Semantic Data Management

Project 1 – DBLP in Neo4j

Sebastian Paglia
MSc Data Science
`sebastian.paglia@estudiantat.upc.edu`

Emmanuel Werr
MSc Data Science
`emmanuel.werr@estudiantat.upc.edu`

Facultat d'Informàtica de Barcelona
Master of Data Science

March 16, 2023

Contents

1 Part A. Modeling, Loading, Evolving

1.1 A.1 Modeling

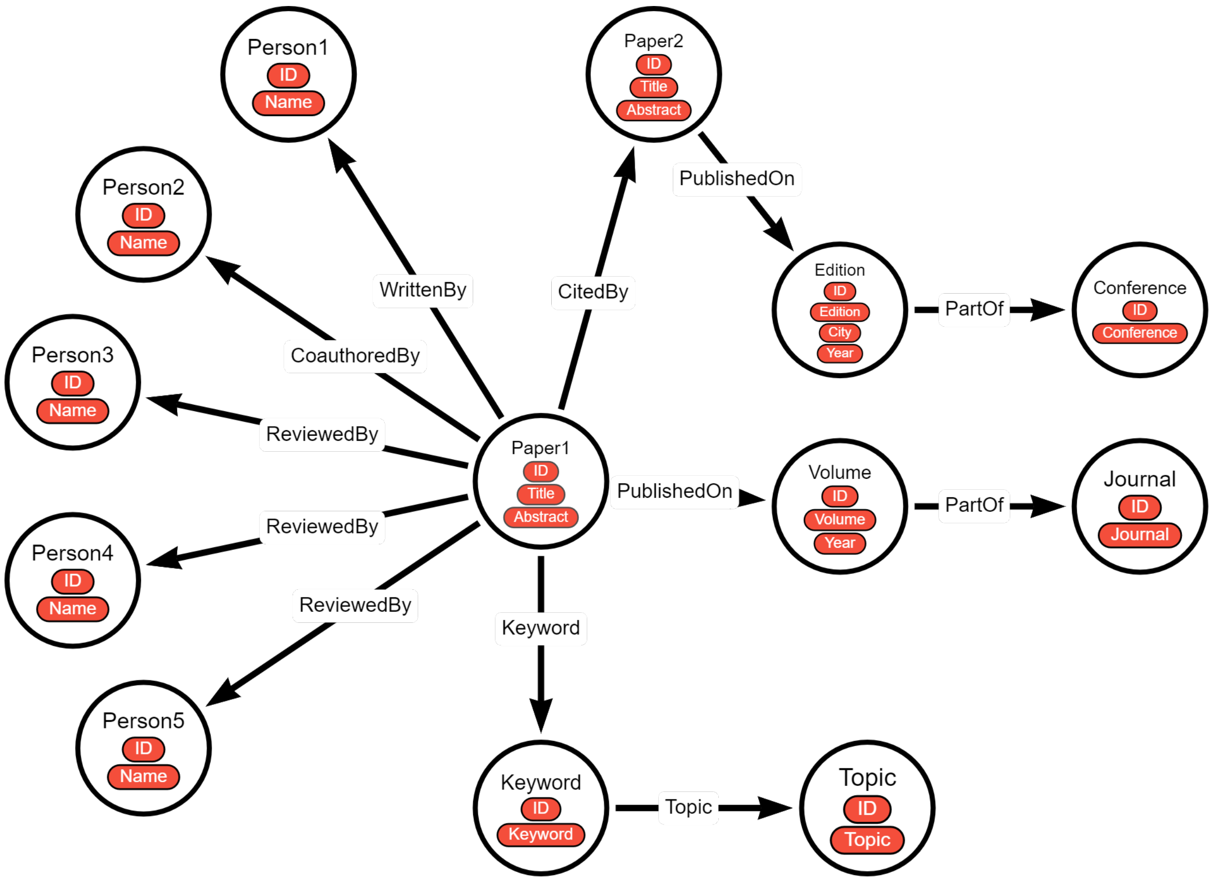


Figure 1: GraphA1

The former (fig. 1) was designed using 'Paper' as the main node, from which many edges emerge connecting it to other nodes, representing the relations and constraints specified in the task.

