Athens University of Economics and Business

School of Business

Department of Management Science & Technology

Master of Science in Business Analytics

Program:	Full-time
Quarter:	3rd (Spring Quarter)
Course:	Big Data Mining
Assignment:	Neo4j Graph Database
Students (Registration №):	f2822202, f2822217

Table of Contents

Big Data Mining Assignment 2

Introduction

In this assignment, we will try to create a Neo4j Graph Database from three dataset files containing information about a subset of the high energy physics theory citation network, which contains authors, articles, journals, and citations between articles. We will proceed with a basic description of our graph model, then we will explain the Cypher commands used for the import of the data to create the forementioned database and finally we will report the code used for answering the required queries.

Graph Model Description

The graph model that we created, that represents a subset of the citation network in the field of high energy physics theory, is presented in the following figure (Figure 1). For the creation of this graph, Neo4j graph database and its commands (Cypher Query Language) were used.

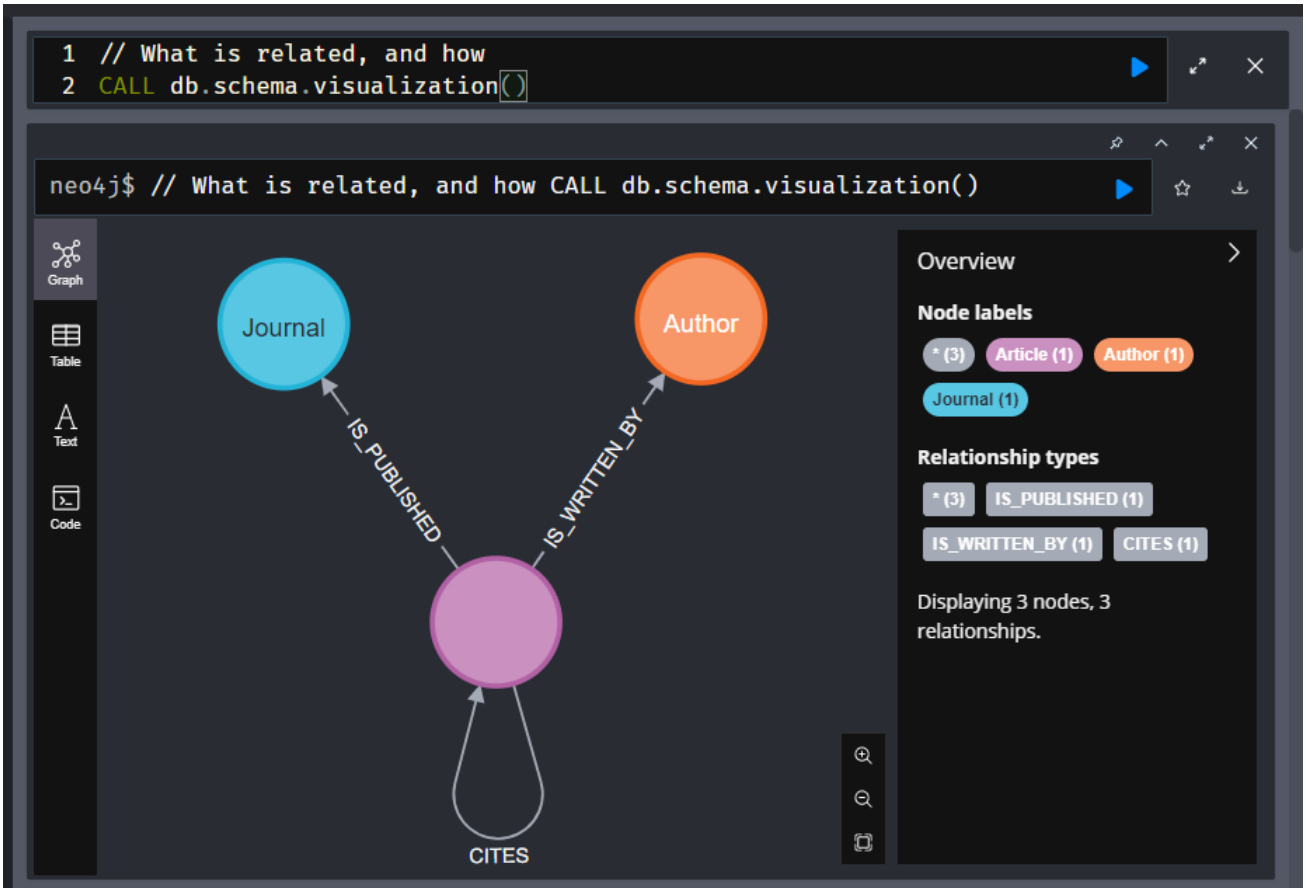

Figure 1 - Graph model visualization (neo4j)

Let's break down the model step by step:

Nodes:

- Author: Represents authors who have written articles in the forementioned field. Each Author node represents a unique author and is connected to one or more Article nodes. It has a name property, representing the author's name in string format.
- Article: Represents individual research articles in the field. Each Article node contains the following information as properties: ID (long data type), title (string), year (long), and abstract (string) and is

connected to the corresponding Author node who authored it. Additionally, each Article node may be connected to a Journal node via the IS_PUBLISHED relationship, if the corresponding Article was published in a specific venue.

- Journal: Represents journals where the articles are published. Each Journal node has a name property, representing its name in string format.

Relationships:

- IS_WRITTEN_BY: Represents the relationship between an Author and an Article. It indicates the author(s), who has written the article. The relationship is directed from the Article node to the Author node.
- IS_PUBLISHED: Represents the relationship between an Article and a Journal. It indicates that the article is published in the specified journal. The relationship is directed from the Article node to the Journal node.
- CITES: Represents the relationship between two Article nodes. It indicates that one article cites another article. The relationship is directed from the citing to the cited article.

In the following figures (Figure 2 and Figure 3), we can view the properties of the graph's nodes and relationships. We can observe that there do not exist properties for any relationship of the graph.

