MINING BIG DATASETS

Neo4j Graph Database Assignment

Supervisor Professors: Y.Kotidis, I.Filippidou

JULY 2, 2023

DIMITRIOS MATSANGANIS, f2822212
FOTEINI NEFELI NOUSKALI, f2822213

MSc. Business Analytics FT 2022-2023

Contents

Table of Figures

1. Case Description - Solution Plan

The dataset provided for this assignment is a subset of the high energy physics theory citation network. It consists of information about authors, articles, journals, and citations between articles. The dataset comprises three files: ArticleNodes.csv, AuthorNodes.csv, and Citations.csv.

- The **ArticleNodes.csv** file contains details about the articles, including their unique identifiers (IDs), titles, publishing years, the journals where they were published, and abstracts.

- The **AuthorNodes.csv** file contains the article IDs from the previous file along with the names of the authors associated with each article.

- The **Citations.csv** file provides information about the citations between articles. It includes pairs of article IDs, where one article is cited by another.

To create a graph model from this dataset, several relationships will be established. First, the ArticleNodes.csv file will be used to create nodes for articles and journals. The properties associated with each article, such as title, year, and abstract, will be included in the corresponding nodes.

Next, the AuthorNodes.csv file will be used to create nodes for authors. The relationship between articles and authors will be represented using the relationship named WRITTEN_BY. Each author node will be connected to the article(s) they have written using the relationship named WRITTEN_BY.

To establish the connection between articles and authors, the PUBLISHED_IN relationship will be created. This relationship will link the Article and Author nodes together, representing that an article is published in a specific journal and authored by certain individual(s).

Last but not least, the Citations.csv file will be used to define the relationship between articles based on citations. The relationship will be named CITES and will connect the citing article with the cited article. By implementing these relationships, the graph model will represent the connections between articles, authors, and journals (representing journal venues etc.), as well as the citation relationships between articles.

2. Creation of Graph Model

Neo4j is a graph database management system that allows for the storage, retrieval, and manipulation of data in the form of a graph. Unlike traditional relational databases, which are based on tables and rows, Neo4j organizes data as nodes, relationships, and properties, providing a more intuitive and flexible way to model and query complex relationships.

The creation of a graph model involves structuring and organizing the data into nodes and relationships to represent the entities and connections within a dataset. In the context of the given high energy physics theory citation network dataset, we will now proceed with the creation of the graph model in Neo4j.

2.1. Import the data into Neo4j

To begin, we will import the dataset files, namely ArticleNodes.csv, AuthorNodes.csv, and Citations.csv, into a local Neo4j database. This will provide us with the necessary data to create the nodes and relationships of our graph model.

To be more precise, we will import the provided dataset and create the graph model, we will be using Neo4j Desktop. **Neo4j Desktop** is a tool that provides a user-friendly interface for managing Neo4j databases and projects. It allows us to create a new project, add a local database, and perform various operations such as importing data, running queries, and visualizing the graph.

In the context of our assignment, we will create a new project **named Mining Big Datasets Assignment 2** in Neo4j Desktop (*See Figure 1*).

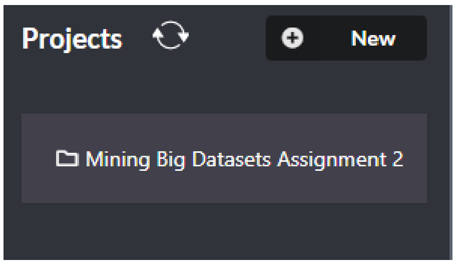
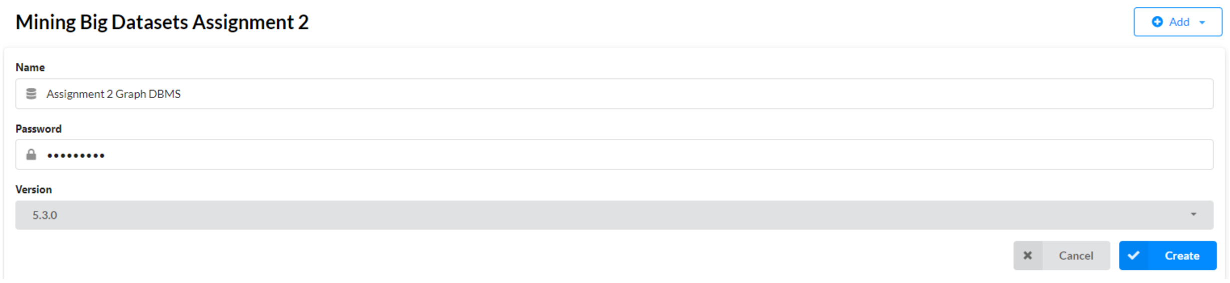


Figure 1: Creation of the Project in Neo4j Desktop.

Within this project, we will add a local database called **Assignment 2 Graph DBMS**, which will serve as the container for our graph model (*See Figure 2*).


Mining Big Datasets Assignment 2

Name: Assignment 2 Graph DBMS
Password: •••••••••
Version: 5.3.0

Figure 2: Creation of a Local Database, Assignment 2 Graph DBMS.

The next step will involve importing the dataset files, namely ArticleNodes.csv, AuthorNodes.csv, and Citations.csv, into the Assignment 2 Graph DBMS database. This will allow us to populate the database with the necessary nodes and relationships based on the provided data. To do so, we select to import the provided CSV files through the Neo4j instructions. The following *Figures 3 & 4* showcase this process and the successful importation of the given dataset files at the right Neo4j folder.

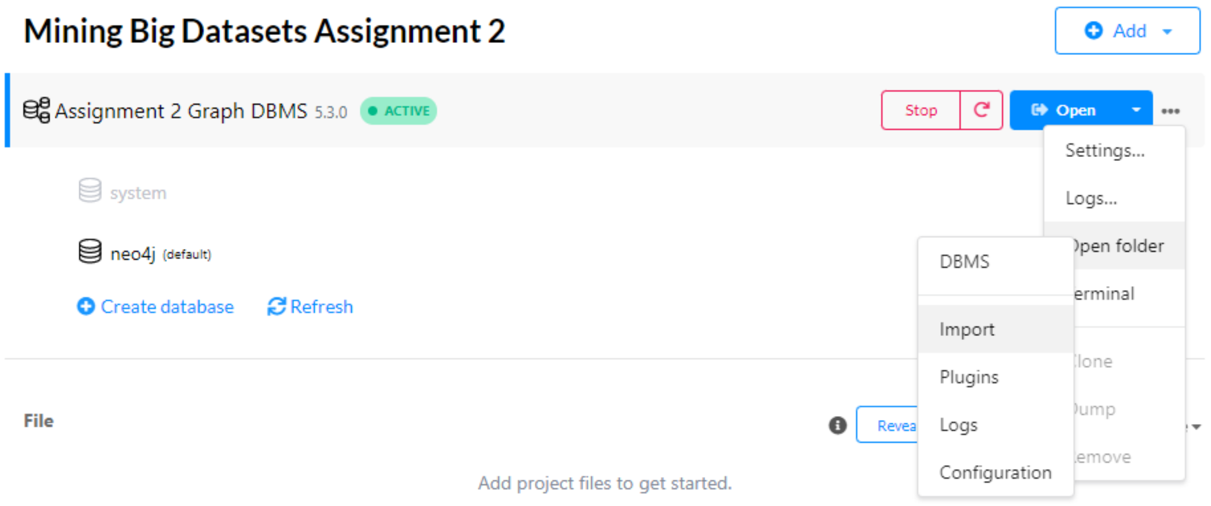
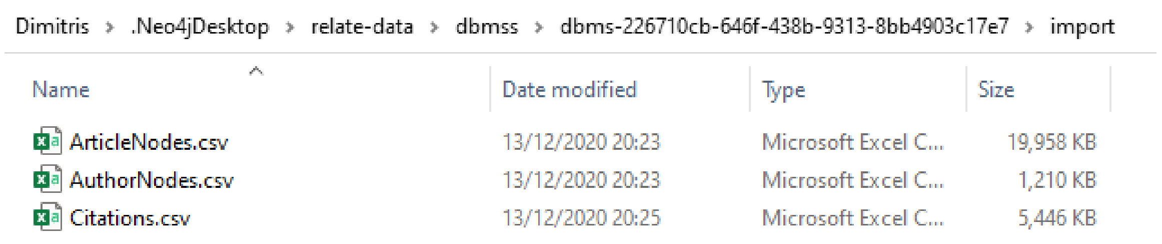


Figure 3: Importation of the provided CSV files (1/2).


Dimitris › .Neo4jDesktop › relate-data › dbmss › dbms-226710cb-646f-438b-9313-8bb4903c17e7 › import

Name	Date modified	Type	Size
ArticleNodes.csv	13/12/2020 20:23	Microsoft Excel C...	19,958 KB
AuthorNodes.csv	13/12/2020 20:23	Microsoft Excel C...	1,210 KB
Citations.csv	13/12/2020 20:25	Microsoft Excel C...	5,446 KB

Figure 4: Importation of the provided CSV files (2/2).

Once the data is successfully imported, we can proceed with creating the graph model by establishing the relationships between articles, authors, and journals as described earlier. In the picture below, you can see the activation of the database and the transition to Neo4j Browser in order to create the constraints, nodes, and relationships through Cypher Query Language.

