

# Neo4j LAB Assignment Implement SDM UPC

Yutao Chen\*, Min Zhang\*

{yutao.chen,min.zhang}@estudiantat.upc.edu

In this project we will be using Neo4j technology to create and manage graphs, and query and process them. We will use innovative methods to solve these problems. The problem list is divided into four parts: A Modeling, Loading, Evolving, B Querying, C Recommender, D Graph algorithms.

## 1 A Modeling, Loading, Evolving

### 1.1 A.1 Modeling

We designed these Database Entities and Attributes:

#### Database Entities and Attributes

##### Author

Attributes: AuthorID, Name, PaperCount, Citationcount, Hindex.

##### Paper

Attributes: PaperID, Title, Year, DOI, CitedBy, Authors, Url.

##### Conference

Attributes: ConferenceID, Name.

##### Journal

Attributes: JournalID, Name, ISSN, Url.

##### Volume

Attributes: VolumeID, Volume, Year.

##### Edition

Attributes: EditionID, StartDate, EndDate, City.

##### Reviewer (also an Author)

Attributes: ReviewerID (same as AuthorID).

##### Keyword

Attributes: Keyword.

We also designed the Relationships:

#### Relationships

- WrittenBy - between Author and Paper.
- BelongsTo - between Paper and Edition.
- CitedBy - between Papers.
- ReviewdBy - between Paper and Reviewer.
- AcceptedBy - between Paper and Volume.
- HasAbstract - between Paper and Abstract.

The figure is in Figure 1.

Here we will introduce some justification for the models we design

**Maintainability:** We break the schema into different entities with clear properties, so it is easy to maintain because we can modify one part but not influence other parts.

**Reusability:** The reusability is good because we separate the entities like paper from Proceedings and Volume, so we can easily reuse paper data for both part.

**Non-redundancy:** Because the entities are separated the non-redundancy is good, like one author could have multiple papers, and we don't need to replica the Author-ID information over and over.

**Performance:** Because the relations between entities are usually direct, the performance is good. For instance, we can easily find all papers written by one author by one query.

### 1.2 A.2 Instantiating/Loading

The composition of the dataset includes the sample data provided by the Semantic Scholar API [1] and the artificially generated data. As the sample datasets on Semantic Scholar is

---

\*Universitat Politècnica de Catalunya (UPC), Erasmus Mundus Big Data Management and Analytics (BDMA)