Then, a certain paper can be written by a person, coauthored by another person and reviewed by three different people. Also, the paper must include keywords that corresponds to a topic. The paper can be cited by another paper and published on a volume or edition which are part of a journal or conference respectively. We decided to set volume and edition as nodes because specifically in the third query it was beneficial for us to keep it that way, improving maintenance and flexibility, and allowing us to store those people attending to each of them, making the query much easier.

In summary, we have as nodes: "Paper", "Person", "Keyword", "Topic", "Edition", "Conference", "Volume", and "Journal"; and as edges: "written_by", "coauthored_by", "reviewed_by", "is_keyword", "Topic", "cited_by", "published_on" and "part_of.

1.2 A.2 Instantiating/Loading

To import the data into Neo4j, first, it was necessary to convert the XML files downloaded from the web-page into CSV files. We also refined our data selection process, ensuring that we only downloaded the information necessary to answer our queries. Additionally, we manually curated some of the data to ensure that it would produce meaningful results, considering the nodes and relationships that we had established earlier.

For 'authors' dataset, we selected 23 people from the original file, with their IDs and names, which will be used to feed 'Person' node. In addition, 'papers' data was created synthetically with 5 variables (id, title, abstract, keywords and topic) and 169 rows, including the words specified in the task for the queries but also another topics to maintain certain reality in the project. The 'conference' data was obtained from the original file, keeping only 3 of them (7 editions for each conference), with their respectively IDs, names, edition, venue, and time (which was divided into season and year). Finally, journals was also obtained from the original files keeping 3 entries (7 volumes for each journal) with id, name, volume and year.

The relations were randomly created using python scripts, bearing in mind the constraints each relation should have. The author, coauthor and each reviewer must be a different person. A paper can be cited by a different paper and must be published on an edition and a volume. Moreover, a paper can have many different keywords and each of them indicates a certain topic.