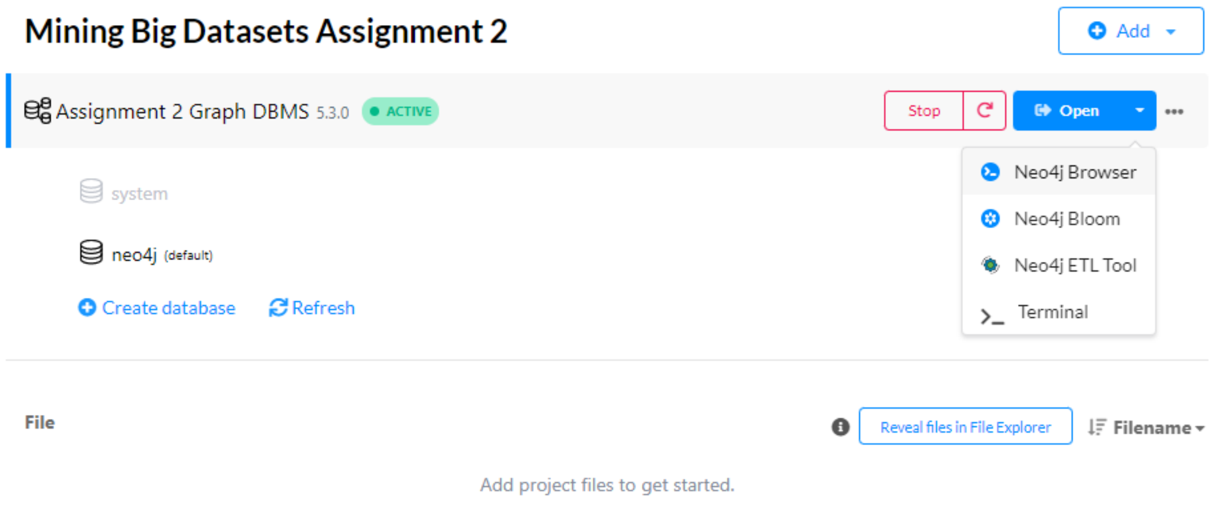

Mining Big Datasets Assignment 2
Add
Assignment 2 Graph DBMS 5.3.0
ACTIVE
Stop
Open
Neo4j Browser
Neo4j Bloom
Neo4j ETL Tool
Terminal
system
neo4j (default)
Create database
Refresh
File
Reveal files in File Explorer
Filename
Add project files to get started.

Figure 5: Activation of Local Database and Transition to Neo4j Browser.

2.2. Nodes and Constraints Creation

In this section, we are going to initialize the graph model by creating the nodes with respect to the given constraints and load the data from the already imported CSV datasets. The first initializations step is to set the constraint of unique id insertion for the node type "article". This constraint will facilitate the insertion of the data into the system as it will discard the duplicate article node insertion and this will lead to abolish further relationships among same node entities that will be presented multiple times. In the code snippet below the constraint and the insertion of article nodes is presented.

```
// Create Constraints and Nodes for Article.

CREATE CONSTRAINT FOR (article:Article) REQUIRE article.id IS UNIQUE;

LOAD CSV
FROM "file:///ArticleNodes.csv" AS Articles
FIELDTERMINATOR ','
CREATE (article:Article{id:toInteger(Articles[0]), title:Articles[1], year:Articles[2],
abstract:Articles[4]});
```

Code snippet: 1

To begin with, the create statement is used in order to put the constraint on the article nodes id uniqueness and in this way the system is pre-informed for the creation of a particular node type before actual data will be inserted into the graph database as article node types. Consequently, the csv file that hosts the data is load after the specification of the corresponding directory where the file is located, the delimiter symbol is determined for the file's original scan and finally, the create statement is used in order to create the article node type along with its corresponding properties and assign the corresponding data of the file to the article nodes components of the schema.

The next step contains the creation of the node type "author", this type contains the entities that represent the authors who wrote the entities articles. The fundamental constraint of the uniqueness of the author's name is set so as not to permit the insertion of author nodes that will represent duplicates that will lead to duplicate relationships establishment. The author's name represents this entity's identity. The csv file that hosts the authors data especially the authors' name.

To be more specific, the provided code segment is a snippet of a Cypher query in the Neo4j graph database. It begins by creating a uniqueness constraint on the **name** property of nodes labeled as **Author**. Then, it loads data from a CSV file called "AuthorNodes.csv" and assigns each row to the variable **row**. The query filters out rows where the second column is empty, and for each valid row, it uses the **MERGE** command to create or match an **Author** node with the name property value from the second column. If a match is found, it updates the **id** property of the matched node using the integer value from the first column. Overall, this code segment establishes a uniqueness constraint and creates nodes labeled as **Author** based on data from a CSV file while updating the **id** property for matched nodes.

```
/ Create Constraints and Nodes for Author.

CREATE CONSTRAINT FOR (author:Author) REQUIRE author.name IS UNIQUE;

LOAD CSV
FROM "file:///AuthorNodes.csv" AS row
WITH row
WHERE row[1] IS NOT NULL
MERGE (n:Author {name: row[1]})
ON MATCH SET n.id = toInteger(row[0]);
```

Code snippet: 2

The last entity of the model's architectural schema is the node type "journal" the node type journal hosts the name of the scientific venues that offer the publication floor for the articles to be published to scientific community. To the code snippet below a uniqueness constraint is set to the journals attribute name that is called journal. The csv file that hosts the journal names of the publication is loaded, the rows with not null journal name are kept and finally the property of he journal name is stored to each corresponding node.

```
CREATE CONSTRAINT FOR (j:Journal) REQUIRE j.journal IS UNIQUE;

LOAD CSV
FROM "file:///ArticleNodes.csv" AS journals
WITH journals
WHERE journals[3] IS NOT NULL
MERGE (j:Journal {journal: journals[3]})
ON MATCH SET j.id = toInteger(journals[0]);
```

Code snippet: 3

Finally, the code sets the uniquely matched journal nodes to have the corresponding id that is converted to integer. With this part of the code the completion of the data load into the graph data base model is completed. In the following figure the nodes created for the data base model's needs is presented below.

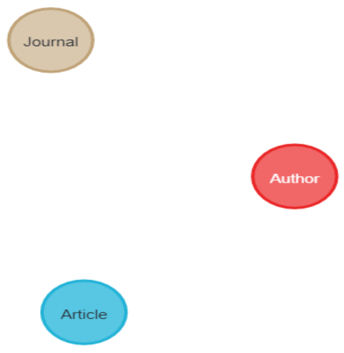


Figure 6: Node types of the graph data base.

2.3. Relationships Creation

The next important step in order to establish the graph data base model is to set the relationships among the node types along with the constraints for each relationship set among the nodes. For the needs of the particular case, we have identified three types of relationships, the first relationship concerns the article nodes and the journal nodes and the relationship that link these entities is the "Published in" and connects the article to the journal that represent the scientific venue in which the particular article is published in. The code below is the implementation code in the system.

```
// Relationship PUBLISHED_IN.

LOAD CSV FROM "file:///ArticleNodes.csv" AS row
MATCH (article:Article), (j:Journal)
WHERE article.id = toInteger(row[0]) AND j.journal = row[3]
CREATE (article) - [:PUBLISHED_IN] -> (j);
```

Code snippet: 3

The csv file that contains in its lines the article id and the journal id is published in is loaded in the system, where we match article nodes with journal nodes and the relationships are created between the corresponding article id and journal id the file implies and expresses the relationship between the specific nodes of these two node types. Finally, the create statement establishes the relationship "Published in" between two nodes.

The second relationship declares the authors that have written a particular article. There is case in which only one author is involved to the writing process of the article or many different authors are involved in ones writing process of the same article. The code snippet below shows the code that establishes the relationship of "Written by" as it declares from which authors an article is written.

```
// Relationship WRITTEN_BY.

LOAD CSV FROM "file:///AuthorNodes.csv" AS row
MATCH (article:Article), (author:Author)
WHERE article.id = toInteger(row[0]) AND author.name = row[1]
CREATE (article) - [:WRITTEN_BY] -> (author);
```

Code snippet: 4

The code starts by loading data from a CSV file named "AuthorNodes.csv" and assigns each row to the variable "row". It then matches nodes labeled as "Article" and "Author".

Filtering is applied to find rows where the "id" property of the article node matches the first column (converted to an integer) of the CSV row, and the "name" property of the author node matches the value in the second column of the CSV row. Finally, a relationship of type "WRITTEN_BY" is created between the article node and the author node, connecting them together using a directed arrow-like symbol "->". This code establishes the relationship between articles and authors based on the provided CSV data, indicating that an article is written by a specific author.