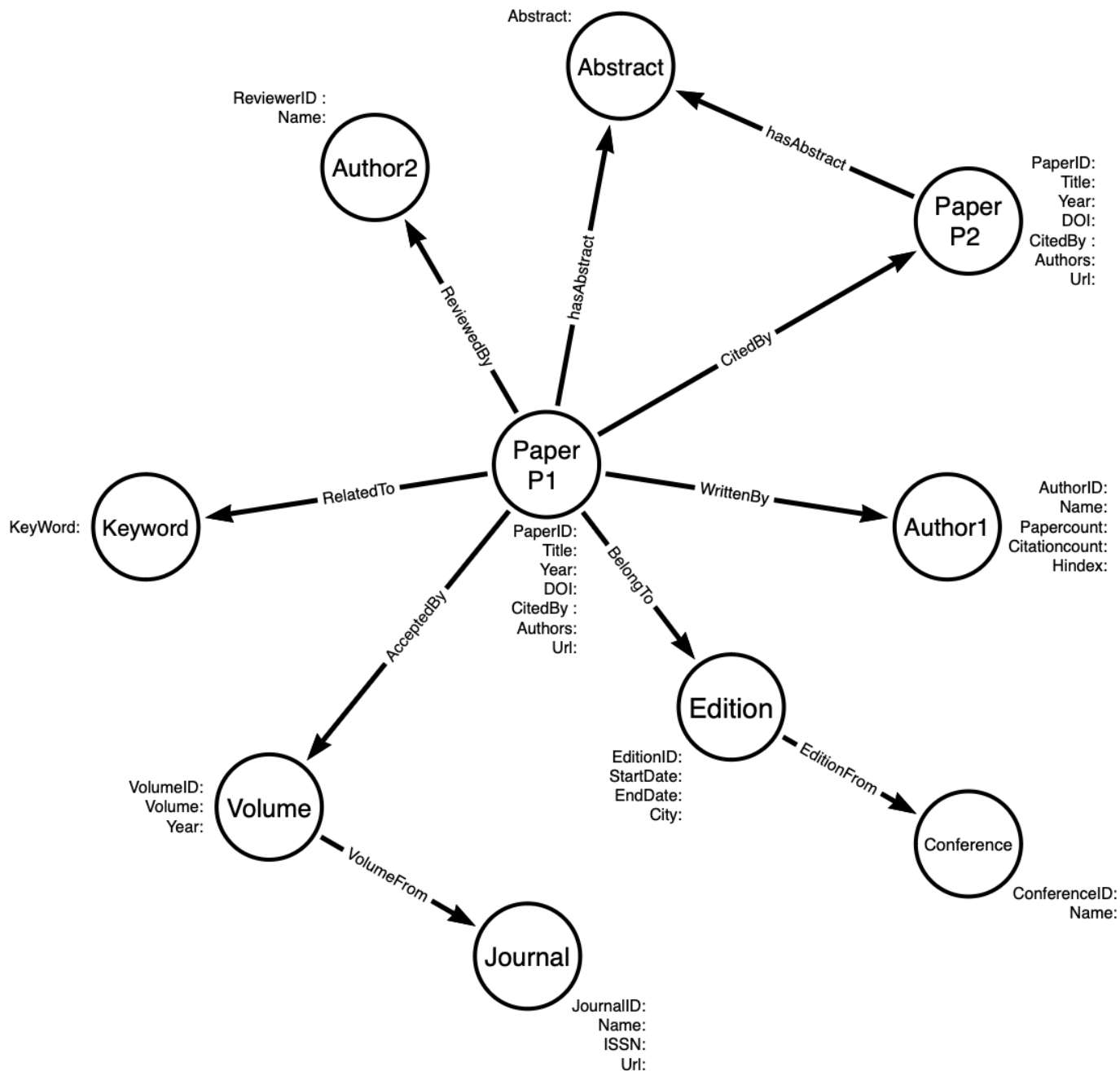


Figure 1: Comprehensive view of the Property Graphs.

very small, and the correlation relationship cannot meet our subsequent query needs, we generated some more data. The generated dataset has the following parts: First, the data extracted from the sample data, such as conferences and journals separated from publication-venues. The second part is some randomly generated data, such as keywords, companies, universities, reviews and so on. The third part is some randomly generated relationships, such as generated\_reviews representing the review results of each paper. The above datasets together constitute the dataset of our study.

Then we load the relevant data into the local neo4j database via python script.

PartA2.YutaoCMinZ.py: Load all datasets into neo4j database.

### 1.3 A.3 Evolving the graph

**Note:** In this section, we primarily completed three modifications: First, we added the reviews section to record the comments of each paper by various reviewers. Second, we added the *accepted\_status* field, initially set to false, and updated it