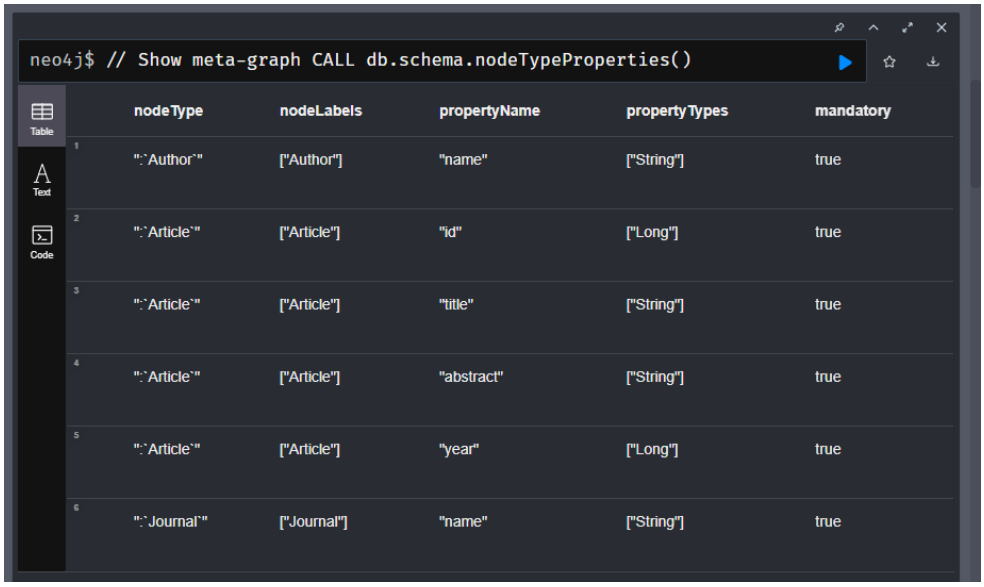

Figure 2 - Properties of the Graph's Nodes

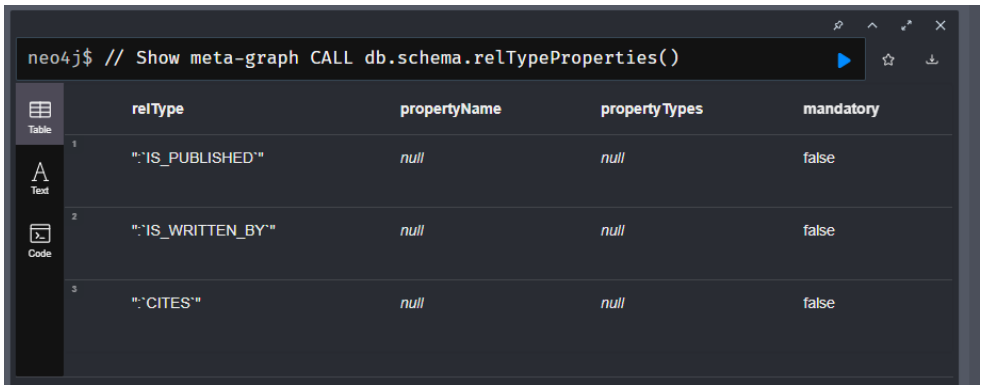

Figure 3 - Properties of the Graph's Relationships

In summary, the graph model represents a network where Authors are connected to the Articles which they have written via the IS_WRITTEN_BY relationship. Each Article is associated with a Journal through the IS_PUBLISHED relationship. Additionally, Articles can cite other Articles through the CITES relationship. This graph model can be used to explore relationships between authors, articles, and journals, analyze collaborations among authors, track citations between articles, and gain insights into the structure and dynamics of the high energy physics theory citation network.

File Import

But before proceeding to querying the graph database, we must firstly import the files to it. The dataset files that needed to be imported are:

- ArticleNodes.csv: Contains info about Article nodes (id, title, year, journal and abstract).
- AuthorNodes.csv: Contains article id and the name of the author(s).
- Citations.csv: Contains info about citations between articles (articleId,--[Cites]->, articleId).

We, firstly, moved the files to the Neo4j's database import folder. Because all files were CSV files, we used the LOAD CSV command of Cypher to load them to the database. Then, we checked if the LOAD CSV command correctly reads the data, and we specified the FIELDTERMINATOR command as '\t' to indicate tab-separated values for the Citations file. We then changed all data types that we did not want to be inserted as a string data type e.g., article's ID and article's year and trimmed the string data types. All forementioned verifications of how the LOAD CSV command will import the data are presented in Figure 4.

```
// DATASET: verify how LOAD CSV sees the data

// Count data rows in ArticleNodes.csv (no headers)
LOAD CSV FROM 'file:///ArticleNodes.csv' AS row
RETURN COUNT(row);

// Count data rows in AuthorNodes.csv (no headers)
LOAD CSV FROM 'file:///AuthorNodes.csv' AS row
RETURN COUNT(row);

// View data rows in Citations.csv
LOAD CSV FROM 'file:///Citations.csv' AS row
FIELDTERMINATOR '\t'
RETURN row
LIMIT 3;

// Change data types and view top 3 data rows in ArticleNodes.csv
LOAD CSV FROM 'file:///ArticleNodes.csv' AS row
WITH toInteger(row[0]) AS articleId, trim(row[1]) AS articleTitle, toInteger(row[2]) AS articleYear, trim(row[3]) as
articleJournal, trim(row[4]) as articleAbstract
RETURN articleId, articleTitle, articleYear, articleJournal, articleAbstract
LIMIT 3;

// Change data types and view top 3 data rows in AuthorNodes.csv
LOAD CSV FROM 'file:///AuthorNodes.csv' AS row
WITH toInteger(row[0]) AS articleId, trim(row[1]) AS authorName
RETURN articleId, authorName
LIMIT 3;

// Change data types and view top 3 data rows in Citations.csv
LOAD CSV FROM 'file:///Citations.csv' AS row
FIELDTERMINATOR '\t'
WITH toInteger(row[0]) AS articleId, toInteger(row[1]) AS articleCitation
RETURN articleId, articleCitation
LIMIT 3;
```