Lastly, the final relationship between the node types is settled. The relationship "Cites" is established among the same node type, is established among the articles when an article contains in its bibliography session another article and in this way a citation is being made for the second article.

```
// Relationship CITES.

LOAD CSV
FROM "file:///Citations.csv" AS citations
FIELDTERMINATOR '\t'
MATCH (article1:Article {id: toInteger(citations[0])})
MATCH (article2:Article {id: toInteger(citations[1])})
MERGE (article1)-[:CITES]->(article2);
```

Code snippet: 5

The code starts by loading data from a tab-separated CSV file named "Citations.csv" and assigns each row to the variable "citations". It specifies the field terminator as '\t' to indicate that the values are separated by tabs. Then, it matches nodes labeled as "Article" where the "id" property of the first article node matches the first column (converted to an integer) of the CSV row, and the "id" property of the second article node matches the second column (converted to an integer) of the CSV row. Finally, a relationship of type "CITES" is created between the first article node and the second article node, connecting them together. This code establishes the relationship between articles based on the provided CSV data, indicating that one article cites another.

After the completion of the data loading process and the formation of the relationships between the graph data base entities the model schema of the graph database is completed and presented in figure 7. All relationships are directed between nodes and are depicted in detail below. The article node type is the centred node that is the source node for any directed relationship. An article node type is written by an author node type, is published in a journal node type and it cites another article node type.

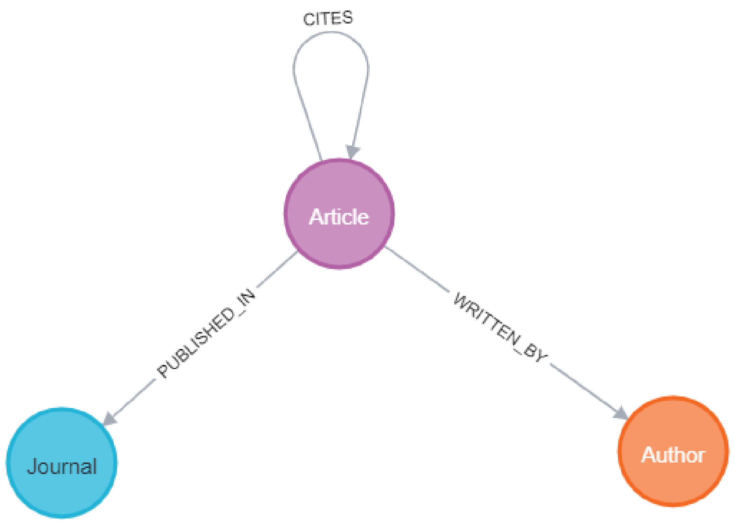


Figure 7: Graph data base schema.

2.4. Examples of complete relationship cases in the graph database.

All the accomplished relationships and the dynamics of the data base are gradually depicted in the sample queries' graph outputs that follow. In figure 8 we see the "written by" relationship that exists in the data base for a particular article. It is important that this type of relationship between these two node types of articles and author is one to many relationships as the same article is written by multiple authors.

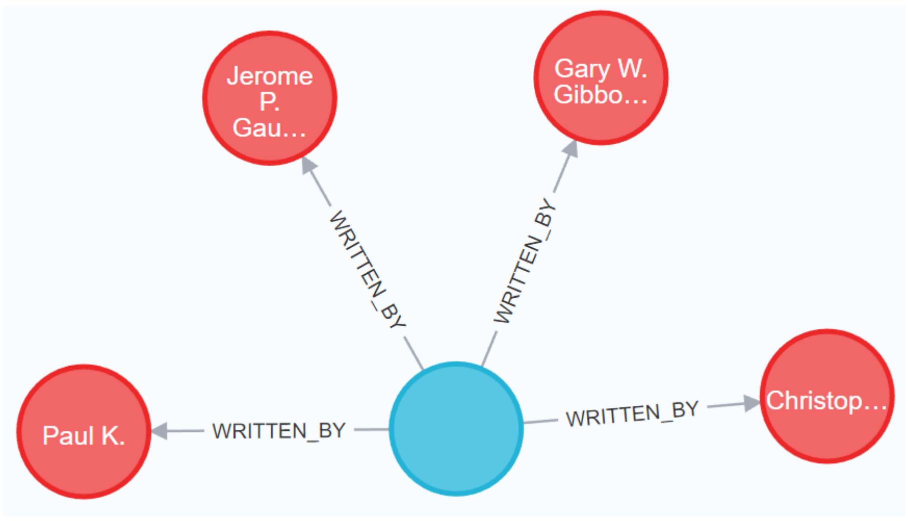


Figure 8: "Written by" relationship among the article nodes and the author nodes.

Another important relationship is the "Published in " relationship that connects the article to the journal node that is the scientific venue that published the particular article. Most of the times an article is published in one venue but this is not a relationship type constraint. In figure 9 we see this relationship among an article node (blue nodes in the graph data base) and a journal node (golden nodes in the graph data base).

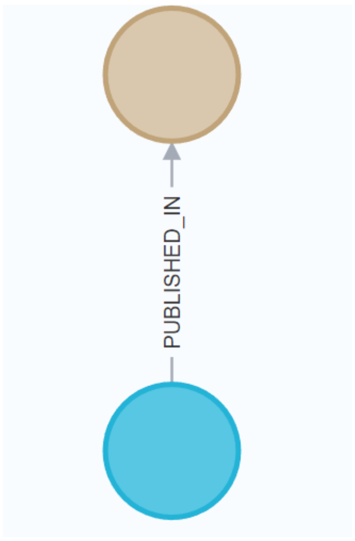


Figure 9: "Published in" relationship between the article nodes and the journal nodes.

Finally, in figure 10 the "Cites" relationship is depicted. This relationship is formed among nodes of the same type inside the database this is the reason that we see only blue article nodes in the graph result representation of the query. The general character of the relationship inside the particular graph data base is many to many as an article cites many other articles and is cited by many other articles of common scientific context as well.

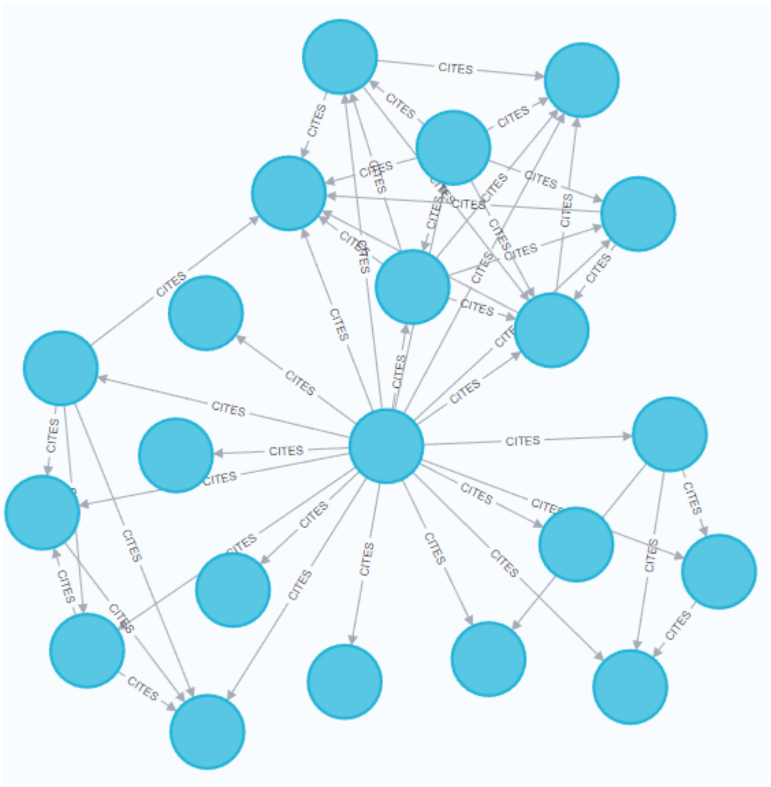


Figure 10: "Cites" relationship between article nodes.

In order to have a more holistic view of the relationships' chain effect development inside the database, figure 11 depicts a restricted example of all possible relationships developed in the graph data base among all node types.

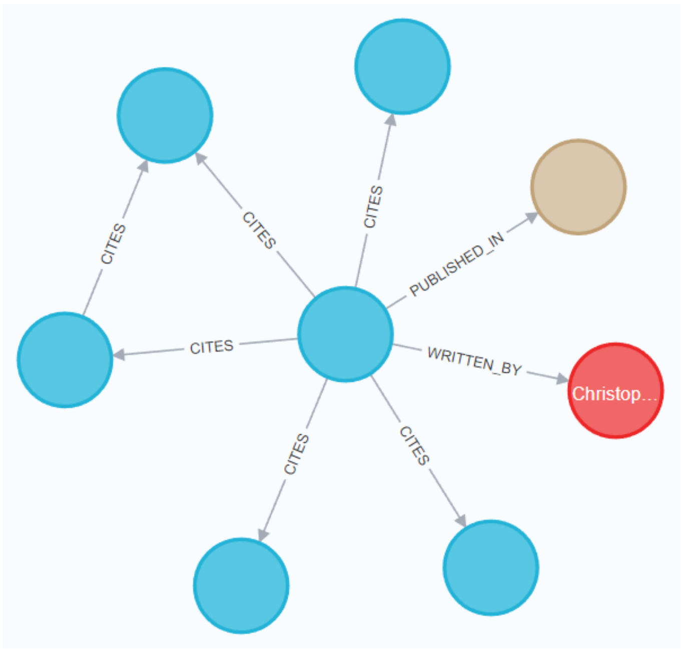


Figure 11: Relationships canvas example among all node types.