1.3 A.3 Evolving the graph

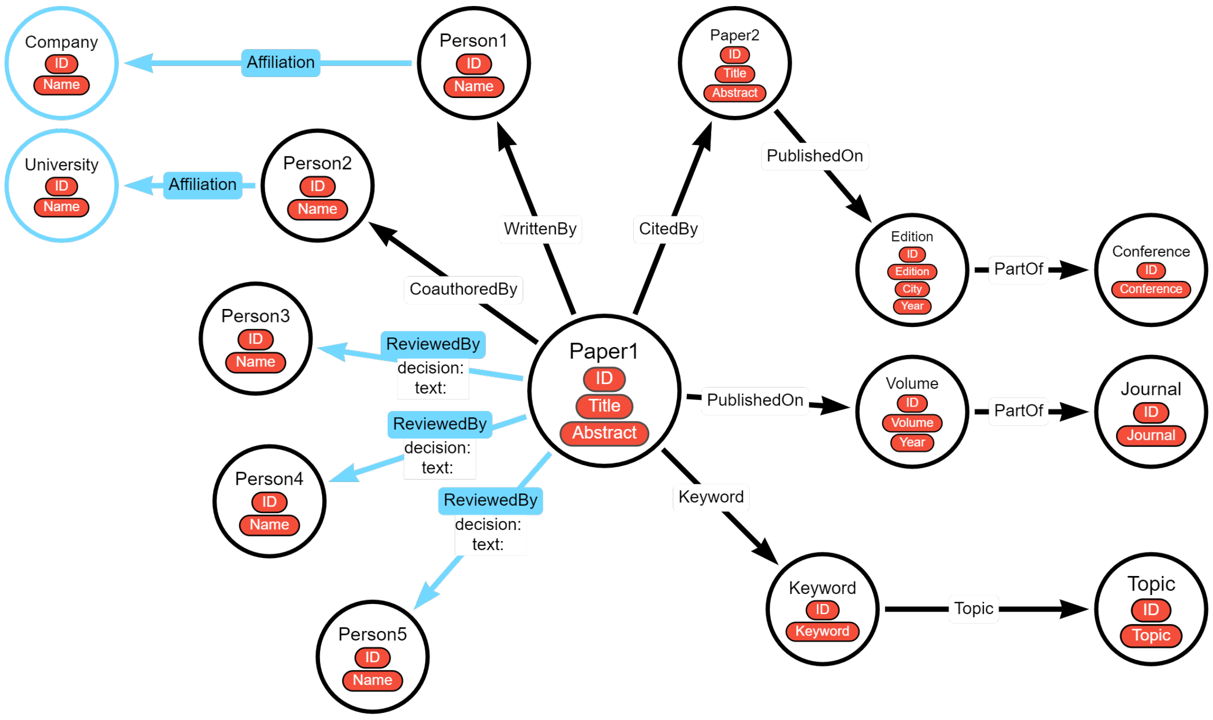


Figure 2: GraphA3

In order to accurately represent the new characteristics described in this section, we have introduced new nodes and edges to the graph. Specifically, we have represented institutions as new nodes that emerge from different people, connected by an edge called Affiliation. Both Company and University nodes have an ID and Name associated with them. We made this decision because it would be beneficial for us to have them as separate nodes in case we needed to execute specific query on authors where the organization type was relevant.

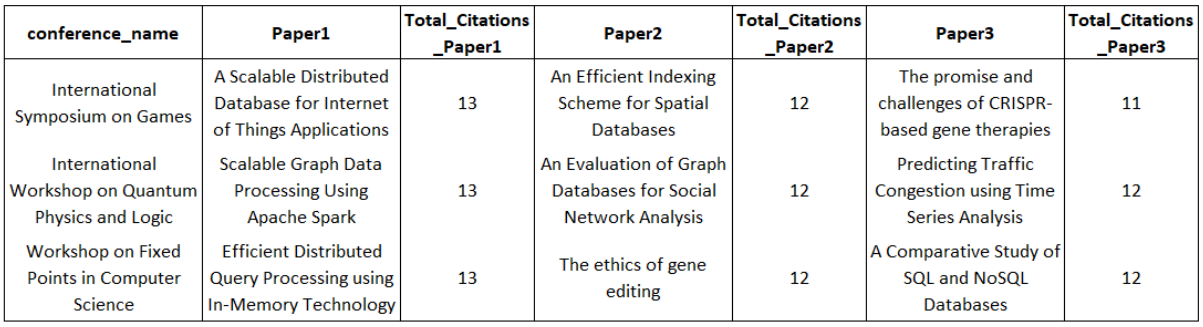
To implement this, we randomly selected four universities and five companies and created a CSV file for each of them with their names and IDs. We then assigned them to every person in the authors table. Additionally, we created a CSV file with the ID relation of both person_TO_university and person_TO_company.

To represent the decision and reviews (text) of each reviewer, we did not need to create a new node or table. Instead, we added the corresponding decision and text of each review made by different people into the existing edge (ReviewedBy). We made this decision to simplify the complexity of the graph, avoiding the creation of new nodes that would not add semantics to the existing model and would be redundant. Query performance remains unchanged, since to get all review decisions for a specific paper, we would anyways have to read every outgoing review of the paper.

2 Part B. Querying

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

```
MATCH (p2:Paper)<-[:CitedBy]-(p1:Paper)-[:PublishedOn]->
    (e1:Edition)-[:PartOf]->(c:Conference)
WITH c.ID as conference_id, c.Conference as conference_name,
    p1.Title as paper, count(*) as citations
ORDER BY conference_id, citations DESC
WITH conference_id, conference_name, collect([paper, citations]) as result
RETURN conference_id, conference_name,
    result[0][0] as Paper1, result[0][1] as Total_Citations_Paper1,
    result[1][0] as Paper2, result[1][1] as Total_Citations_Paper2,
    result[2][0] as Paper3, result[2][1] as Total_Citations_Paper3;
```