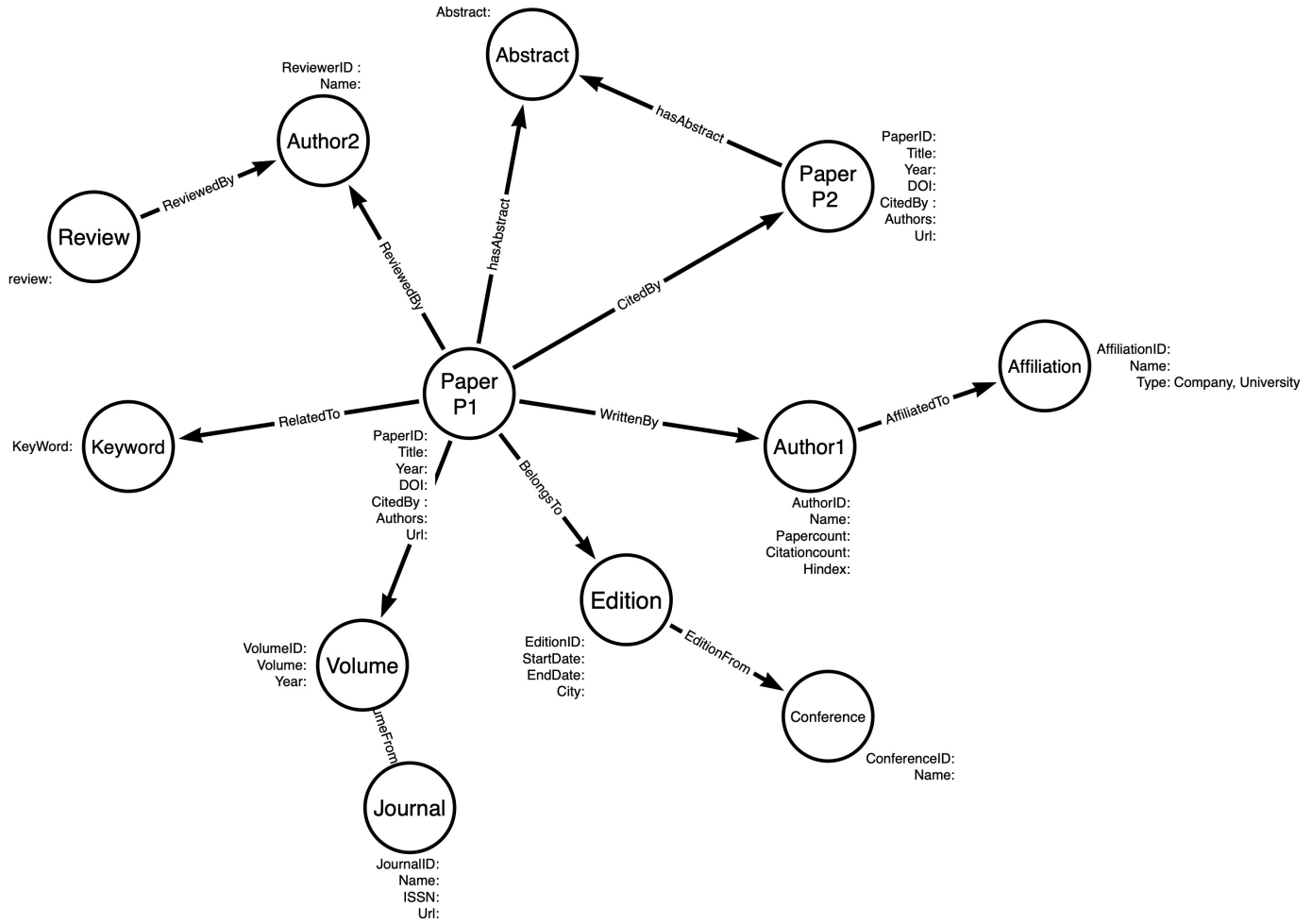


Figure 2: Property Graphs After Involving

based on rules. Third, we added Affiliations information, randomly assigning each author to an institution, either a company or a school.

After involving, the updated figure is in Figure 2. We implemented this process through a Python script.

PartA3.YutaoCMinZ.py: Evolving the graph.

After completing the update, we implemented a query to verify that the update worked.

```
MATCH (paper:Paper {accepted_status: true})
RETURN count(paper) AS accepted_paper_count
```

```
MATCH (paper:Paper {accepted_status: true})
RETURN count(paper) AS accepted_paper_count
```

	accepted_paper_count
1	54

Figure 3: Result of the Query.

## 2 B Querying

### 2.1 Find the top 3 most cited papers of each conference.

The following Cypher query finds the top 3 most cited papers of each conference:

```
1 MATCH (c:Conference) <-[:IS_FROM]-(e:Edition) <-[:
  BELONGS_TO]-(p:Paper)
2 WITH c, p, size([(p)-[:CITED_BY]->(cited) WHERE
  cited:Paper | cited]) AS citations
3 ORDER BY c, citations DESC
4 WITH c, COLLECT({ id: p.id, title: p.title,
  citations: citations }) AS allPapers
5 RETURN c.id AS ConferenceID, c.name AS
  ConferenceName, allPapers[0..3] AS
  TopCitedPapers
```

Listing 1: 3 most cited papers.

Code Design Ideas:

Used a merged WATCH statement to reduce the total steps of the query and simplify the logic. Using SIZE instead of

COUNT, for calculating the number of papers that cite a particular paper, `SIZE((p)-[:CITED_BY]-[:Paper])` is used instead of going through a separate MATCH statement and `count(cb)`. All papers (including their IDs, titles, and citation counts) are collected into a list and sorted by conference and citation count in descending order. The RETURN statement then returns only the top three most-cited papers from each conference. By slicing the collected list of papers directly in the RETURN statement `[0..3]`, this query simplifies the process of extracting the top papers, making the whole query more intuitive and concise.

ConferenceID	ConferenceName
"b1714395-f13b-4720-875e-b9c982709761"	"Mexican International Conference on Computer Science"
"Acta medica et biologica"	"Acta medica et biologica"

Figure 4: Result of the Query.

