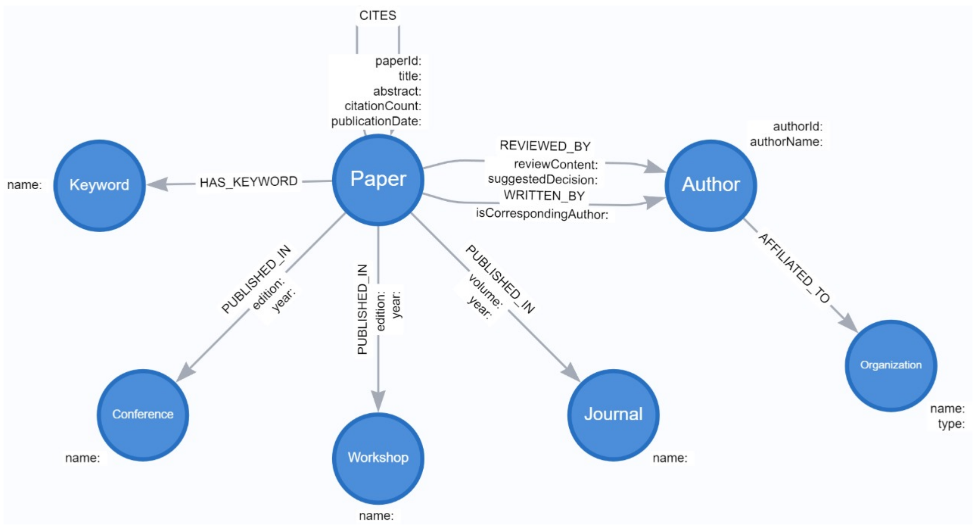
1. **Review Information:** For each paper, we now add the content of the review sent by each reviewer. This is achieved by extending the REVIEWED_BY relationship to include property named reviewContent, a textual representation of the reviewer's feedback.
2. **Suggested Decision:** Alongside the review content, the suggested decision (E.g. accept, reject, revise) proposed by each reviewer is also stored. This is reflected as the suggestedDecision property within the REVIEWED_BY relationship.

Storing Author Affiliations

In the second part of graph evolution, the model was enriched with author affiliations. Organization names and relationships with each author were generated and assigned randomly. The modifications done to accommodate this change are explained below.

1. **Organization Node:** A new node with label Organization was created to represent entities such as universities, research institutes, and companies. Each organization node is characterized by a name and a type.
2. **Affiliation Relationship:** The AFFILIATED_TO relationship was introduced, connecting Author node to Organization node, indicating organization to which an author is affiliated to.

The following diagram depicts the visual representation of the graph after evolution.



B. Querying

We executed the queries below in a graph model created with the following number of nodes and relationships.

Type	Label	Count
Node	Keyword	2408
	Journal	301
	Conference	120
	Workshop	13
	Paper	1848
Relationship	WRITTEN_BY	8076
	HAS_KEYWORD	11920
	PUBLISHED_IN	1848
	CITES	310794
	REVIEWED_BY	5544

Table 1: Number of nodes and relationships in the graph model

Query 1: Find the top 3 most cited papers of each conference

```
MATCH (p:Paper)-[:PUBLISHED_IN]->(c:Conference)
OPTIONAL MATCH (p1:Paper)-[:CITES]->(p)
WITH c, p.paperId AS paperId, count(distinct p1.paperId) AS citationCount
ORDER BY c.name ASC, citationCount DESC
WITH c, collect({paperId: paperId, citationCount: citationCount}) AS
    papersByConference
RETURN c.name AS ConferenceName, papersByConference[0..3] AS TopPapers;
```

Query 2: For each conference find its community

```
MATCH (p:Paper)-[:WRITTEN_BY]->(a:Author)
MATCH (p)-[pi:PUBLISHED_IN]->(c:Conference)
WITH a.authorId AS authorId, c.name AS conferenceName, COUNT(DISTINCT pi.edition
    ) AS numDistinctEditions
WHERE numDistinctEditions >= 4
RETURN conferenceName, COLLECT(authorId) AS community;
```

Query 3: Find the impact factors of the journals in your graph

```
MATCH (j:Journal)
WITH j.name AS journalName, 2021 AS year
OPTIONAL MATCH (last1_p:Paper)-[last1:PUBLISHED_IN{year:year-1}]->(j:Journal{
    name:journalName})
WITH journalName,year,collect(last1_p.paperId) as paperId1
OPTIONAL MATCH (last2_p:Paper)-[last2:PUBLISHED_IN{year:year-2}]->(j:Journal{
    name:journalName})
WITH journalName, year,paperId1,collect(last2_p.paperId) as paperId2
with journalName, year, paperId1+paperId2 as paperIds,size(paperId1)+size(
    paperId2) as numPaperPublished_last2yrs
UNWIND paperIds AS paperId
with journalName,year, paperId, numPaperPublished_last2yrs
MATCH (p1:Paper)-[:CITES]->(p2:Paper{paperId:paperId})
MATCH (p1)-[:PUBLISHED_IN{year:year}]->(j:Journal)
with journalName,year,numPaperPublished_last2yrs,count(*) AS numCitations
return journalName, numCitations/numPaperPublished_last2yrs AS impact_factor
ORDER BY impact_factor DESC;
```