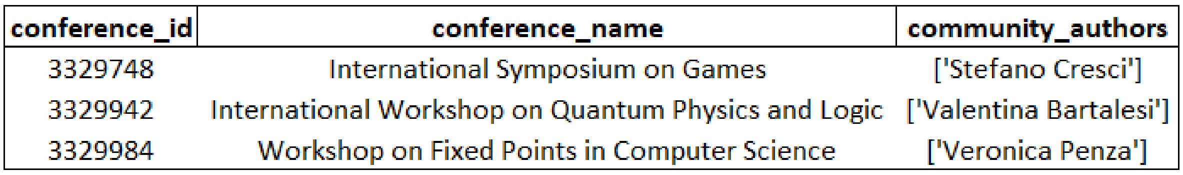


conference_name	Paper1	Total_Citations_Paper1	Paper2	Total_Citations_Paper2	Paper3	Total_Citations_Paper3
International Symposium on Games	A Scalable Distributed Database for Internet of Things Applications	13	An Efficient Indexing Scheme for Spatial Databases	12	The promise and challenges of CRISPR-based gene therapies	11
International Workshop on Quantum Physics and Logic	Scalable Graph Data Processing Using Apache Spark	13	An Evaluation of Graph Databases for Social Network Analysis	12	Predicting Traffic Congestion using Time Series Analysis	12
Workshop on Fixed Points in Computer Science	Efficient Distributed Query Processing using In-Memory Technology	13	The ethics of gene editing	12	A Comparative Study of SQL and NoSQL Databases	12

Figure 3: Result Part B.1

2.2 B.2 - For each conference find its community:

```
MATCH (p:Person)<-[:WritenBy]-(p1:Paper)-[:PublishedOn]->
    (e1:Edition)-[:PartOf]->(c:Conference)
WITH c.ID as conference_id, c.Conference as conference_name, p.Name as author,
    count(DISTINCT e1.Edition) as publications
ORDER BY conference_id, publications DESC
WHERE publications > 3
WITH conference_id, conference_name, collect(author) as community_authors
RETURN conference_id, conference_name, community_authors;
```

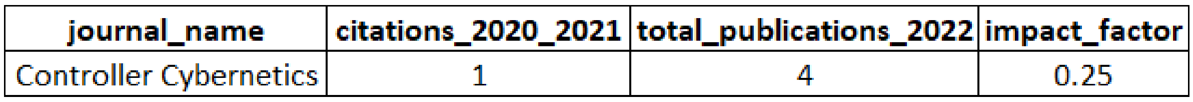


conference_id	conference_name	community_authors
3329748	International Symposium on Games	['Stefano Cresci']
3329942	International Workshop on Quantum Physics and Logic	['Valentina Bartalesi']
3329984	Workshop on Fixed Points in Computer Science	['Veronica Penza']

Figure 4: Result Part B.2

2.3 B.3 - Find the impact factors of the journals in your graph

```
MATCH (j:Journal)
CALL{
    with j
    MATCH (p2:Paper)-[:CitedBy]->(p1:Paper)-[:PublishedOn]->
        (v:Volume{Year: ’2022’})-[:PartOf]->(j)
    WHERE EXISTS {
    MATCH(j)<-[:PartOf]-(v1:Volume)<-[:PublishedOn]-(p2)
    WHERE v1.Year IN ["2021","2020"]}
    REturn j.Journal as journal_name, p2.ID as paper2_id
}
CALL{
    with j
    MATCH(j)<-[:PartOf]-(v:Volume)<-[:PublishedOn]-(p1:Paper)
    WHERE v.Year IN ["2021","2020"]
    WITH j.Journal as journal_name, p1.Title as paper,
        count(*) as num_publications
    WITH sum(num_publications) AS total_num_publications
    WHERE total_num_publications > 0
    RETURN total_num_publications;
}
WITH journal_name, count(paper2_id) as citations_2020_2021,
    total_num_publications as total_publications_2022
RETURN journal_name, citations_2020_2021, total_publications_2022,
    round( (1.0*citations_2020_2021) / (total_publications_2022), 2)
    as impact_factor;
```

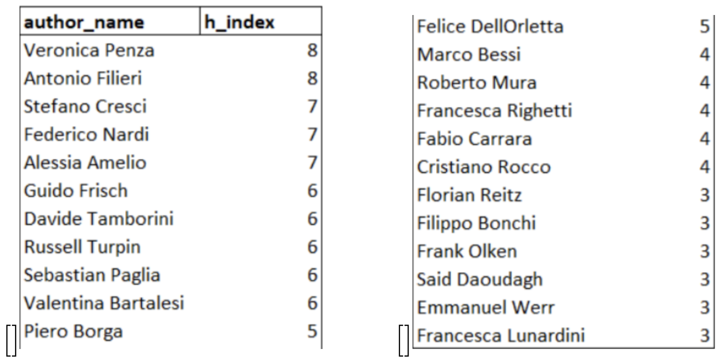


journal_name	citations_2020_2021	total_publications_2022	impact_factor
Controller Cybernetics	1	4	0.25