## 2.2 For each conference find its community.

The following Cypher query finds its community:

```
1 MATCH (author:Author)-[:WRITTEN_BY]->(:Paper)-[:BELONGS_TO]->(edition:Edition)-[:IS_FROM]->(conf:Conference)
2 WITH conf, author, COUNT(DISTINCT edition) AS editions
3 WHERE editions >= 4
4 WITH conf.name AS Conference, COLLECT(author.name) AS CommunityAuthors
5 RETURN Conference, CommunityAuthors, SIZE(CommunityAuthors) AS NumberOfAuthors
```

Listing 2: find community.

Code design logic: Directly uses `COUNT(DISTINCT edition)` to determine the number of different conference editions in which the author participated. By placing the WHERE clause directly after the count of different conference editions, we can more efficiently filter out authors who have participated in at least 4 different editions. Collecting author names directly when the condition is met simplifies the query structure and potentially improves query efficiency.

## 2.3 Find the impact factors of the journals in the graph.

The following is find impact factors of journals:

```
1 MATCH (journal:Journal)-[:VOLUME_FROM]-(vol:Volume)-[:PUBLISHED_IN]-(paper:Paper)
```

conference	authors	activeAuthors
"Jurnal Kesehatan Gigi"	"Glarado Donobhan Preschi"	1
"Acta medica et biologica"	"Nathakki Krobnob"	1
"Zosen-kyokai shi"	"C. Jourdon"	1
"Anuarul Institutului de Cercetari Soco-Umane"	"Luzinete Simoes Minela"	1

Figure 5: Result of the Query.

```
2 WITH journal, toInteger(vol.year) AS publicationYear, COUNT(paper) AS citationsCount
3 OPTIONAL MATCH (journal)-[:VOLUME_FROM]-(volPreviousYear:Volume{year: publicationYear - 1})->[relPublishedInPrevYear: PUBLISHED_IN]-(paper)
4 WITH journal, publicationYear, citationsCount, COUNT(relPublishedInPrevYear) AS publicationsLastYear
5 OPTIONAL MATCH (journal)-[:VOLUME_FROM]-(volTwoYearsAgo:Volume{year: publicationYear - 2})->[relPublishedInTwoYearsAgo: PUBLISHED_IN]-(paper)
6 WITH journal, publicationYear, citationsCount, publicationsLastYear, COUNT(relPublishedInTwoYearsAgo) AS publicationsTwoYearsAgo
7 RETURN journal.id AS JournalIdentifier, publicationYear, citationsCount, publicationsLastYear, publicationsTwoYearsAgo,
8 CASE publicationsLastYear + publicationsTwoYearsAgo WHEN 0 THEN 0.0 ELSE toFloat(citationsCount) / (publicationsLastYear + publicationsTwoYearsAgo) END AS ImpactFactor
9 ORDER BY JournalIdentifier, publicationYear
```

Listing 3: impact factors.

Optimization Logic and Idea Explanation Combine statistical operations: By handling both the number of published papers and the number of citations in the same WITH clause, it reduces the repetitive MATCH and OPTIONAL MATCH operations, making the query more efficient and easy to maintain.

Simplify citation counting: use CITED\_BY relationship directly on the volume related to the journal to count the number of citations, which reduces the length of the path to be traversed and improves the query efficiency.

Reduce hard-coding: Try to avoid using hard-coded year numbers directly in the query, but dynamically determine the year range through calculation and conditional judgment, which makes the code more flexible and adaptable.

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

the following is h index calculation:

JournalIdentifier	publicationYear	citationsCount	publicationsLastYear	public
"000a917-7f08-43f3-836f-43bc8431e615"	2020	1	0	0
"000a917-7f08-43f3-836f-43bc8431e615"	2021	2	1	0
"000a917-7f08-43f3-836f-43bc8431e615"	2022	1	2	1
"000a917-7f08-43f3-836f-43bc8431e615"	2023	2	1	2
"0777b4d8-4819-4a02-b471-3b757b56bec3"	2020	1	0	0

Figure 6: Result of the Query.

```

1 MATCH (a:Author) <-[:WRITTEN_BY]-(p:Paper)
2 OPTIONAL MATCH (p)-[:CITED_BY]->(:Paper)
3 WITH a, p, COUNT(c) AS citations
4 ORDER BY a.name, citations DESC
5 WITH a.name AS authorName, COLLECT(citations) AS
  citationsList
6 UNWIND range(1, SIZE(citationsList)) AS idx
7 WITH authorName, citationsList, idx
8 WHERE citationsList[idx-1] >= idx
9 WITH authorName, COLLECT(idx) AS validHIndex
10 RETURN authorName, REDUCE(s = 0, i IN validHIndex |
  CASE WHEN i > s THEN i ELSE s END) AS h_index
11 ORDER BY authorName

```

Listing 4: h-indexes.