3. Querying the database with Cypher

Cypher is a declarative query language specifically designed for graph databases, with **Neo4j** being its most prominent implementation. It provides a powerful and intuitive way to interact with graph data by expressing graph patterns and traversal operations in a concise and readable syntax.

Furthermore, Cypher allows users to create, retrieve, update, and delete graph data using a pattern-matching approach, which makes it well-suited for tasks such as data modeling, graph exploration, and graph-based analytics. Its expressive nature enables developers and analysts to write complex queries that uncover valuable insights and relationships within connected data. With Cypher, users can seamlessly navigate the graph's nodes, relationships, and properties, making it an essential tool for working with graph databases and leveraging the power of connected data.

In the following sections we will answer the demanded queries though **Cypher** declarative query language and **Neo4j.**

3.1. Which are the top 5 authors with the most citations (from other papers). Return author names and number of citations.

The first query we are asked to execute is refereeing to the top five authors. More specifically, the problem at hand is to determine the top 5 authors who have received the highest number of citations from other papers and output the authors names and the numbers of citations they received.

To answer the question of identifying the top 5 authors with the most citations from other papers, we can develop a query using the Cypher language in Neo4j. By executing the following query, we can obtain the desired result:

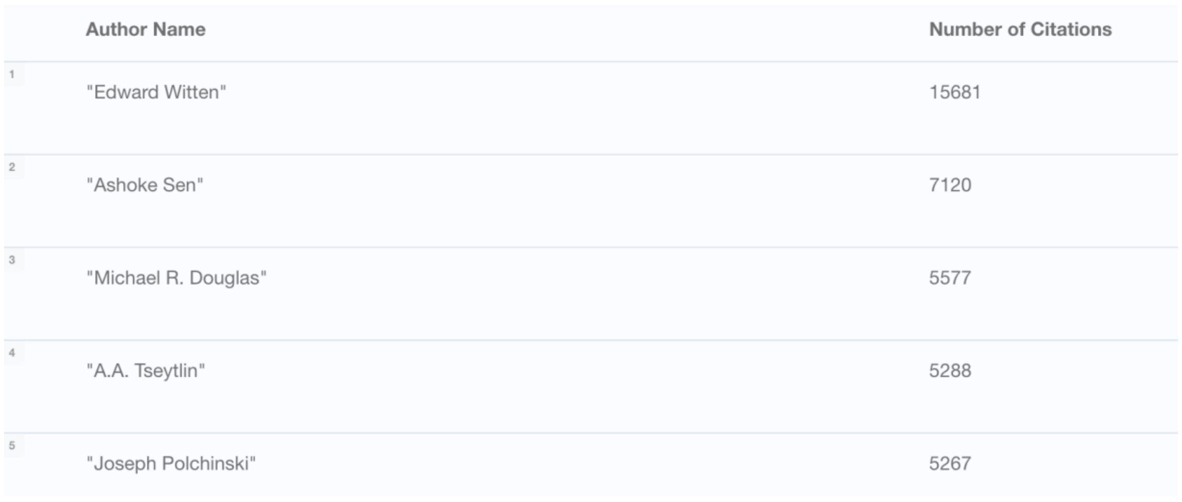
```
MATCH (article1:Article)-[:CITES]->(article2:Article)-[:WRITTEN_BY]->(author:Author)
WITH author, count(article2) AS `Number of Citations`
ORDER BY `Number of Citations` DESC
RETURN author.name AS `Author Name`, `Number of Citations`
LIMIT 5
```

Code Snippet: 6

Explaining what we have done code wise, we start by matching the relevant nodes in the graph. We specify that we want to match an 'Article' node labeled as 'article1' that has a 'CITES' relationship to another 'Article' node labeled as 'article2'. Furthermore, we establish a 'WRITTEN_BY' relationship from 'article2' to an 'Author' node labeled as 'author'.

After matching the nodes, we use the 'WITH' clause to continue working with the retrieved data. We keep the 'author' node and calculate the count of 'article2' nodes connected to it, representing the number of citations received by the author. We assign this count to the alias 'Number of Citations'. Then, we order the results in descending order based on the 'Number of Citations' using the 'ORDER BY' clause. This ensures that authors with higher citation counts appear first in the result set.

Moving forward, we specify what information to return in the result using the 'RETURN' clause. We select the 'author.name' property to retrieve the name of the author, and we include the 'Number of Citations' alias to display the corresponding citation count for each author. Lastly, to limit the results to the top 5 authors, we utilize the 'LIMIT' clause, which restricts the number of returned results to 5. By executing this query, we obtain a list of the top 5 authors along with the number of citations they have received from other papers, which can be seen in the below figure:



	Author Name	Number of Citations
1	"Edward Witten"	15681
2	"Ashoke Sen"	7120
3	"Michael R. Douglas"	5577
4	"A.A. Tseytlin"	5288
5	"Joseph Polchinski"	5267

Figure 12: The top 5 authors with the most citations (from other papers).

3.2. Which are the top 5 authors with the most collaborations (with different authors). Return author names and number of collaborations.

The second question asks for the identification of the top 5 authors who have the highest number of collaborations with different authors. The objective is to retrieve the names of these authors along with the corresponding count of collaborations. The result will provide insights into the most prolific and well-connected authors in terms of collaborations with diverse peers.

To answer this question, we execute the following query:

```
MATCH (author1:Author)<-[:WRITTEN_BY]- (article:Article) - [:WRITTEN_BY] -> (author2:Author)
WHERE author1.name<>author2.name
RETURN author1.name AS `Author Name`, COUNT(DISTINCT author2) AS `Number of Collaborations`
ORDER BY `Number of Collaborations` DESC
LIMIT 5
```

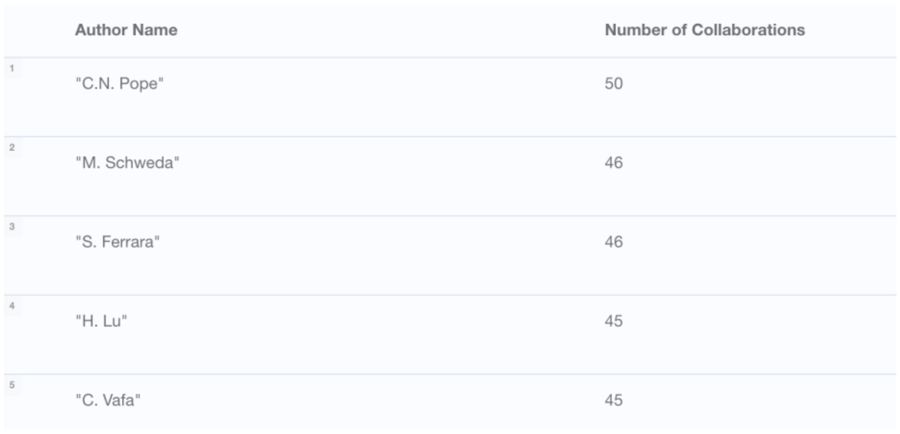
Code snippet: 7

Let's break down the above presented query code and explain its functionality in detail. To begin with, the first line of the code, **MATCH (author1:Author)<-[:WRITTEN_BY]- (article:Article) - [:WRITTEN_BY] -> (author2:Author)**, sets up the matching pattern in the graph. It starts by matching an 'Author' node labeled as 'author1' that is connected to an 'Article' node through the 'WRITTEN_BY' relationship. The 'Article' node is then connected to another 'Author' node labeled as 'author2' through the same 'WRITTEN_BY' relationship. This pattern captures collaborations between authors via shared articles.

The second line of the code, **WHERE author1.name<>author2.name**, adds a filter condition. With this filter it ensures that only collaborations between different authors are considered. By comparing the 'name' property of 'author1' and 'author2', we exclude collaborations where an author is matched with themselves.

Moving to the third line, **RETURN author1.name AS 'Author Name', COUNT(DISTINCT author2) AS 'Number of Collaborations'**, we define the information to be returned as the result. We select the 'name' property of 'author1' as 'Author Name' and use the 'COUNT' function along with 'DISTINCT' to count the distinct occurrences of 'author2' nodes. This count represents the number of collaborations for each author. The fourth line, **ORDER BY 'Number of Collaborations' DESC**, sorts the results in descending order based on the 'Number of Collaborations'. Authors with a higher number of collaborations will appear first in the result set, providing a ranking of the most collaborative authors.

Finally, the fifth line, **LIMIT 5**, limits the number of results to 5, ensuring that only the top 5 authors with the most collaborations are included in the final output. In the next figure, you will find the results of the above query, which will include the names of the top 5 authors and the corresponding number of collaborations for each author.



	Author Name	Number of Collaborations
1	"C.N. Pope"	50
2	"M. Schweda"	46
3	"S. Ferrara"	46
4	"H. Lu"	45
5	"C. Vafa"	45

Figure 13: The top 5 authors with the most collaborations (with different authors).

3.3. Which is the author who has wrote the most papers without collaborations. Return author name and number of papers.

The problem assigned by the third question is to determine the author who has written the highest number of papers without collaborations. The objective is to identify the author who has authored the most papers without having any co-authors. The desired output will provide the name of the author along with the count of papers they have written individually, showcasing their prolific output as a sole author.