Query 4: Find the h-indexes of the authors in your graph

```
MATCH (a:Author)<-[:WRITTEN_BY]-(p:Paper)
OPTIONAL MATCH (p)<-[:CITES]-(citingPaper:Paper)
WITH a, p, COUNT(citingPaper) AS citations
ORDER BY a.name, citations DESC
WITH a, COLLECT(citations) AS citationCounts
WITH a, citationCounts, RANGE(1, SIZE(citationCounts)) AS indices
UNWIND indices AS idx
WITH a.name AS authorName, idx AS h, citationCounts
WHERE citationCounts[idx-1] >= idx
RETURN authorName, MAX(h) AS hIndex
ORDER BY hIndex DESC;
```

For each query, the query execution time along with the number of records returned are summarized below.

Query	Number of Rows Returned	Execution Time (s)
1	120	2.2300
2	4	0.0100
3	121	4.8117
4	3507	0.4656

Table 2: Query Performance Metrics

C. Recommender

For the first part of the recommender, a node with the label **Community** and a property named **name** with the value **Database Community** is created. The Database Community contains keywords like Data Management, Indexing, Big Data, Data Processing, Data Storage, Data Querying represented by the relationship **CONTAINS** between Community and Keyword.

```
CREATE (c:Community{name: 'Database Community'});
UNWIND ['Data Management', 'Indexing', 'Big Data', 'Data Processing','Data
    Storage', 'Data Querying'] AS keywordName
MATCH (k:Keyword {keyword:keywordName})
MATCH (c:Community{name: 'Database Community'})
MERGE (c)-[:CONTAINS]->(k);
```

Recommender Part 1: Create Community node of type Database

The determination of whether a conference or journal is related to the Database Community is based on the examination of the papers they publish and the keywords they contain. If 90% of the papers published in a conference or journal contain at least one keyword from the database community, it is related to that community. This relationship is represented by the **RELATED_TO** relationship between the Journal/Conference and the Community.

```
MATCH (k1:Keyword)<-[:HAS_KEYWORD]-(p:Paper)-[:PUBLISHED_IN]->(jc)
WHERE jc:Journal OR jc:Conference
OPTIONAL MATCH (dc:Community {name:"Database Community"})-[:CONTAINS]->(k1)
WITH
    jc.name AS journalOrConference,
    p.paperId AS paperId,
    count(DISTINCT dc) AS isComm
WITH
    journalOrConference,
    count(DISTINCT paperId) AS numPaperPublished,
    sum(isComm) AS numPaperPublishedComm,
    ((toFloat(sum(isComm)) / toFloat(count(DISTINCT paperId))) * 100) AS
   percentage
WHERE toFloat(numPaperPublishedComm) > 0 AND percentage > 90
WITH journalOrConference, numPaperPublished, numPaperPublishedComm, percentage
MATCH (entity {name: journalOrConference})
MATCH (dc:Community {name:"Database Community"})
MERGE (entity)-[:RELATED_TO]->(dc);
```

Recommender Part 2: Find journals and conferences related to Database Community

The top 100 papers from these journals and conferences, based on the number of citations from papers within the same database community, are identified and labeled as **TopDBCommPaper**.

```
MATCH (p1:Paper)-[:PUBLISHED_IN]->(jc1)-[:RELATED_TO]->(dc:Community {name:"
    Database Community"})
MATCH (p2:Paper)-[:PUBLISHED_IN]->(jc2)-[:RELATED_TO]->(dc:Community {name:"
    Database Community"})
MATCH (p2)-[:CITES]->(p1)
WITH p1, count(distinct p2.paperId) AS citationCount
ORDER BY citationCount DESC
LIMIT 100
SET p1:TopDBCommPaper;
```

Recommender Part 3: Identify Top Papers of these Conference/Journal

Authors of any of these top-100 papers are potential good matches for reviewing database papers, represented by the **POTENTIAL_REVIEWER** relationship between the Author and the Community. If an author has written more than two top-100 papers, they are considered a guru for the community, denoted by the **GURU** relationship between the Author and the Community.

```
MATCH (p:TopDBCommPaper)-[:WRITTEN_BY]->(a:Author)
MATCH (c:Community{name:"Database Community"})
MERGE (a)-[:POTENTIAL_REVIEWER]->(c);

MATCH (p:TopDBCommPaper)-[:WRITTEN_BY]->(a:Author)
MATCH (c:Community{name:"Database Community"})
WITH a,count(p) as cnt
WHERE cnt>=2
MERGE (a)-[:GURU]->(c);
```

Recommender Part 4: Identify Potential Good Match and Gurus

D. Graph Algorithms

Finding Similarity between two Papers

The first graph algorithm we implemented performs **node similarity analysis** on a graph named `paper_has_keyword_graph`. This graph represents papers and their associated keywords, where the relationship HAS_KEYWORD connects papers to keywords.

Data Obtained: The result provides pairs of papers along with their similarity scores, where similarity indicates how closely are papers related based on keywords they share.