Figure 4 - Verifications of how Cypher's LOAD CSV command reads the data

Then, we set constraints and indexes on specific properties (where necessary) to optimize performance and significantly speed up data import and query execution (Figure 5). We created an index for the Article's title property to improve the performance of the queries that involve searching articles by their titles and constraints (automatically indexes were created also) to enforce uniqueness on the following properties:

- Author's name (authorNameConstraint)
- Article's ID (articleIdConstraint)
- Journal's name (journalNameConstraint)

```
// GRAPH DATA MODEL CONSTRAINTS AND INDEXES

// Author's unique name property
CREATE CONSTRAINT authorNameConstraint FOR (au:Author) REQUIRE au.name IS UNIQUE;

// Article's unique id property and index on title property
CREATE CONSTRAINT articleIdConstraint FOR (ar:Article) REQUIRE ar.id IS UNIQUE;
CREATE INDEX FOR (ar:Article) ON (ar.title);

// Journal's unique name property
CREATE CONSTRAINT journalNameConstraint FOR (j:Journal) REQUIRE j.name IS UNIQUE;
```

Figure 5 - Graph DB's Constraints and Indexes

Finally, we imported the data using the LOAD CSV command, creating nodes and relationships in the graph database with the use of CREATE, MERGE and MATCH clauses (Figure 6). For each dataset we executed a single command:

- For the ArticleNodes.csv file, two variants of the command are provided to create the Article and Journal nodes, together with the IS_PUBLISHED relationship (if present) that connects them. When the Article's Journal column has a missing value, only the Article node should be created. The first variant uses a FOREACH loop to handle cases where an article has a missing value (NULL) in the journal where it was published. The second variant splits the command into two separate sections to handle the NULL case. We used the former variant because it was faster.
- The AuthorNodes.csv file is imported by matching the corresponding Article node based on its ID and creating the Author node connected to the corresponding Article node via the IS_WRITTEN_BY relationship.
- The Citations.csv file is imported by matching the Article nodes based on its ID on the `articleId` and `articleCitation` columns and creating the CITES relationship between them.

```
// IMPORT DATA USING "LOAD CSV"

// Import from ArticleNodes.csv
// Faster (1482 ms)
LOAD CSV FROM 'file:///ArticleNodes.csv' AS row
WITH toInteger(row[0]) AS articleId, trim(row[1]) AS articleTitle, toInteger(row[2]) AS articleYear, trim(row[3]) as articleJournal, trim(row[4]) as
articleAbstract
CREATE (ar:Article {id: articleId, title: articleTitle, year: articleYear, abstract: articleAbstract})
FOREACH (
x IN CASE WHEN articleJournal IS NULL THEN [] ELSE [1] END |
    MERGE (j:Journal {name: articleJournal})
    CREATE (ar)-[:IS_PUBLISHED]->(j)
);
// Slower (1185 + 443 = 1628 ms)
//LOAD CSV FROM 'file:///ArticleNodes.csv' AS row
//WITH toInteger(row[0]) AS articleId, trim(row[1]) AS articleTitle, toInteger(row[2]) AS articleYear, trim(row[3]) as articleJournal, trim(row[4]) as
articleAbstract
//WHERE articleJournal IS NOT NULL
//MERGE (j:Journal {name: articleJournal})
//CREATE (ar:Article {id: articleId, title: articleTitle, year: articleYear, abstract: articleAbstract})
//CREATE (ar)-[:IS_PUBLISHED]->(j);
//LOAD CSV FROM 'file:///ArticleNodes.csv' AS row
//WITH toInteger(row[0]) AS articleId, trim(row[1]) AS articleTitle, toInteger(row[2]) AS articleYear, trim(row[3]) as articleJournal, trim(row[4]) as
articleAbstract
//WHERE articleJournal IS NULL
//CREATE (ar:Article {id: articleId, title: articleTitle, year: articleYear, abstract: articleAbstract});

// Import from AuthorNodes.csv (1386 ms)
LOAD CSV FROM 'file:///AuthorNodes.csv' AS row
WITH toInteger(row[0]) AS articleId, trim(row[1]) AS authorName
MATCH (ar:Article {id: articleId})
MERGE (au:Author {name: authorName})
CREATE (au)<-[:IS_WRITTEN_BY]-(ar);

// Import from Citations.csv (3933 ms)
LOAD CSV FROM 'file:///Citations.csv' AS row
FIELDTERMINATOR '\t'
WITH toInteger(row[0]) AS articleId, toInteger(row[1]) AS articleCitation
MATCH (ar1:Article {id: articleId}), (ar2:Article {id: articleCitation})
CREATE (ar1)-[:CITES]->(ar2);
```

Figure 6 - Import data using Cypher's LOAD CSV, CREATE, MERGE, and MATCH clauses

Finally, we performed a set of checks to verify that the graph database is comprised of 29,555 articles; 15,420 authors; 836 journals; and 352,807 citations, as shown in Figure 7.

```
// CHECKS

// 29555 Articles
MATCH (n:Article) RETURN COUNT(n);

// 15420 Authors
MATCH (n:Author) RETURN COUNT(n);

// 836 Journals
MATCH (n:Journal) RETURN COUNT(n);

// 352807 citations
MATCH (n:Article)-[r:CITES]->() RETURN COUNT(r);
```

Figure 7 - Checks that data are correctly imported

Queries

After the creation of the graph database, we answered the following queries:

Top 5 Authors with the Most Citations

This query retrieves authors (au) who are connected to other papers through the CITES relationship (r) i.e., papers that cite the papers which the author has written. It returns the author names (au.name) and the count of citations. The results are ordered in descending order of the number of citations and limited to the top 5 authors, as shown in Figure 8.