Figure 5: Result Part B.3

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

```
MATCH (pe:Person)<-[:WritenBy]-(p1:Paper)-[:CitedBy]->(p2:Paper)
WITH pe.Name as author_name, p1.Title as Title, count(*) as NumCites
ORDER BY NumCites desc
WITH author_name, collect(NumCites) as list_NumCites
WITH author_name, [ x IN range(1,size(list_NumCites))
    WHERE x <= list_NumCites[x-1] | [list_NumCites[x-1],x] ]
    as list_hindex
RETURN author_name, list_hindex[-1][1] as h_index
ORDER BY h_index desc;
```



author_name	h_index
Veronica Penza	8
Antonio Filieri	8
Stefano Cresci	7
Federico Nardi	7
Alessia Amelio	7
Guido Frisch	6
Davide Tamborini	6
Russell Turpin	6
Sebastian Paglia	6
Valentina Bartalesi	6
Piero Borga	5
Felice DellOrletta	5
Marco Bessi	4
Roberto Mura	4
Francesca Righetti	4
Fabio Carrara	4
Cristiano Rocco	4
Florian Reitz	3
Filippo Bonchi	3
Frank Olken	3
Said Daoudagh	3
Emmanuel Werr	3
Francesca Lunardini	3

Figure 6: PartB - Query 4 Output

3 Part C. Recommendation System

3.1 Defining the Database Research Community

The first step is to create a new node for the database community and create relationships between the provided database-related keywords. The two Cypher queries below show the desired functionality.

```
Create_CommunityDB = '''CREATE (cdb:Community {name:'CommunityDB'});'''

Match_Keyword_CommunityDB = '''
   MATCH (cdb:Community {name:'CommunityDB'})
   MATCH (k:Keyword)
   WHERE k.Keyword in ['data management', 'indexing', 'data modeling',
    'big data', 'data processing', 'data storage' , 'data querying']
   MERGE (k)-[:RelatedTo]->(cdb);'''
```

3.2 Conferences/Journals Related to the Database Community

The second step is to find the conferences and journals related to the database. We use the provided rule by the lab that specifies that these conferences/journals are those in which at least 90% of the published papers containg our database-related keywords. With the data we generated, this high threshold provided no conferences part of the comunity, so we have lowered our threshold to 15% which reduces the total number of both conferences and journals in the community from 3 to 2, and allows us to test proper functionality of our Cypher queries, which are provided below along with terminal output of the response.

Conference:

```
Match_Conference_CommunityDB = '''
    MATCH (c:Conference)
    CALL{WITH c
        MATCH (p1:Paper)-[:PublishedOn]->(e1:Edition)-[:PartOf]->(c)
        WITH c, c.Conference as conference_name, count(*) as total_papers
        RETURN total_papers}
    MATCH (c)<-[:PartOf]-(e1:Edition)<-[:PublishedOn]-(p1:Paper)-
        [:HasKeyword]->(k:Keyword)-[:RelatedTo]->(cdb:Community)
    WITH c, cdb, c.Conference as conference_name, total_papers,
        count(distinct p1.ID) as total_papers_CommunityDB
    WITH c, cdb, conference_name, total_papers, total_papers_CommunityDB,
        ( (1.0*total_papers_CommunityDB) / (total_papers) ) as threshold
    WHERE threshold >= 0.15
    MERGE (c)-[:RelatedTo]->(cdb)
    RETURN conference_name, total_papers, total_papers_CommunityDB, round(threshold,
```