Domain Interpretation: In the context of academic papers, this analysis helps to identify papers that have similar topics or themes based on the keywords they contain. Papers with higher similarity scores are likely to cover similar subject matter or contribute to related research areas. This information can be valuable for researchers looking for related work or exploring interdisciplinary connections within a field. By identifying papers with high similarity scores, researchers can gain insights into the landscape of existing literature and potentially discover new avenues for research or collaboration.

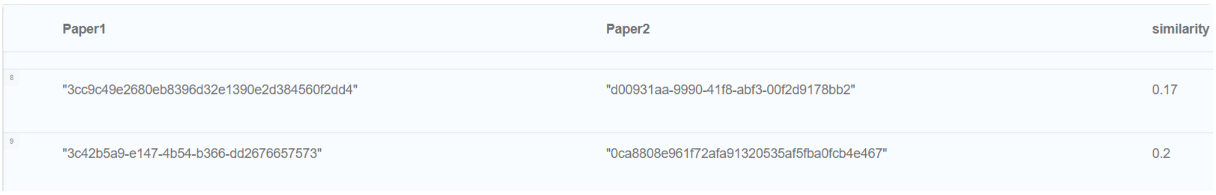
```
CALL gds.graph.exists("paper_has_keyword_graph") YIELD exists
WITH exists
WHERE exists
CALL gds.graph.drop("paper_has_keyword_graph") YIELD graphName
RETURN graphName + " dropped" AS status;

CALL gds.graph.project(
    "paper_has_keyword_graph",
    ["Paper", "Keyword"],
    {
        HAS_KEYWORD: {
            properties: {
                strength: {
                    property: "strength",
                    defaultValue: 1.0
                }
            }
        }
    }
);

CALL gds.nodeSimilarity.stream("paper_has_keyword_graph")
YIELD node1, node2, similarity
RETURN distinct gds.util.asNode(node1).paperId AS Paper1, gds.util.asNode(node2)
    .paperId AS Paper2, round(100 * similarity)/ 100 as similarity
ORDER BY similarity DESCENDING, Paper1, Paper2;
```

Query for finding Similarity between two Papers

The following diagram depicts the result of first graph algorithm implemented.



	Paper1	Paper2	similarity
8	"3cc9c49e2680eb8396d32e1390e2d384560f2dd4"	"d00931aa-9990-41f8-abf3-00f2d9178bb2"	0.17
9	"3c42b5a9-e147-4b54-b366-dd2676657573"	"0ca8808e961f72afa91320535af5fba0fcb4e467"	0.2

Finding PageRank of Publications

The second graph algorithm we implemented performs **PageRank analysis** on a graph named `paper_cites_graph`. This graph represents academic papers and their citation relationships, where the CITES relationship denotes citations between papers.

Data Obtained: The result provides a list of papers along with their PageRank scores, ordered by score in descending order.

Domain Interpretation: PageRank analysis helps to identify the most influential papers within the citation network. Papers with higher PageRank scores are those that are frequently cited by other papers, indicating their significance and impact within their respective research fields. Researchers can use this information to identify seminal works, track the spread of ideas, and assess the influence of specific publications on the broader academic community.

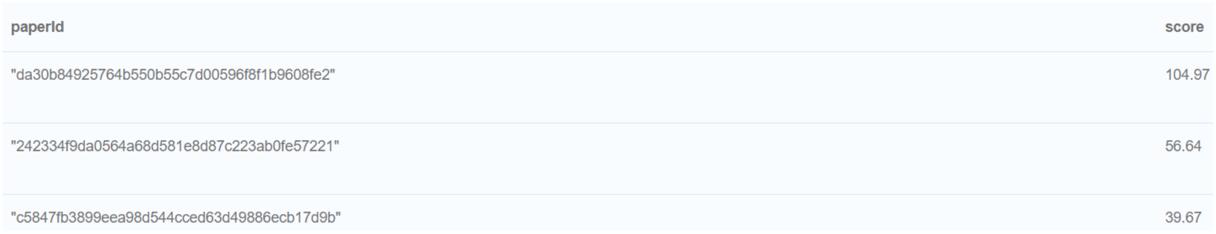
```
CALL gds.graph.exists("paper_cites_graph") YIELD exists
WITH exists
WHERE exists
CALL gds.graph.drop("paper_cites_graph") YIELD graphName
RETURN graphName + " dropped" AS status;

CALL gds.graph.project(
  "paper_cites_graph",
  "Paper",
  "CITES"
);

CALL gds.pageRank.stream("paper_cites_graph")
YIELD nodeId, score
RETURN gds.util.asNode(nodeId).paperId AS paperId, round(100*score)/100 as score
ORDER BY score DESC, paperId ASC;
```

Query for finding PageRank of Publications

The following diagram depicts the result of second graph algorithm implemented.



paperId	score
"da30b84925764b550b55c7d00596f8f1b9608fe2"	104.97
"242334f9da0564a68d581e8d87c223ab0fe57221"	56.64
"c5847fb3899eea98d544cced63d49886ecb17d9b"	39.67