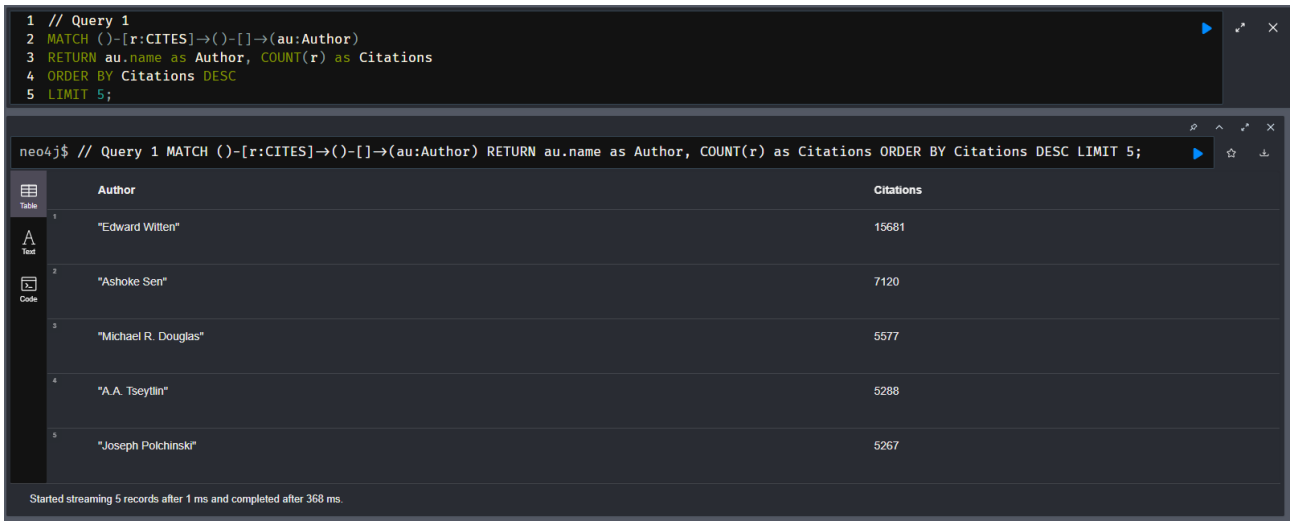

Figure 8 - Query 1 code and results

Top 5 Authors with the Most Collaborations

This query finds authors (au1) who have collaborated with different authors (au2) through the IS_WRITTEN_BY relationship i.e., authors who have collaborated for authoring a paper. It ensures that the same author is not counted more than once as a collaboration. The author's names (au1.name) and the count of distinct collaborations are returned. The results are ordered in descending order of the number of collaborations and limited to the top 5 authors, as shown in Figure 9.

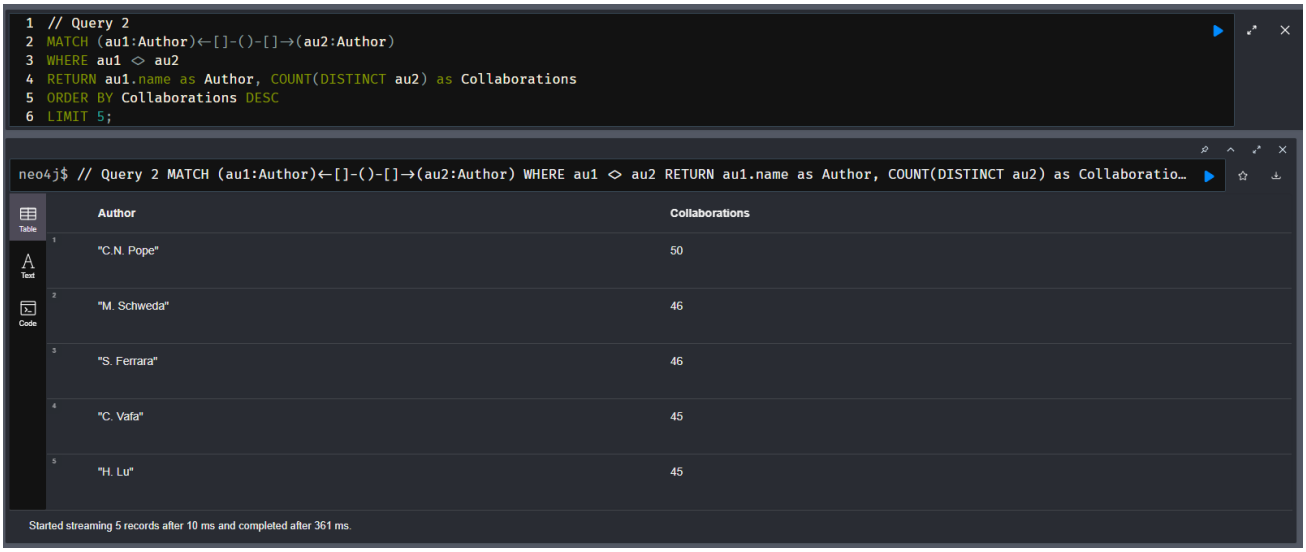

Figure 9 - Query 2 code and results

Author with the Most Papers without Collaborations

This query matches authors (au) who are connected to articles (ar) through the IS_WRITTEN_BY relationship (r) and does not exist any other IS_WRITTEN_BY relationship from this article (ar) to other authors i.e., there exists only one author that has written this article. The result is limited to the author with the highest count of papers, whose name and count of papers is returned, as shown in Figure 10. Note: From this query, we have found that there exists twice a specific paper in the dataset, named "Orientifold Limit of F-theory Vacua", which is written by the same author in the same year (1997) with different IDs (9702165 and 9709159). The difference across all fields of the forementioned papers is in the journal (venue) that was published and in the abstract. So, if we used `COUNT(DISTINCT ar.title) AS Papers` instead of `COUNT(ar) AS Papers`, we would have found 54 papers, because we would have been counting the distinct titles instead of the article nodes. However, we decided to continue with the former case instead of the latter one, because that was the data provided and we were also not certain if there was a difference in the publication to the journals (venues).