Journal:

```
Match_Journal_CommunityDB = '''
    MATCH (j:Journal)
    CALL{WITH j
        MATCH (p1:Paper)-[:PublishedOn]->(v1:Volume)-[:PartOf]->(j)
        WITH j, j.Journal as journal_name, count(*) as total_papers
        RETURN total_papers}
    MATCH (j)<-[:PartOf]-(v1:Volume)<-[:PublishedOn]-(p1:Paper)-
    [:HasKeyword]->(kw:Keyword)-[:RelatedTo]->(cdb:Community)
    WITH j, cdb, j.Journal as journal_name, total_papers,
    count(distinct p1.ID) as total_papers_CommunityDB
    WITH j, cdb, journal_name, total_papers, total_papers_CommunityDB,
    ((1.0*total_papers_CommunityDB) / (total_papers)) as threshold
    WHERE threshold >= 0.15
    MERGE (j)-[:RelatedTo]->(cdb)
    RETURN journal_name, total_papers, total_papers_CommunityDB, round(threshold, 2)
```

3.3 Identifying their Top Papers

Create subgraph of conference papers in CommunityDB

```
init_ConfPapers_CommunityDB = '''
    CALL gds.graph.project.cypher(
        'ConfPapers_CommunityDB',
        'MATCH (p:Paper) RETURN DISTINCT id(p) As id',
        'MATCH (cdb1:Community)<-[:RelatedTo]-(c1:Conference)<-
        [:PartOf]-(e1:Edition)<-[:PublishedOn]-(p1:Paper)-[:CitedBy]->
        (p2:Paper)-[:PublishedOn]->(e2:Edition)-[:PartOf]->(c2:Conference)-
        [:RelatedTo]->(cdb2:Community) WHERE cdb1.name="CommunityDB" and
        cdb2.name="CommunityDB" RETURN id(p1) as source, id(p2) as target');'''

Top_ConfPapers_CommunityDB = '''
    CALL gds.pageRank.stream('ConfPapers_CommunityDB',
    {maxIterations: 50, dampingFactor: 0.85})
        YIELD nodeId, score
        WITH gds.util.asNode(nodeId) as node, nodeId as paper_id,
        gds.util.asNode(nodeId).Title AS paper_name, score as page_rank
        MATCH (cdb:Community {name:'CommunityDB'})
        MERGE (node)-[:RelatedTo]->(cdb)
        RETURN paper_id, paper_name, page_rank
        ORDER BY page_rank DESC
        LIMIT 100;'''
```

Create subgraph of conference papers in CommunityDB

```
init_JournPapers_CommunityDB = '''
    CALL gds.graph.project.cypher(
        'JournPapers_CommunityDB',
        'MATCH (p:Paper) RETURN DISTINCT id(p) As id',
        'MATCH (cdb1:Community)<-[:RelatedTo]-(j1:Journal)<-[:PartOf]-
        (v1:Volume)<-[:PublishedOn]-(p1:Paper)-[:CitedBy]->(p2:Paper)-
        [:PublishedOn]->(v2:Volume)-[:PartOf]->(j2:Journal)-[:RelatedTo]->
        (cdb2:Community) WHERE cdb1.name="CommunityDB" and
        cdb2.name="CommunityDB" RETURN id(p1) as source, id(p2) as target');'''

Top_JournPapers_CommunityDB = '''
    CALL gds.pageRank.stream('JournPapers_CommunityDB',
    {maxIterations: 50, dampingFactor: 0.85})
        YIELD nodeId, score
        WITH gds.util.asNode(nodeId) as node, nodeId as paper_id,
        gds.util.asNode(nodeId).Title AS paper_name, score as page_rank
        MATCH (cdb:Community {name:'CommunityDB'})
        MERGE (node)-[:RelatedTo]->(cdb)
        RETURN paper_id, paper_name, page_rank
        ORDER BY page_rank DESC
        LIMIT 100;'''
```