The following query was executed for this reason:

```
MATCH (article:Article)-[:WRITTEN_BY]->(author1:Author)
MATCH (article:Article)-[:WRITTEN_BY]->(author2:Author)
WITH author1, COUNT(article) AS `Number of Papers Without Collaboration`, COUNT(DISTINCT author2) AS PapersCou
nter
WHERE PapersCounter = 1
RETURN author1.name AS `Author Name`, `Number of Papers Without Collaboration`
ORDER BY `Number of Papers Without Collaboration` DESC
LIMIT 1
```

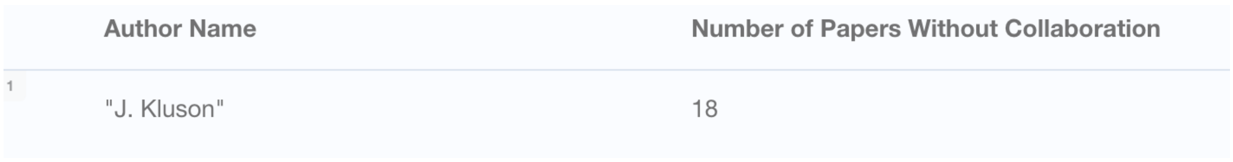
Code Snippet: 8

To explain the code we will break it down line by line. Therefore, in the first line of the query, **MATCH (author1:Author)<-[:WRITTEN_BY]-(article:Article)-[:WRITTEN_BY]->(author2:Author)**, a pattern is defined to match author nodes and article nodes connected by the "WRITTEN_BY" relationship. The query aims to capture collaborations between authors who have co-authored the same article. The second line, **WHERE author1.name<>author2.name**, adds a condition to exclude self-collaborations. It ensures that only collaborations between different authors are considered, preventing authors from being matched with themselves.

Moving to the next line, **RETURN author1.name AS 'Author Name', COUNT(DISTINCT author2) AS 'Number of Collaborations'**, the query specifies the information to be returned as the result. It selects the name of author1 as 'Author Name' and uses the 'COUNT' function along with 'DISTINCT' to count the distinct occurrences of author2, representing the number of collaborations for each author.

The fourth line, **ORDER BY 'Number of Collaborations' DESC, sorts the results in descending order based on the 'Number of Collaborations'**. This ensures that authors with the highest number of collaborations will appear first in the result set. Finally, the fifth line, **LIMIT 5**, limits the number of results to 5, so only the top 5 authors with the most collaborations are returned.

In the figure below, can be found the results of executing the above query, which will include the names of the top 5 authors and the corresponding number of collaborations for each author.



	Author Name	Number of Papers Without Collaboration
1	"J. Kluson"	18

Figure 14: The author who has wrote the most papers without collaboration.

3.4. Which author published the most papers in 2001? Return author name and number of papers.

The fourth question is asking for the name of the author who published the highest number of papers in the year 2001, along with the corresponding number of papers. In other words, it seeks information about the author who had the most publications during that specific year.

To resolve this, we wrote the following query:

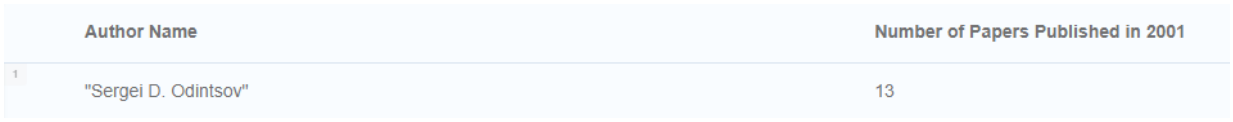
```
MATCH (journal:Journal)<-[:PUBLISHED_IN]-(article:Article)-[:WRITTEN_BY]-
>(author:Author)
WHERE article.year = '2001'
RETURN author.name AS `Author Name`, COUNT(article) AS `Number of Papers Published in 20
01`
ORDER BY `Number of Papers Published in 2001` DESC
LIMIT 1
```

Code Snippet: 9

Codewise firstly, we matched three nodes: Journal, Article, and Author. By establishing the relationships between them, we connected articles to their corresponding journals and authors. This allowed us to navigate the graph and gather the necessary information. Then, we applied a condition to filter the articles based on their publication year. We specifically looked for articles with a year value of '2001', as we were interested in publications from that specific year.

Moving on, we defined what data we wanted to retrieve. We specified that we wanted to return the name of the author, which we labeled as `Author Name`, as well as the count of articles written by that author in 2001, labeled as `Number of Papers Published in 2001`. These aliases make the output more readable and easier to understand. To identify the author with the highest number of papers, we ordered the results in descending order based on the count of articles. This arrangement allowed us to see which authors had the most publications in 2001.

Finally, we used the `LIMIT` clause to limit the output to only one result. By doing this, we obtained the author who had the highest count of published papers in 2001. This query helped us find the author's name and the corresponding number of papers, enabling us to identify the most prolific author in that particular year. This can be depicted in the figure below.



	Author Name	Number of Papers Published in 2001
1	"Sergei D. Odintsov"	13

Figure 15: The author published the most papers in 2001.

3.5. Which is the journal with the most papers about “gravity” (derived only from the paper title) in 1998. Return name of journal and number of papers.

The fifth question seeks to identify the journal that published the highest number of papers related to the topic of "gravity" in the year 1998. The focus is solely on the paper titles to determine the relevance to gravity. The query aims to retrieve the name of the journal and the corresponding number of papers published on this subject, allowing us to identify the journal with the most extensive coverage of gravity-related research in 1998.

There are two distinct cases considered in the provided queries to identify the journal with the highest number of papers related to "gravity" in 1998. In Case 1, the query counts only articles that contain the word "gravity" exactly as written, maintaining case sensitivity. This case ensures that only titles with the exact word "gravity" are considered, such as "Gravity and its Effects." On the other hand, Case 2 takes into account all occurrences of the word "gravity" in various case formats, such as "Gravity" or "GRAVITY."

Both cases are presented below:

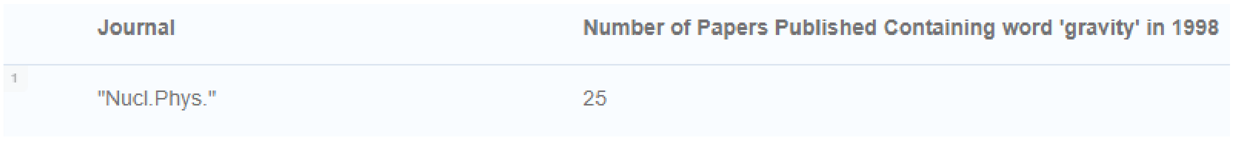
```
// Case 1 count only articles containing the word gravity - written this way.
MATCH (journal:Journal)<-[:PUBLISHED_IN]-(article:Article)
WHERE article.year = '1998' AND article.title CONTAINS "gravity"
RETURN journal.journal AS `Journal`, COUNT(article) AS `Number of Papers Published Containing word 'gravity' in 1998`
ORDER BY `Number of Papers Published Containing word 'gravity' in 1998` DESC
LIMIT 1

// Case 2 all occurencies of gravity - like Gravity or GRAVITY.
MATCH (journal:Journal)<-[:PUBLISHED_IN]-(article:Article)
WHERE article.year = '1998' AND toLower(article.title)CONTAINS"gravity"
RETURN journal.journal AS `Journal`, COUNT(article) AS `Number of Papers Published Containing word 'gravity' in 1998`
ORDER BY `Number of Papers Published Containing word 'gravity' in 1998` DESC
LIMIT 1
```

Code Snippet: 10

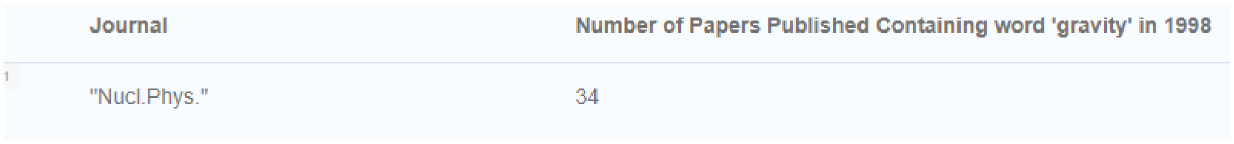
Let's break down the above queries, which except from the matching expression are identical. In Case 1, the query matches Journal and Article nodes, filters the articles based on the year 1998, and checks for an exact match of the word "gravity" in the titles using the `CONTAINS` function. The query returns the name of the journal and the count of articles meeting the criteria, ordered in descending order by the count.

On a similar note for the case 2, the query follows a similar structure but additionally applies the `toLower()` function to convert the titles to lowercase. This enables a case-insensitive search, capturing all occurrences of "gravity" in various case formats. The `CONTAINS` function is then used on the lowercase titles to identify articles containing the word "gravity". The query returns the name of the journal, the count of articles, and limits the result to the top journal with the highest count. Both cases provide insights into the journals with the most papers related to "gravity" in 1998, employing different methods of matching the word in the titles. The results are presented in the figures below, each one related to each case:



	Journal	Number of Papers Published Containing word 'gravity' in 1998
1	"Nucl.Phys."	25

Figure 16: The journal with the most papers about “gravity” in 1998 (Case 1).



	Journal	Number of Papers Published Containing word 'gravity' in 1998
1	"Nucl.Phys."	34

Figure 17: The journal with the most papers about "gravity" in 1998 (Case 2).