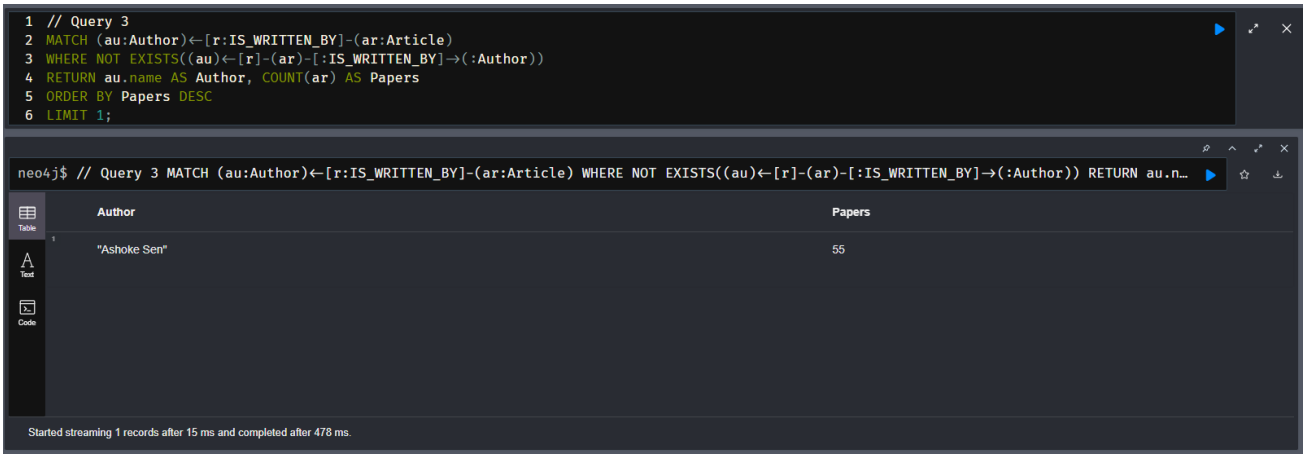

Figure 10 - Query 3 code and results

Author with the Most Papers in 2001

If the question is referring to the author (au) who wrote the most papers in 2001, we used the first query to count the number of papers connected to the author through the IS_WRITTEN_BY relationship. The result is limited to the author with the highest count of papers, whose name (au.name) and count of papers are returned, as shown in Figure 11.

If the question is referring to the author (au) who wrote the most papers in 2001 that were published to a journal, we used the second query to include a matching condition with the Journal node. Again, the result is limited to the author with the highest count of papers, as shown in Figure 12.

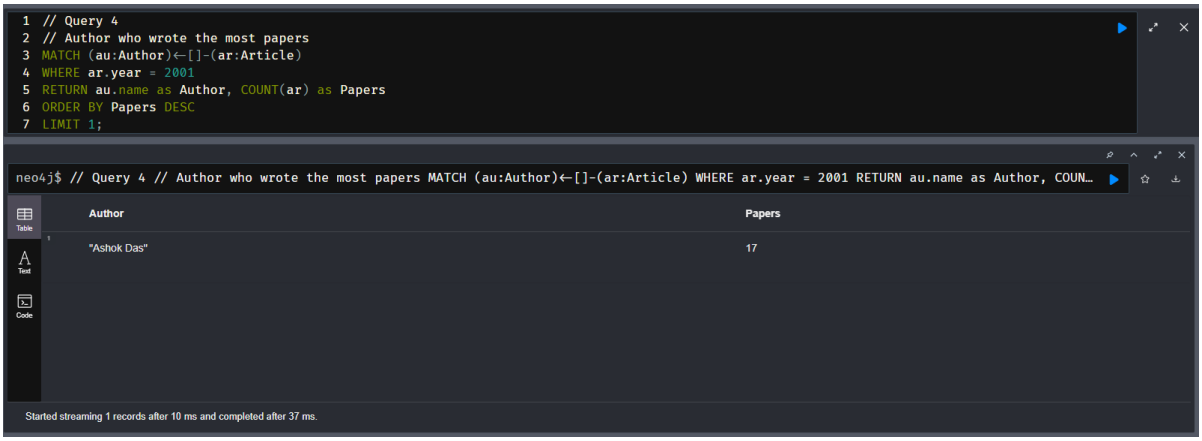

Figure 11 - Query 4 code and results of the Author who wrote the most Articles

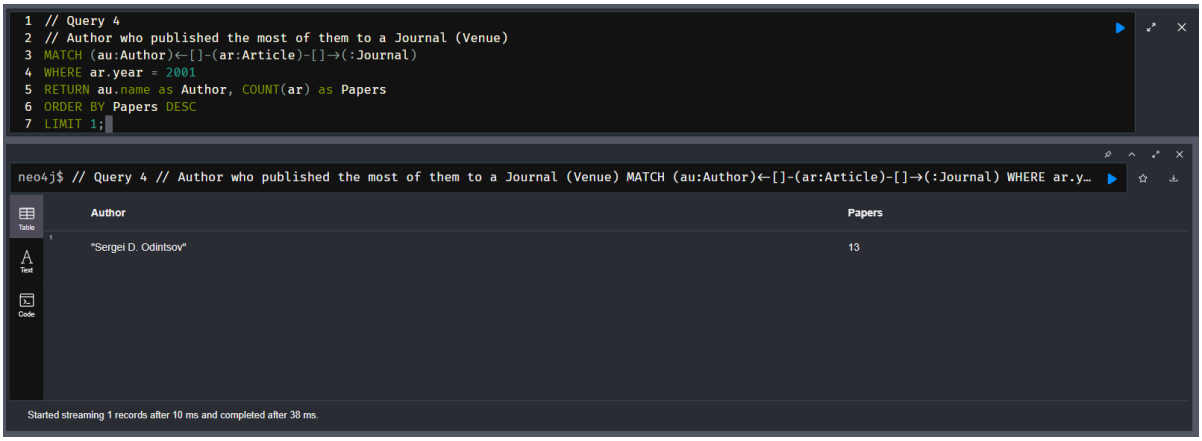

Figure 12 - Query 4 code and results of the Author who has the most published Articles

Journal with the Most Papers about "Gravity" in 1998

This query finds articles (ar) connected to journals (j) through the IS_PUBLISHED relationship. It uses a regular expression to match the title of the articles containing the word "gravity" (case-insensitive) for the articles written in 1998. The name of the journal (j.name) and the count of papers are returned. The result is limited to the journal with the highest count of papers, as shown in Figure 13.