3.4 Identifying Gurus

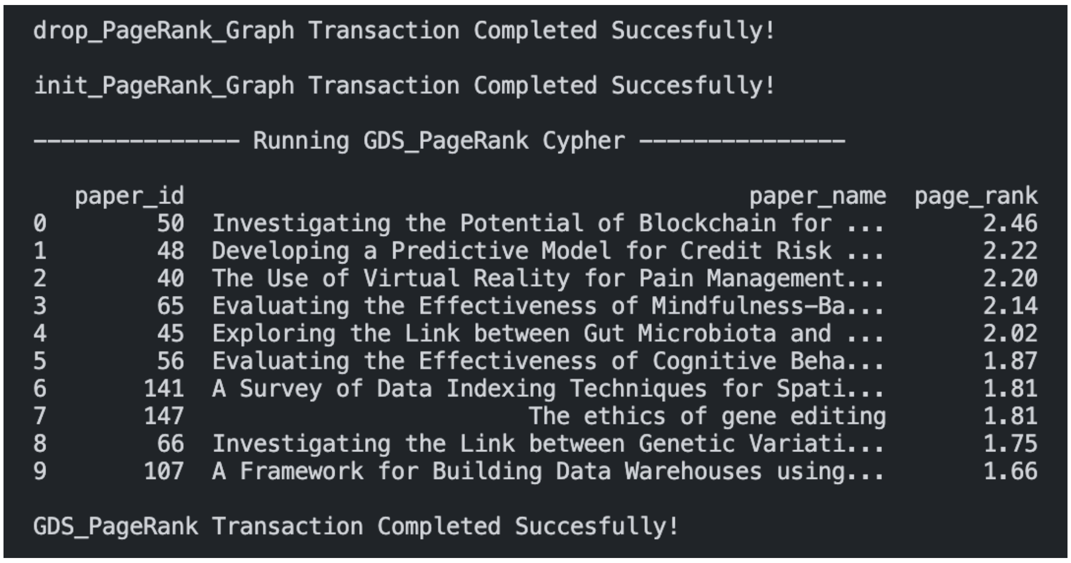
In order to Identify Gurus, the procedure is quite similar to identifying related conferences and journals above. We need to include a node query Match of person (authors) as well as a relation query of author-¿paper1-¿journal/conference-¿paper2-¿journal/conference-¿author. Then we assign the value of guru to those authors who pass this query, and normal reviewers to those who dont. The Cypher query that provides this functionality is shown below.

4 Part D. Graph Algorithms

4.1 D.1 - Page Rank

We considered the link between papers as the number of citations that each paper has, setting the parameters of the algorithm at 50 of maximum iterations and 0.85 as the threshold of damping factor. We have obtained expected results, where we have the full list of all papers as well as their pagerank when taking into account only their citation relations. This made the most sense.

```
CALL gds.graph.project('PartD_PageRank', 'Paper', 'CitedBy');
CALL gds.pageRank.stream('PartD_PageRank', {maxIterations: 50,
    dampingFactor: 0.85})
YIELD nodeId, score
RETURN nodeId as paper_id, gds.util.asNode(nodeId).Title AS paper_name,
    round(score, 2) as page_rank
ORDER BY score DESC LIMIT 10;
```