From the above Figures (16 & 17) we can see that both cases queries output the **same** journal that published the most papers related to gravity (from their titles), but with a different number of papers. This happens since the second case match more papers due to containing all the variations of the word gravity as mentioned above.

3.6. Which are the top 5 papers with the most citations? Return paper title and number of citations.

The sixth case aims to identify the top five papers with the highest number of citations. The focus is on retrieving the paper titles and their corresponding citation counts. The executed query can be found below:

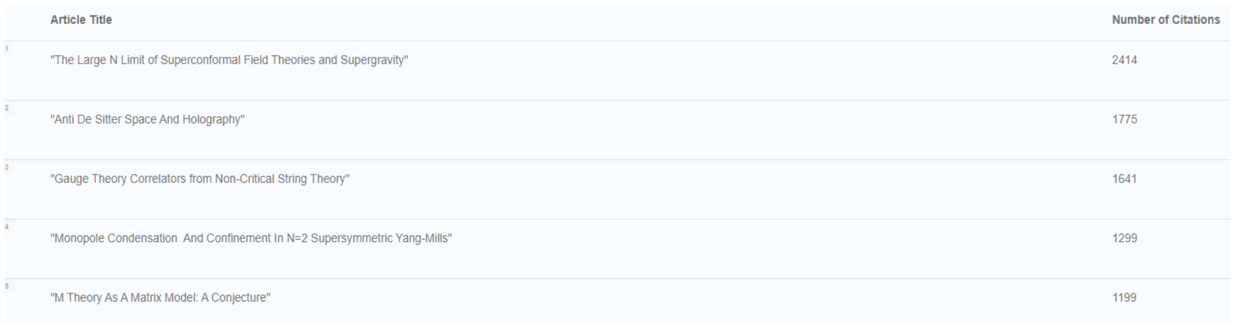
```
MATCH (article1:Article)-[cites:CITES]->(article2:Article)
RETURN article2.title AS `Article Title`, COUNT(cites) AS `Number of Citations`
ORDER BY `Number of Citations`
DESC LIMIT 5
```

Code Snippet: 11

In the above query, we used the MATCH clause to match two Article nodes: `article1` and `article2`. These nodes represent a relationship where `article1` cites `article2`. Afterwards, to retrieve the desired information, we used the RETURN clause. We specified that we wanted to return the title of `article2`, representing the cited article, using the alias `Article Title`. Additionally, we counted the number of `cites` relationships between the two articles and returned this count as `Number of Citations`.

To determine the top papers, we applied sorting using the ORDER BY clause. We ordered the results based on the `Number of Citations` in descending order. This arrangement ensured that the papers with the highest number of citations would appear at the top of the results. Furthermore, we used the LIMIT clause to restrict the output to only the top five papers. By setting the limit to 5, we ensured that the query would return the titles and citation counts of the most cited papers.

The results of this query when executed can be found below:



	Article Title	Number of Citations
1	"The Large N Limit of Superconformal Field Theories and Supergravity"	2414
2	"Anti De Sitter Space And Holography"	1775
3	"Gauge Theory Correlators from Non-Critical String Theory"	1641
4	"Monopole Condensation And Confinement In N=2 Supersymmetric Yang-Mills"	1299
5	"M Theory As A Matrix Model: A Conjecture"	1199

Figure 18: The top 5 papers with the most citations.

3.7. Which were the papers that use "holography" and "anti de sitter" (derived only from the paper abstract). Return authors and title.

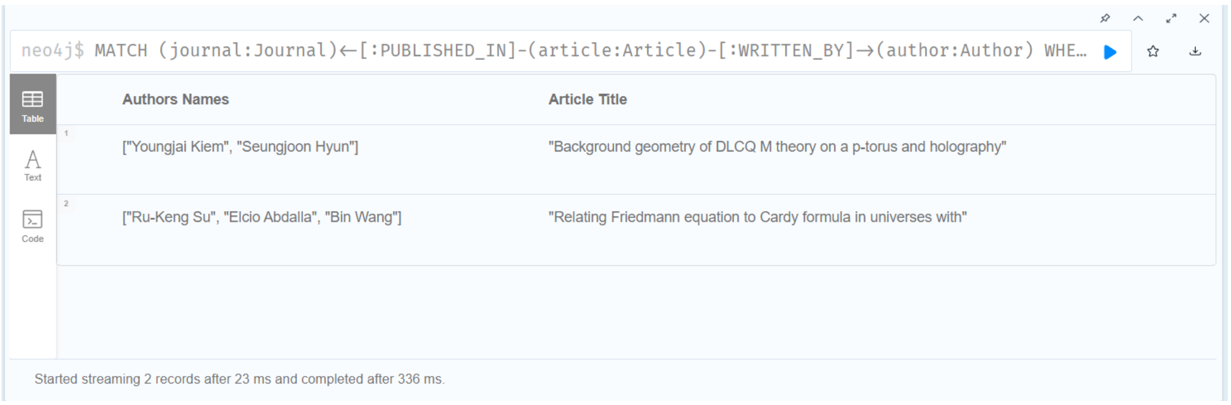
In this question we need to extract the articles along with their authors, that contain in their abstract descriptions the key words "holography" and "anti de sitter" that highlight the field of research this article analyses and projects. In the code snippet below we se the cypher query that extracts this information.

```
MATCH (journal:Journal)<-[:PUBLISHED_IN]-(article:Article)-[:WRITTEN_BY]->(author:Author)
WHERE toLower(article.abstract) CONTAINS "holography" AND toLower(article.abstract) CONTAINS "anti de sitter"
RETURN COLLECT(author.name) AS `Authors Names`, article.title AS `Article Title`
// Matching 2 articles and output all authors.
```

Code Snippet: 12

The query matches patterns in the graph where there is a relationship of type **"PUBLISHED_IN"** from the journal node to the article node, and a relationship of type **"WRITTEN_BY"** from the article node to the author node. It applies filtering conditions to check if the lowercase version of the article's abstract contains both the terms "holography" and "anti de sitter".

We should highlight that the step of turning both abstract and the terms of interest in lowercase characters permits the query to match the information and finally extract the articles with the respective content in their abstract, otherwise the different combinations of uppercase and lowercase of the terms inside the abstracts did not allow the program to find any matches. To be more specific, the system returns the collected names of authors as the "Authors Names" result and the title of the article as the "Article Title" result. The query aims to find articles that meet the specified abstract criteria and retrieve all the authors associated with those articles. In figure… the query result is presented.



```
neo4j$ MATCH (journal:Journal)←[:PUBLISHED_IN]-(article:Article)-[:WRITTEN_BY]→(author:Author) WHE…
```

	Authors Names	Article Title
1	["Youngjai Kiem", "Seungjoon Hyun"]	"Background geometry of DLCQ M theory on a p-torus and holography"
2	["Ru-Keng Su", "Elcio Abdalla", "Bin Wang"]	"Relating Friedmann equation to Cardy formula in universes with"

```
Started streaming 2 records after 23 ms and completed after 336 ms.
```

Figure 19: Articles and their authors that concern "holography" and "anti de sitter".

3.8. Find the shortest path between 'C.N. Pope' and 'M. Schweda' authors (use any type of edges). Return the path and the length of the path. Comment about the type of nodes and edges of the path.

In this question we will explore the path of any type of relationship that links two author nodes with less recorded steps of intermediate nodes. In this way we see how connected these two authors are and if they are directly connected through a relationship of co-authoring or are indirectly connected. In the code snippet below we have the query used.

```
MATCH path = shortestPath((au1:Author)-[*]-(au2:Author))
WHERE au1.name = "C.N. Pope" AND au2.name = "M. Schweda"
RETURN [n in nodes(path)] AS Path, length(path) AS `Path Length`
```

Code Snippet: 13

The provided query finds the shortest path between two authors in the graph, specifically between an author with the name "C.N. Pope" (au1) and an author with the name "M. Schweda" (au2). It utilizes the **shortestPath** function to search for the shortest path considering all possible relationships between the two authors.

The query then returns the path as a collection of nodes using the **nodes'** function applied to the **path** variable, and the length of the path as **Path Length**. The aim is to find the shortest path and provide information about the nodes along that path and its length, representing the connections between the authors "C.N. Pope" and "M. Schweda" in the graph. The output of the query is the following in figure 20.