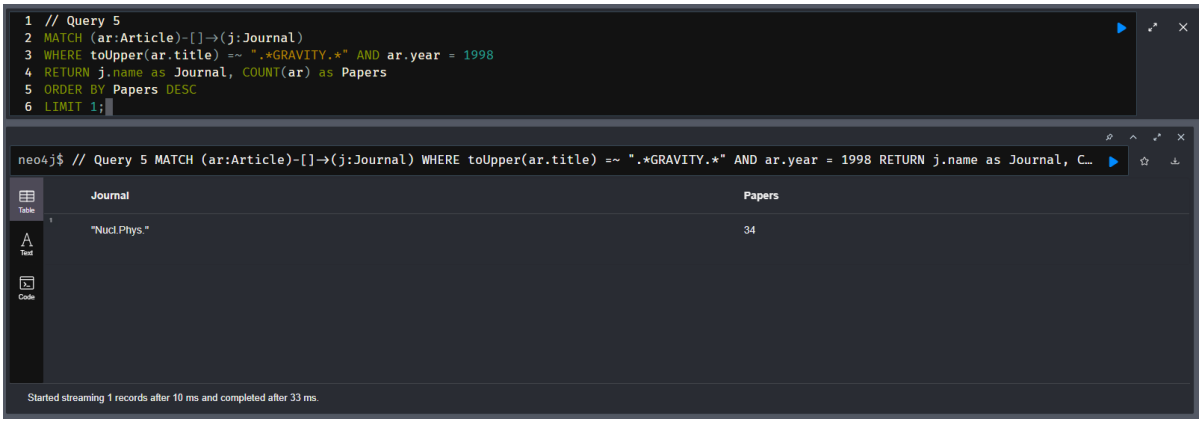

Figure 13 - Query 5 code and results

Top 5 Papers with the Most Citations

This query retrieves cited articles (ar) connected to other (citing) articles through the CITES relationship (r). It returns the titles of the articles (ar.title) and the count of citations. The results are ordered in descending order of the number of citations and limited to the top 5 papers, as shown in Figure 14.

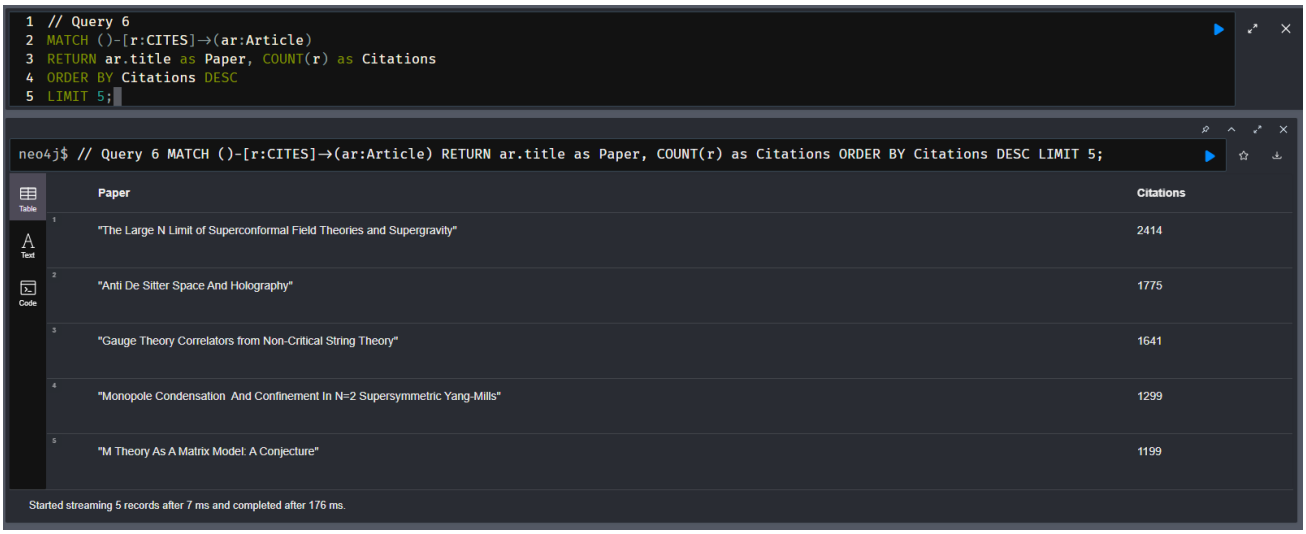

Figure 14 - Query 6 code and results

Papers that use "Holography" and "Anti de Sitter"

This query matches authors (au) connected to articles (ar) through the IS_WRITTEN_BY relationship. It uses regular expressions to match the abstracts of the articles containing the words "holography" and "anti de sitter" (case-insensitive). Also, for the "anti de sitter" term, articles including underscores, non-word characters, non-digit characters, periods, asterisks, or whitespace characters that can occur zero or more times between the three words of the term are being captured e.g., Anti_de_Sitter, (Anti) de Sitter, etc. The query returns the titles of the matched articles and the author(s) who authored them, as shown in Figure 15.