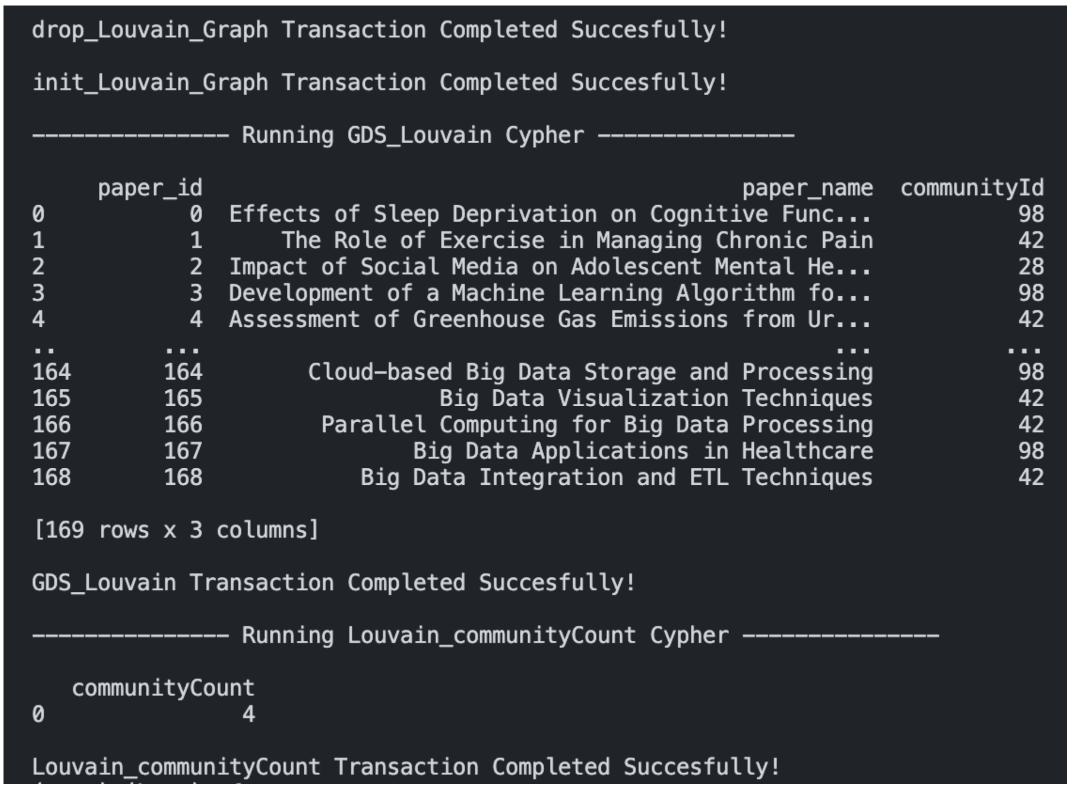
```
drop_PageRank_Graph Transaction Completed Succesfully!

init_PageRank_Graph Transaction Completed Succesfully!

--------------- Running GDS_PageRank Cypher ---------------

   paper_id                                         paper_name  page_rank
0        50  Investigating the Potential of Blockchain for ...       2.46
1        48  Developing a Predictive Model for Credit Risk ...       2.22
2        40  The Use of Virtual Reality for Pain Management...       2.20
3        65  Evaluating the Effectiveness of Mindfulness-Ba...       2.14
4        45  Exploring the Link between Gut Microbiota and ...       2.02
5        56  Evaluating the Effectiveness of Cognitive Beha...       1.87
6       141  A Survey of Data Indexing Techniques for Spati...       1.81
7       147                        The ethics of gene editing       1.81
8        66  Investigating the Link between Genetic Variati...       1.75
9       107  A Framework for Building Data Warehouses using...       1.66

GDS_PageRank Transaction Completed Succesfully!
```

Figure 7: Result Part D Page Rank

4.2 D.2 - Louvain Community Detection

We decided to apply this algorithm because it is one of the most popular community detection algorithms and it seemed like a fun one to experiment with. We used the same type of subgraph as above with papers and citations and obtained 4 different distinct communities

```
CALL gds.graph.project('PartD_Louvain', 'Paper', 'CitedBy');
CALL gds.louvain.stream('PartD_Louvain')
YIELD nodeId, communityId, intermediateCommunityIds
RETURN nodeId as paper_id, gds.util.asNode(nodeId).Title AS paper_name,
    communityId
ORDER BY paper_id ASC;
CALL gds.louvain.stats('PartD_Louvain')
YIELD communityCount;
```



```
drop_Louvain_Graph Transaction Completed Succesfully!

init_Louvain_Graph Transaction Completed Succesfully!

--------------- Running GDS_Louvain Cypher ---------------

     paper_id                                       paper_name  communityId
0           0  Effects of Sleep Deprivation on Cognitive Func...           98
1           1      The Role of Exercise in Managing Chronic Pain           42
2           2  Impact of Social Media on Adolescent Mental He...           28
3           3  Development of a Machine Learning Algorithm fo...           98
4           4  Assessment of Greenhouse Gas Emissions from Ur...           42
..        ...                                              ...          ...
164       164       Cloud-based Big Data Storage and Processing           98
165       165                 Big Data Visualization Techniques           42
166       166        Parallel Computing for Big Data Processing           42
167       167              Big Data Applications in Healthcare           98
168       168           Big Data Integration and ETL Techniques           42

[169 rows x 3 columns]

GDS_Louvain Transaction Completed Succesfully!

--------------- Running Louvain_communityCount Cypher ---------------

   communityCount
0               4

Louvain_communityCount Transaction Completed Succesfully!
```

Figure 8: Result Part D Louvain