Optimization Logic and Idea Explanation Simplify aggregation steps: By directly calculating the number of citations for each paper in a single WITH statement, the use of intermediate variables is reduced, making the query more straightforward and easier to understand.

Effective use of lists and indexes: the UNWIND and range functions are used to generate the index of the list of citations for each author's paper, and then a conditional expression is used to filter out the index values that satisfy the definition of h-index. This approach reduces complex list operations and makes the query more efficient.

Efficient h-index calculation: Use the REDUCE function to find the largest valid h-index value instead of manually manipulating the list. This not only reduces the complexity of the code, but also improves the execution efficiency.

authorName	h_index
"Abdelaziz Hamed Elbadawy"	3
"Ahmad A. Abo Gabal"	1
"Ambuchi John Justo"	4
"Asiya Kuzembayeva"	3
"B. Jusmianti"	3

Figure 7: Result of the Query.

## 3 C Recommender

### 3.1 Find/define the research communities

the following is the definition of communities:

```

1 CREATE CONSTRAINT communityNameConstraint IF NOT
  EXISTS FOR (c:Community) REQUIRE c.name IS
  UNIQUE;
2
3 MATCH (comm:Community {name: 'database'})-[:r]-()
4 DELETE r;
5
6 MATCH (comm:Community {name: 'database'})
7 MATCH (k:Keyword)
8 WHERE k.keyword IN ['data management', 'indexing', '
  data modeling', 'big data', 'data processing',
  'data storage', 'data querying']
9 CREATE (comm)-[:DEFINED_BY]->(k);

```

Listing 5: research communities.

communityNameConstraint	communityNameConstraint	communityNameConstraint
communityNameConstraint	communityNameConstraint	communityNameConstraint
communityNameConstraint	communityNameConstraint	communityNameConstraint
communityNameConstraint	communityNameConstraint	communityNameConstraint

Figure 8: Result of the Query.

First create a communityNameConstraint that detects if there is a pre-existing community and deletes it if there is one, and then creates and generates new relations based on the given keyword list.

Batch creation of relationships, use of uniqueness constraints, batch deletion of relationships, and conditional filtering of relationship creation significantly improve the efficiency of the entire query.

### 3.2 Find the conferences and journals related to the database community

find the conferences and journals:

```

1 MATCH (paper:Paper)-[:BELONGS_TO|PUBLISHED_IN]->(
  edition)-[:VOLUME_FROM|IS_FROM]->(venue)
2 WITH venue.id AS venueId, COUNT(paper) AS numPapers,
  venue
3 OPTIONAL MATCH (venue)-[:rel:IN_COMMUNITY]->()
4 DELETE rel
5 WITH venueId, numPapers
6
7 MATCH (community:Community)-[:DEFINED_BY]->(keyword)
  <-[:RELATED_TO]-(paper:Paper)-[:PUBLISHED_IN|
  BELONGS_TO]->(edition)-[:VOLUME_FROM|IS_FROM
  ]->(relatedVenue)
8 WHERE relatedVenue.id = venueId

```

```

9 WITH relatedVenue, COUNT(DISTINCT paper) AS
  numPapersWithKeywords, community, numPapers
10 WITH relatedVenue, numPapersWithKeywords, community,
  (toFloat(numPapersWithKeywords) / numPapers)
  AS percentage, numPapers
11 WHERE percentage >= 0.6
12 MERGE (relatedVenue)-[:IN_COMMUNITY]->(community);

```

Listing 6: find conferences and journals community.

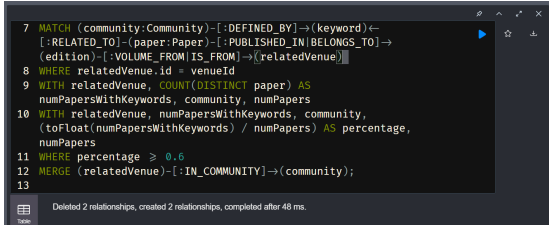


Figure 9: Result of the Query.