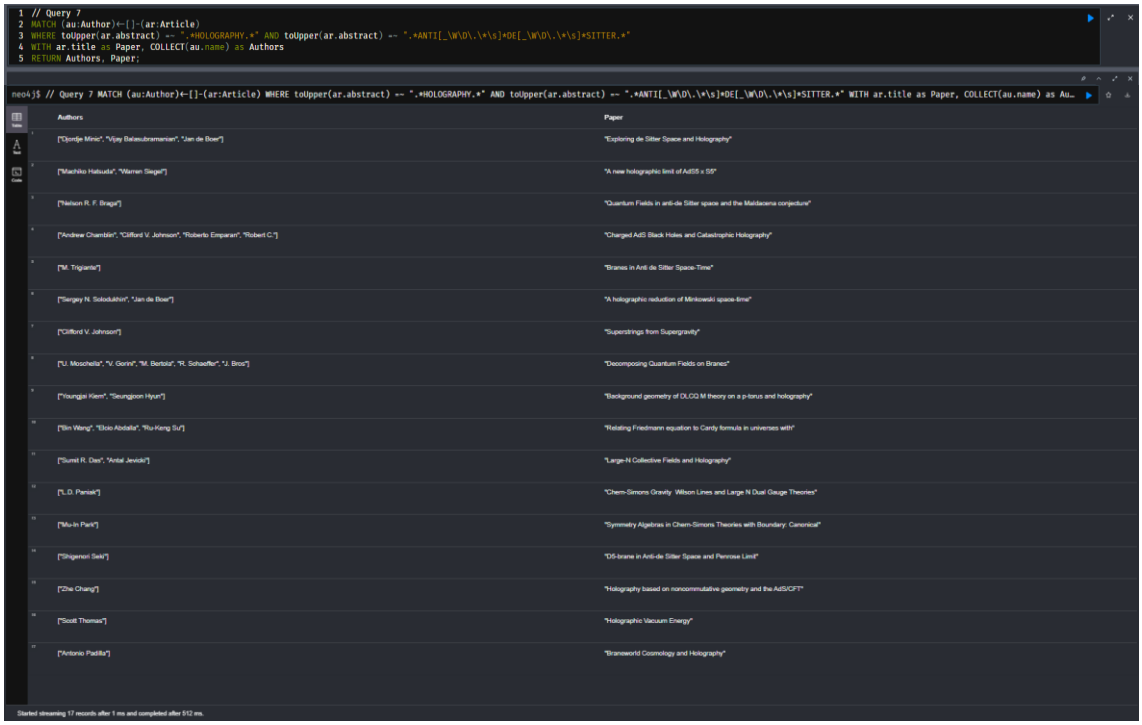

Figure 15 - Query 7 code and results

Shortest Path between "C.N. Pope" and "M. Schweda"

This query finds the shortest path (p) between two authors (au1 and au2) using any type of edges. Since any type of relationship is accepted, any type of node is also accepted in the path between the two authors. It returns the path (p) and the length of the path, as shown in Figure 16.

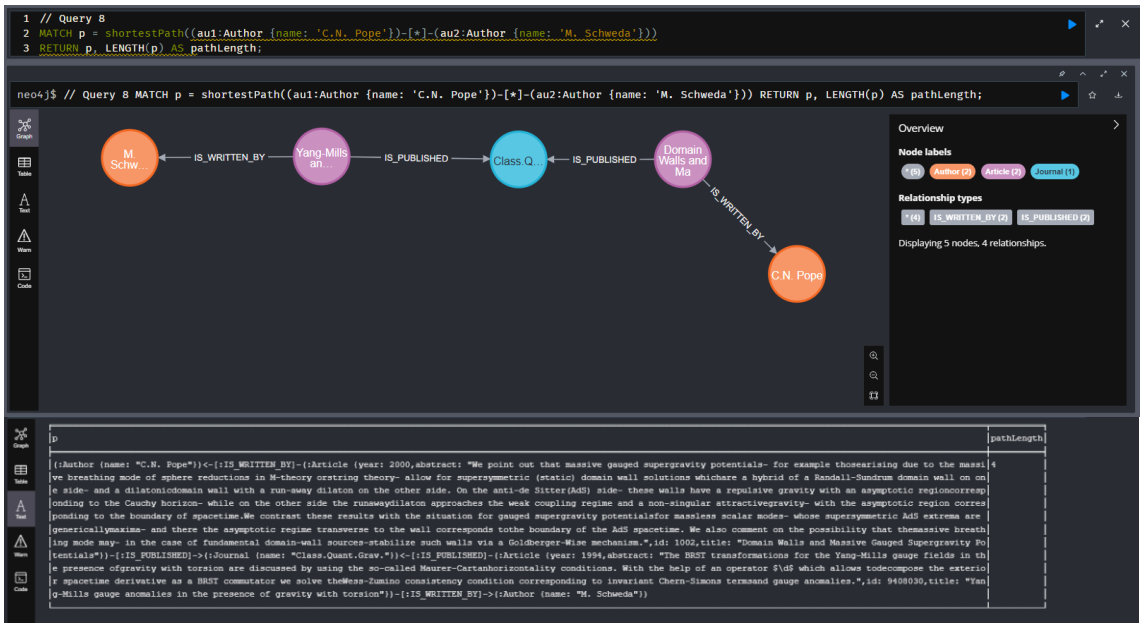

Figure 16 - Query 8 code and results

Shortest Path between "C.N. Pope" and "M. Schweda" using only Author-Paper Edges

This query is similar to the previous one but restricts the path to only edges between authors and papers (IS_WRITTEN_BY relationship). Since only one type of relationship is accepted, only the types of nodes that are connected through this relationship (Authors and Articles) are accepted. This is an extra limitation compared to the previous query and thus we expect to find a path of equal or greater length than the previous query's path length. It returns the shortest path (p) between the two authors and its length, as shown in Figure 17.