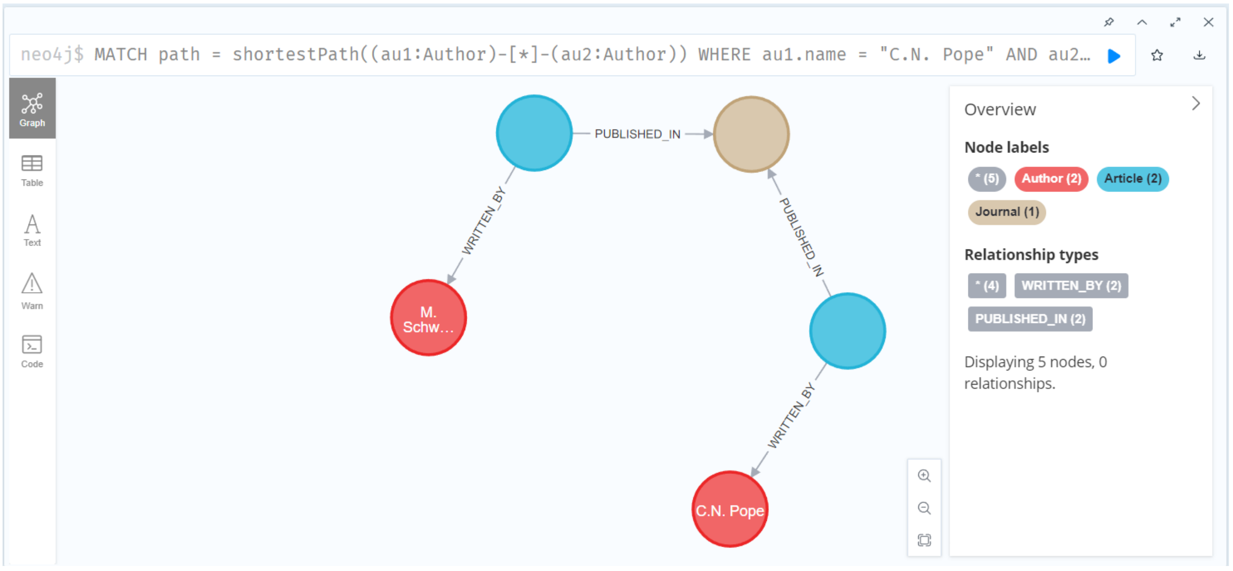

```
neo4j$ MATCH path = shortestPath((au1:Author)-[*]-(au2:Author)) WHERE au1.name = "C.N. Pope" AND au2…
```
Overview
Node labels
* (5) Author (2) Article (2) Journal (1)
Relationship types
* (4) WRITTEN_BY (2) PUBLISHED_IN (2)
Displaying 5 nodes, 0 relationships.

Figure 20: Shortest path between authors "C.N. Pope" and "M.Schweda" with all possible relationship edge types.

We see that the two authors are not directly connected in a co-authoring relationship of the same article but they are indirectly connected with 4 intermediate steps and three nodes. The nodes that are between the two authors are article node types (2 out of 3 nodes), a one journal node (one out of three nodes).

The relationships that are implicated are of two different types and are the "Written_BY" relationship and the "Published_IN" relationship. To be more specific, two out of four relationships presented are "Written_BY" an the other two are type "Published_IN". The actual connection between those tow authors is that they have writen articles that have been published in the same venue.

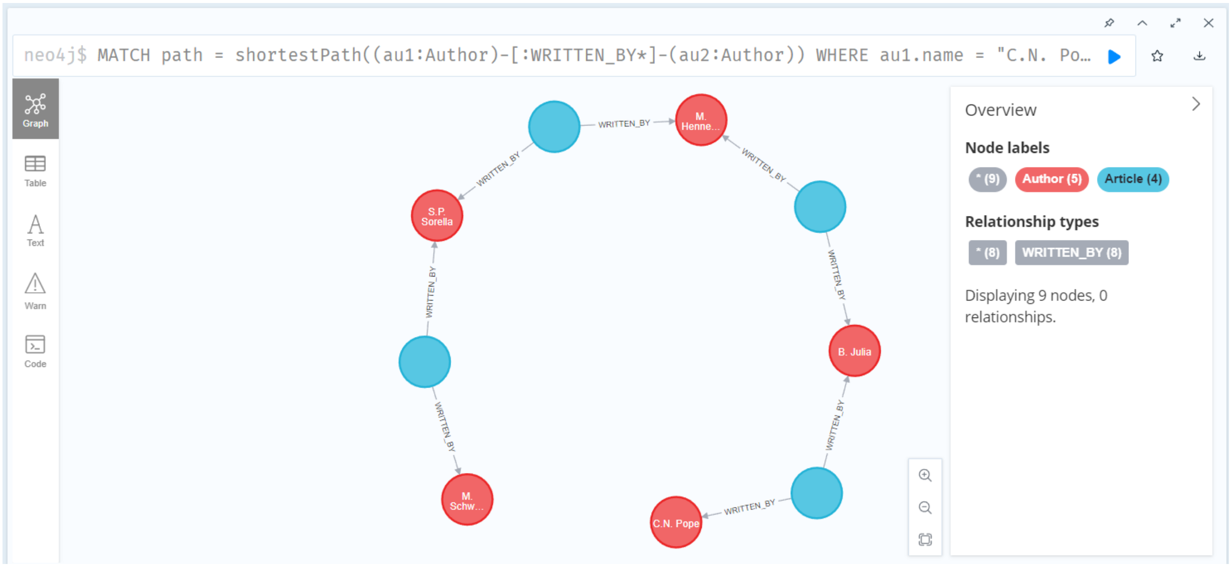
3.9. Run again the previous query (8) but now use only edges between authors and papers. Comment about the type of nodes and edges of the path. Compare the results with query 8.

In this question, we have to identify the shortest path of connection between the same author nodes but putting restrictions on the intermediate node types and relationship types that are permitted to use in order to link the two authors and more specifically only author and article nodes can be used and consequently only the "Written_BY" relationship can be used as a permitted edge type for linking the nodes. In the code snippet below we see how these constraints are implemented in the previous query.

```
MATCH path = shortestPath((au1:Author)-[:WRITTEN_BY*]-(au2:Author))
WHERE au1.name = "C.N. Pope" AND au2.name = "M. Schweda"
RETURN [n in nodes(path)] AS Path, length(path) AS `Path Length`
```

Code Snippet: 14

The provided query utilizes the **shortestPath** function to find the shortest path between two authors in the graph, specifically between an author named "C.N. Pope" (au1) and an author named "M. Schweda" (au2). The relationship pattern specified in the query is **[:WRITTEN_BY*]**, indicating that any number of "WRITTEN_BY" relationships can exist between the authors along the path. The query filters the paths to only consider those starting with "C.N. Pope" and ending with "M. Schweda". It then returns the path as a collection of nodes using the **nodes** function applied to the **path** variable, along with the length of the path as **Path Length**. The aim is to find the shortest path and provide information about the nodes along that path and its length, representing the connections between the authors "C.N. Pope" and "M. Schweda" based on the "WRITTEN_BY" relationships in the graph.


neo4j$ MATCH path = shortestPath((au1:Author)-[:WRITTEN_BY*]-(au2:Author)) WHERE au1.name = "C.N. Po…

Overview
Node labels
* (9) Author (5) Article (4)
Relationship types
* (8) WRITTEN_BY (8)
Displaying 9 nodes, 0 relationships.

Figure 21: Shortest path between the author nodes "C.N. Pope" and "M.Schweda" connected only through the "Written_BY" relationship.

As expected, the shortest path length is increased between the two authors as the constraint on the relationship type for indirectly connected author nodes will lead to a broader path of connection. In this shortest path we find out that there are 8 edges if the same relationship type and seven nodes of article and author type only as it is posed from the restriction and the link between the two authors is that they are co-authors with two other authors that have cooperated and they have written another article together.

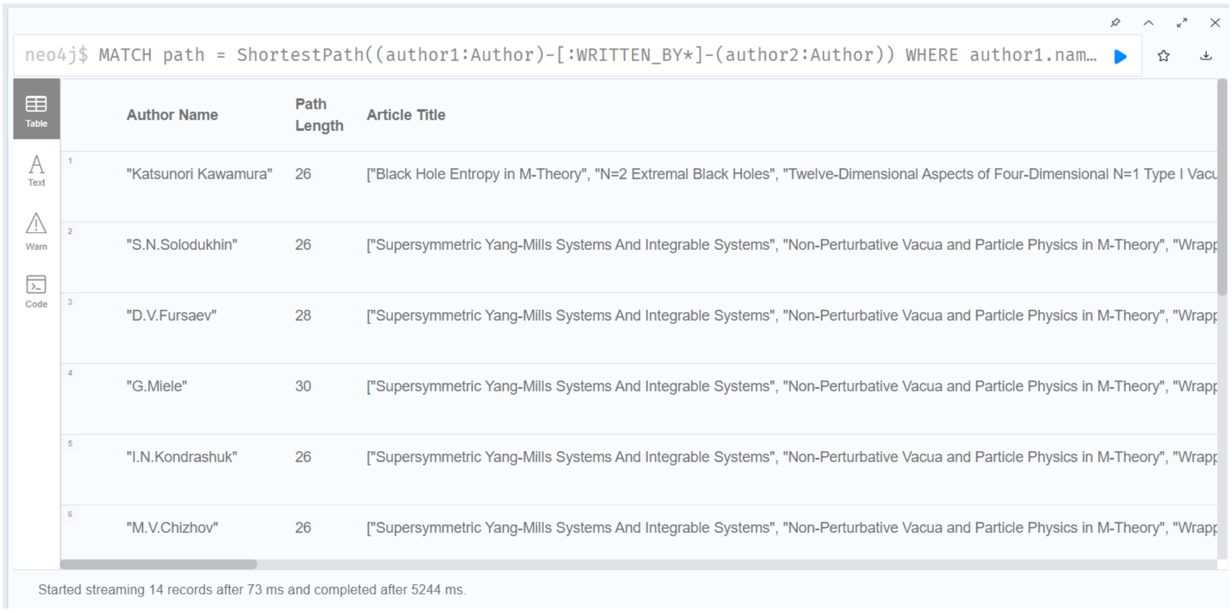
3.10. Find all authors with shortest path lengths > 25 from author 'Edward Witten'. The shortest paths will be calculated only on edges between authors and articles. Return author name, the length and the paper titles for each path.