Here because our generated dataset is smaller and more parsed than usual, so we changed the threshold from 0.9 to 0.6 to get the result.

A retrieval from paper to EDITION to VENUE is first performed by Match, and then the content already generated by previous experiments is removed. The main text part performs a search from community to paper to avenue, and decides whether it belongs to the community by calculating the ratio of the number of articles that meet the keyword requirements in a related venue to the number of all articles.

Batch creation of relationships, conditional filtering, and batch deletion of relationships significantly improves the efficiency of the entire query.

### 3.3 Identify the top papers of these conferences/journals

find the top journals:

```

1 MATCH (paper:Paper)-[:PUBLISHED_IN|BELONGS_TO]->(
  publication)-[:VOLUME_FROM|IS_FROM]->(venue)-[:
  IN_COMMUNITY]->(community:Community {name: '
  database'})
2 WITH paper, SIZE([(paper)-[:CITED_BY]->(:Paper) |
  1]) AS citations
3 ORDER BY citations DESC
4 LIMIT 100
5 SET paper.is_database_com_top = true
6 RETURN paper.title, citations;

```

Listing 7: find top journals.

The highest ranked papers are selected based on Pagerank, first filtered by Match to find papers that meet the community requirements, sorted based on citations, and finally, a property

is\_database\_com\_top is added to the paper label, setting the property to true.

Conditional filtering using pattern matching, using aggregation functions, and limiting the result set size significantly improves the efficiency of the entire query.

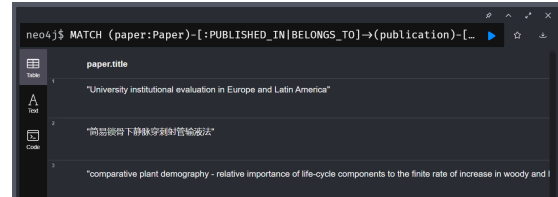


Figure 10: Result of the Query.

### 3.4 Find a potentially good match to review database papers and identify gurus

find the potential good match:

```

1 MATCH (author:Author)-[:WRITTEN_BY]->(paper:Paper {
  is_database_com_top: true})
2 SET author.potential_database_com_rev = true
3 WITH author
4 MATCH (author)-[:WRITTEN_BY]->(topPaper:Paper {
  is_database_com_top: true})
5 WITH author, COUNT(topPaper) AS papersCount
6 WHERE papersCount >= 2
7 SET author.database_com_guru = true
8 RETURN author.name, papersCount, author.
  potential_database_com_rev, author.
  database_com_guru;

```

Listing 8: good match.

Use Match to find the authors of the top 100 articles, add the potential\_database\_com\_rev property to them and set it to true. when the author's potential\_database\_com\_rev is true and papersCount<sub>i</sub>=2, the we consider him to be a suitable reviewer.

Chaining processing nodes, conditional filtering significantly improves the efficiency of the entire query.

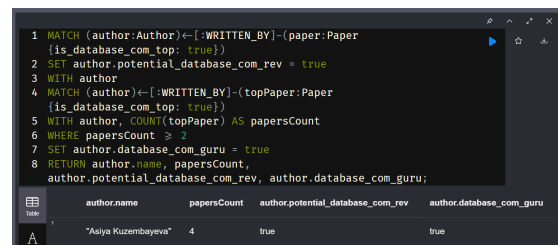


Figure 11: Result of the Query.

## 4 D Graph algorithms

### 4.1 Closeness Centrality

Closeness centrality is a way to detect nodes that can propagate information very efficiently through the graph. Its mathematical algorithm is:  $\text{normalized closeness centrality}(u) = (\text{number of nodes} - 1) / \text{sum}(\text{distance from } u \text{ to all other nodes})$  is used in the study of organizational networks, where individuals with high closeness centrality are in an advantageous position to control and gain access to important information and resources within the organization. From the results we can see which nodes are closer to the center of the whole graph and which nodes are farther away from the graph center[2].