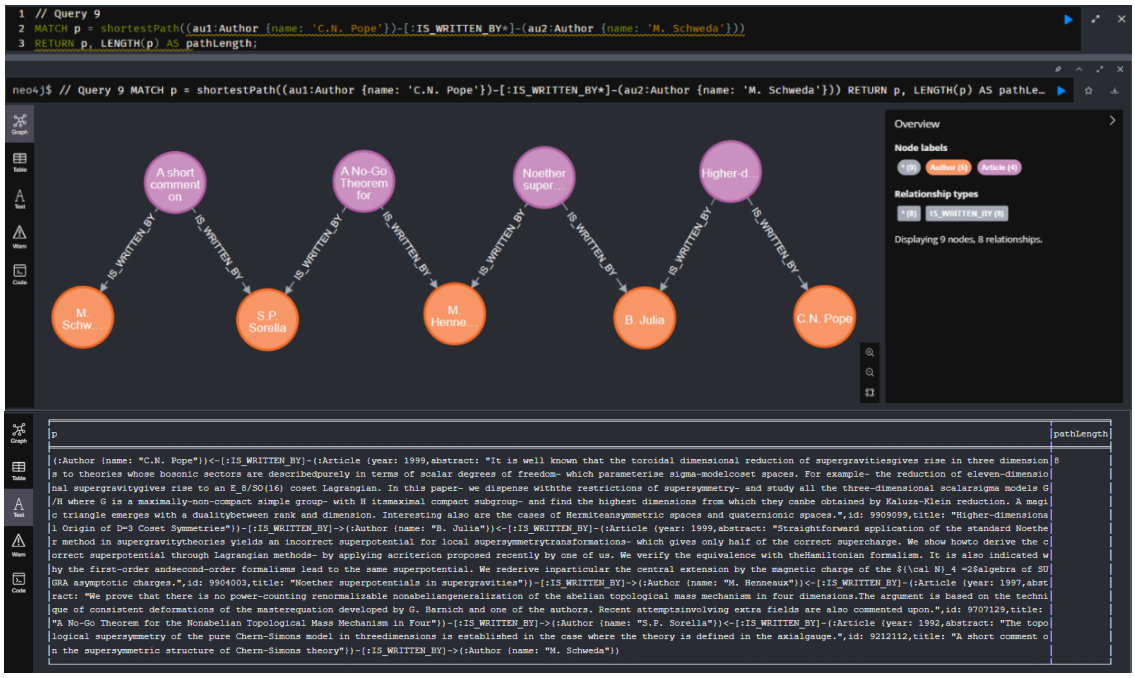

Figure 17 - Query 9 code and results

Authors with Shortest Path Lengths > 25 from "Edward Witten"

This query matches the starting author (startAuthor) with the name "Edward Witten". It finds the shortest paths (p) between the starting author and other authors (au) using only edges between authors and articles (IS_WRITTEN_BY relationship). The query returns the author's names (au.name), the length of each path, and the titles of the papers along each path. Only paths with a length greater than 25 are returned, as shown in Figure 18.

Planning the shortest paths in Cypher can lead to different query plans depending on the predicates that need to be evaluated. Internally, Neo4j will use a fast bidirectional breadth-first search algorithm if the predicates can be evaluated whilst searching for the path. If the predicates need to inspect the whole path before deciding on whether it is valid or not, this fast algorithm cannot be relied on to find the shortest path, and Neo4j may have to resort to using a slower exhaustive depth-first search algorithm to find the path. This query has to check that the whole path follows the predicate (in our case path length greater than 25) before we know if it is valid or not. However, the inclusion of the WITH - WHERE clause means that the query plan will not include the fallback to the slower exhaustive search algorithm. Instead, any paths found by the fast algorithm will subsequently be filtered (Neo4j, 2023), leading to the same result but with the use of the faster algorithm. The following query runs for approximately 5.5 seconds.

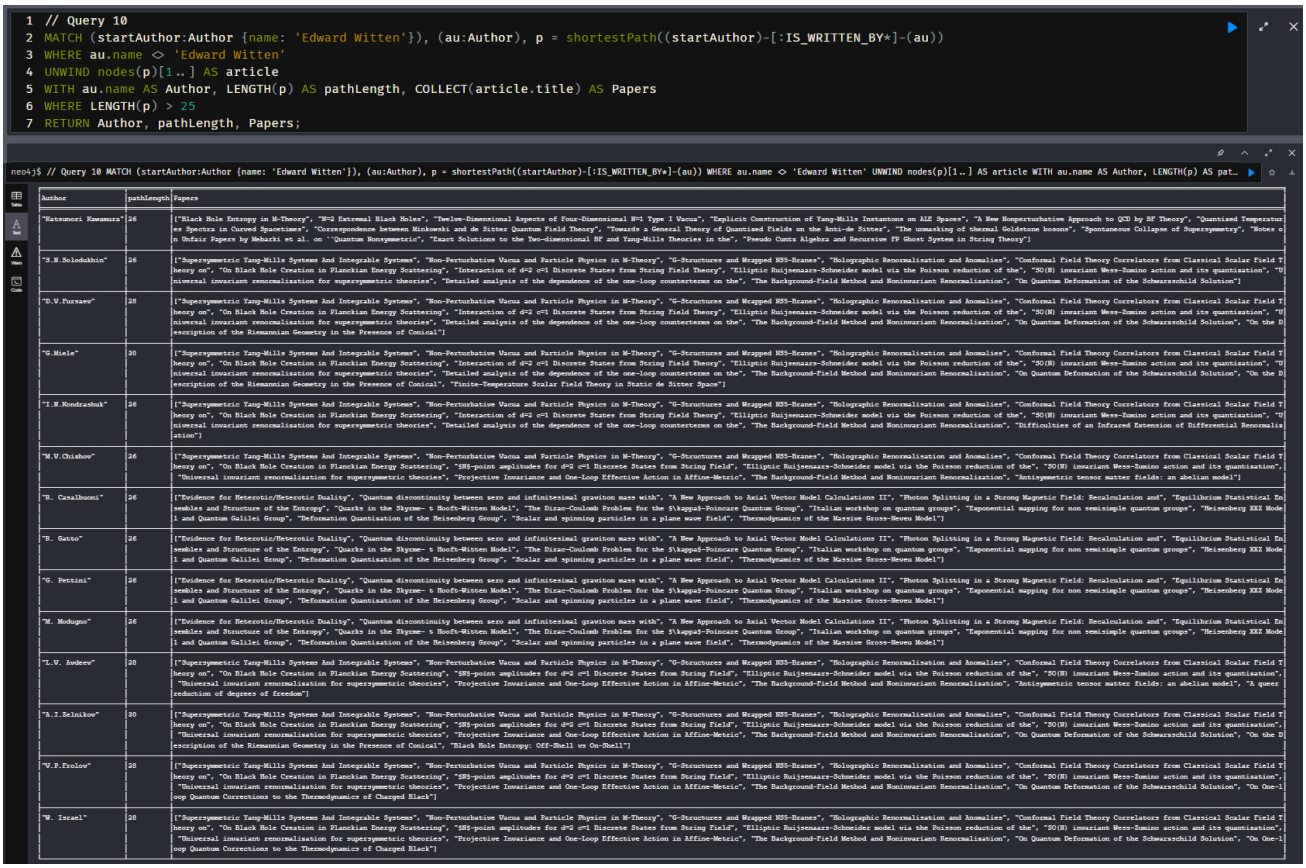

Figure 18 - Query 10 code and results

References

Neo4j. (2023, June 12). *Neo4j Docs*. Retrieved from Neo4j: https://neo4j.com/docs/cypher-manual/current/execution-plans/shortestpath-planning/