In this question we want to find all authors that are connected with the author "Edward Written" through at least 24 other author nodes that are co-authors in sequential articles. In the code snippet below is the corresponding query.

```
MATCH path = ShortestPath((author1:Author)-[:WRITTEN_BY*]-(author2:Author))
WHERE author1.name = 'Edward Witten' AND author1<>author2
WITH author2.name AS `Author Name`, length(path) AS `Path Length`, [n in nodes(path) WHE
RE n.title IS NOT NULL| n.title] AS `Article Title`
WHERE `Path Length` > 25
RETURN `Author Name`, `Path Length`, `Article Title`
```

Code Snippet: 15

The provided query searches for the shortest path between two authors in the graph, namely an author with the name "Edward Witten" (author1) and any other author (author2). The relationship pattern **[:WRITTEN_BY*]** allows for any number of "WRITTEN_BY" relationships between the authors along the path. The query filters the paths to ensure that author1 and author2 are different and not the same person. It then extracts the **Author Name** of author2, the **Path Length** of the shortest path, and the **Article Title** of each node in the path where the title is not null. The query further filters the results to only include paths with a length greater than 25. Finally, it returns the **Author Name**, **Path Length**, and **Article Title** of the qualifying paths. The goal is to identify the shortest paths between "Edward Witten" and other authors in the graph, showcasing the associated article titles and their lengths while excluding paths that are too short. In figure 21 we have the results of the query.

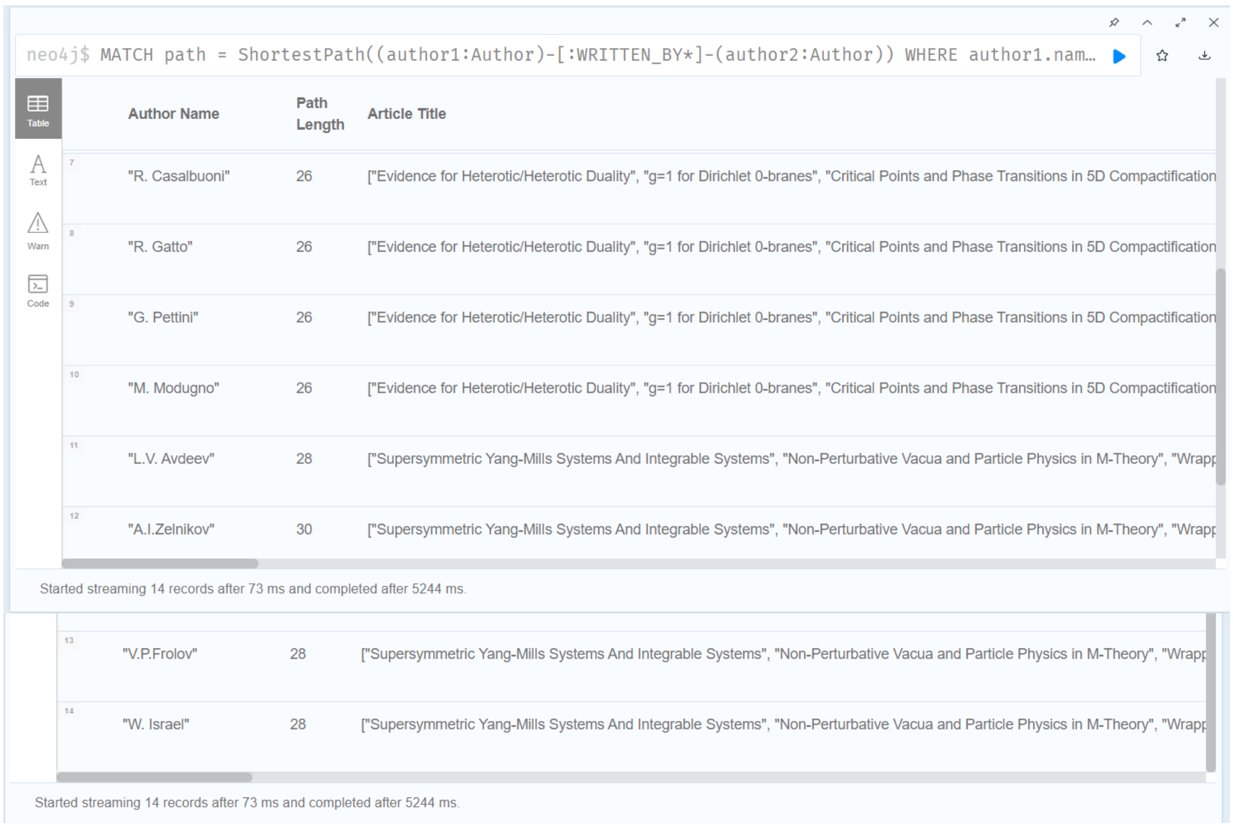


```
neo4j$ MATCH path = ShortestPath((author1:Author)-[:WRITTEN_BY*]-(author2:Author)) WHERE author1.nam…
```

	Author Name	Path Length	Article Title
1	"Katsunori Kawamura"	26	["Black Hole Entropy in M-Theory", "N=2 Extremal Black Holes", "Twelve-Dimensional Aspects of Four-Dimensional N=1 Type I Vacu
2	"S.N.Solodukhin"	26	["Supersymmetric Yang-Mills Systems And Integrable Systems", "Non-Perturbative Vacua and Particle Physics in M-Theory", "Wrapp
3	"D.V.Fursaev"	28	["Supersymmetric Yang-Mills Systems And Integrable Systems", "Non-Perturbative Vacua and Particle Physics in M-Theory", "Wrapp
4	"G.Miele"	30	["Supersymmetric Yang-Mills Systems And Integrable Systems", "Non-Perturbative Vacua and Particle Physics in M-Theory", "Wrapp
5	"I.N.Kondrashuk"	26	["Supersymmetric Yang-Mills Systems And Integrable Systems", "Non-Perturbative Vacua and Particle Physics in M-Theory", "Wrapp
6	"M.V.Chizhov"	26	["Supersymmetric Yang-Mills Systems And Integrable Systems", "Non-Perturbative Vacua and Particle Physics in M-Theory", "Wrapp

Started streaming 14 records after 73 ms and completed after 5244 ms.

Figure 22: Authors and final article that are indirectly connected to "Edward Written" in more than 25 steps of the "Written_BY" relationship between authors and articles (1/2).



```
neo4j$ MATCH path = ShortestPath((author1:Author)-[:WRITTEN_BY*]-(author2:Author)) WHERE author1.nam…
```

	Author Name	Path Length	Article Title
7	"R. Casalbuoni"	26	["Evidence for Heterotic/Heterotic Duality", "g=1 for Dirichlet 0-branes", "Critical Points and Phase Transitions in 5D Compactification
8	"R. Gatto"	26	["Evidence for Heterotic/Heterotic Duality", "g=1 for Dirichlet 0-branes", "Critical Points and Phase Transitions in 5D Compactification
9	"G. Pettini"	26	["Evidence for Heterotic/Heterotic Duality", "g=1 for Dirichlet 0-branes", "Critical Points and Phase Transitions in 5D Compactification
10	"M. Modugno"	26	["Evidence for Heterotic/Heterotic Duality", "g=1 for Dirichlet 0-branes", "Critical Points and Phase Transitions in 5D Compactification
11	"L.V. Avdeev"	28	["Supersymmetric Yang-Mills Systems And Integrable Systems", "Non-Perturbative Vacua and Particle Physics in M-Theory", "Wrapp
12	"A.I.Zelnikov"	30	["Supersymmetric Yang-Mills Systems And Integrable Systems", "Non-Perturbative Vacua and Particle Physics in M-Theory", "Wrapp
13	"V.P.Frolov"	28	["Supersymmetric Yang-Mills Systems And Integrable Systems", "Non-Perturbative Vacua and Particle Physics in M-Theory", "Wrapp
14	"W. Israel"	28	["Supersymmetric Yang-Mills Systems And Integrable Systems", "Non-Perturbative Vacua and Particle Physics in M-Theory", "Wrapp

```
Started streaming 14 records after 73 ms and completed after 5244 ms.
```

Figure 23: Authors and final article that are indirectly connected to "Edward Written" in more than 25 steps of the "Written_BY" relationship between authors and articles (2/2).

There are fourteen writers that relate to the "Edward Written" with more than 24 intermediate nodes of type article and author through the "Written_BY" relationship. The shortest paths' lengths vary between 26 and 30 steps, the names of he final authors that are indirectly connected with the "Edward Written" are noted to the first column and finally the article title that is the last article written from the desired author in the relations path is printed on the third column.

Deliverables

The deliverables of this assignment will be a compressed file, which will contain the followings:

- The current documentation of the assignment, Mining Big Datasets.pdf
- The initialization file of the database, graph_model.txt
- The demanded queries along with the results commended out, queries.txt

Note: The code files are in txt format to prevent any possible issues.