First let's create the graph projection:

```
1 CALL gds.graph.project(
2   'Big_Graph1',
3   ['Author', 'Paper', 'Edition', 'Conference'],
4   {
5     RELATIONSHIP_TYPE_1: {
6       type: 'BELONGS_TO',
7       orientation: 'REVERSE'
8     },
9     RELATIONSHIP_TYPE_2: {
10      type: 'WRITTEN_BY',
11      orientation: 'NATURAL'
12    },
13    RELATIONSHIP_TYPE_3: {
14      type: 'IS_FROM',
15      orientation: 'NATURAL'
16    },
17    RELATIONSHIP_TYPE_4: {
18      type: 'REVIEWED_BY',
19      orientation: 'NATURAL'
20    },
21    RELATIONSHIP_TYPE5: {
22      type: 'CITED_BY',
23      orientation: 'NATURAL'
24    }
25  }
26 )
```

Listing 9: graph projection.

The Closeness Centrality code:

```
1 CALL gds.closeness.stream('Big_Graph1')
2 YIELD nodeId, score
3 RETURN gds.util.asNode(nodeId).id AS id, score
4 ORDER BY score DESC
```

Listing 10: Closeness Centrality.

The results available at Figure 12:

In this subsection we link more labels, we introduce the four labels 'Author', 'Paper', 'Edition', 'Conference', and also the five relations BELONGS\_TO, WRITTEN\_BY, IS\_FROM, REVIEWED\_BY, and CITED\_BY, so that we in fact create a PRO-



id	score
"cfcece5a-810b-4ec1-af5b-f370420c70d"	1.0
"95376550-8cb6-4de1-9eaa-b434cf1265ff"	1.0
36833670	0.4295774647887324
221873644	0.427170883473389
220192299	0.4206896551724138

Figure 12: Result of the Query.

JECTION centered on the PAPER, with other nodes expanding to the left and right.

However, in the final result we observe that the Edition is in fact the center of the graph, but this is indeed logical because we only have several Editions, and the rest of the hundreds of Authors are all around the Edition, so it makes sense that the Edition is closer to the other nodes.

Using streaming processing, the delayed loading of node attributes significantly improves the efficiency of the entire query.

### 4.2 Node Similarity

The Graph Projection Code:

```
1 CALL gds.graph.project(
2   'Paper_Keyword',
3   ['Keyword', 'Paper'],
4   {
5     WRITTEN_BY: {
6       type: 'RELATED_TO',
7       orientation: 'NATURAL'
8     }
9   }
10 )
```

Listing 11: Paper<sub>Keyword</sub>Projection.

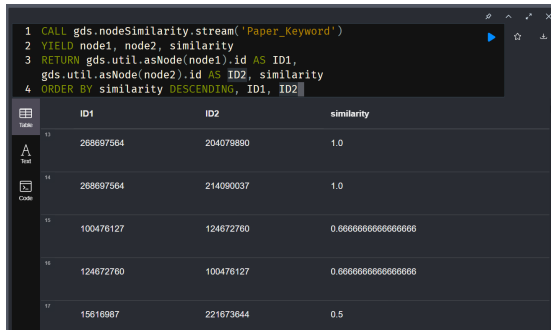
The Graph Projection code:

```
1 CALL gds.nodeSimilarity.stream('Paper_Keyword')
2 YIELD node1, node2, similarity
3 RETURN gds.util.asNode(node1).id AS ID1, gds.util.
4   asNode(node2).id AS ID2, similarity
5 ORDER BY similarity DESCENDING, ID1, ID2
```

Listing 12: The Node Similarity.

The results available at Figure 13:

The node similarity algorithm compares a set of nodes based on the nodes they are connected to. Two nodes are considered similar if they share many of the same neighbors. In this subsection, we have chosen Paper and Keyword label for PROJECTION and RELATED\_TO for RELATION, so we are looking for similarity of all articles by comparing the similarity of all articles their KEYWORDS [2].



```

1 CALL gds.nodeSimilarity.stream('Paper_keyword')
2 YIELD node1, node2, similarity
3 RETURN gds.util.asNode(node1).id AS ID1,
  gds.util.asNode(node2).id AS ID2, similarity
4 ORDER BY similarity DESCENDING, ID1, ID2

```

	ID1	ID2	similarity
13	268697564	204079890	1.0
14	268697564	214090037	1.0
15	100476127	124672760	0.6666666666666666
16	124672760	100476127	0.6666666666666666
17	15616987	221673644	0.5

Figure 13: Result of the query.

Using streaming processing, the delayed loading of node attributes significantly improves the efficiency of the entire query.

## References

- [1] Allen Institute for AI. S2 Folks Repository, current.
- [2] Neo4j. Graph Data Science Algorithms Documentation, current.