**6CS371: Advanced Database System Lab**

**Neo4j Graph Database**

**PRN:** 21510042  **Name:** Omkar Rajesh Auti

**Title: Neo4j Graph Database**

Designing and implementing a Graph Database System for Research Papers using Neo4j

**Objective/Aim:**

The objective of this assignment is to create a database system using Neo4j that efficiently manages research papers, their authors, citations, and classifications. The aim is to develop a user-friendly Python desktop application capable of performing various queries on the database, such as checking citations between papers and displaying the full classification of a paper.

**Introduction:**

In today's interconnected world, traditional relational databases sometimes struggle to efficiently manage complex relationships between data. This is where graph databases come into play. Graph databases, unlike traditional relational databases, excel at managing interconnected data by representing entities as nodes and relationships between them as edges in a graph structure.

Now, consider the domain of academic research papers. Papers are authored by multiple individuals, cite other papers, and belong to various classifications. Managing this intricate web of relationships efficiently is crucial for researchers, institutions, and academic platforms. This is where the significance of designing an efficient database system tailored for research papers becomes apparent.

Efficiently managing research papers involves handling not just the papers themselves but also their authors, citations, and classifications. Traditional databases might struggle to model and query such complex relationships effectively. Graph databases, on the other hand, excel in this domain due to their inherent ability to represent and traverse interconnected data.

**Theory/Algorithms:**

Neo4j is one of the leading graph database management systems, renowned for its capabilities in handling graph data efficiently. It stores data in the form of nodes, which represent entities, and relationships, which represent connections between entities. Neo4j provides a flexible and intuitive query language called Cypher, which allows users to perform complex queries on the graph data.

In the context of the research papers database scenario, modeling the data using Neo4j involves defining nodes for papers, authors, and classifications, and relationships between them. For example, a paper node may have attributes such as title, publication year, and abstract, while an author node may have attributes like name and affiliation. Relationships between nodes, such as "authored by" or "cites," capture the connections between papers and authors or between papers themselves.

Loading data into Neo4j can be achieved through various methods, such as using Cypher queries to create nodes and relationships manually or importing data from external sources like CSV files. Additionally, Neo4j provides tools and libraries for bulk data import, making it efficient to handle large datasets.

Querying the Neo4j database involves using Cypher queries to retrieve information from the graph. Cypher offers powerful capabilities for traversing the graph structure and performing operations like filtering, matching patterns, and aggregating data. This allows users to extract valuable insights from the interconnected research paper data efficiently.

In summary, Neo4j's capabilities in handling graph data, coupled with its flexible query language and efficient data loading mechanisms, make it an ideal choice for modeling and querying the research papers database scenario.

**Procedure:**

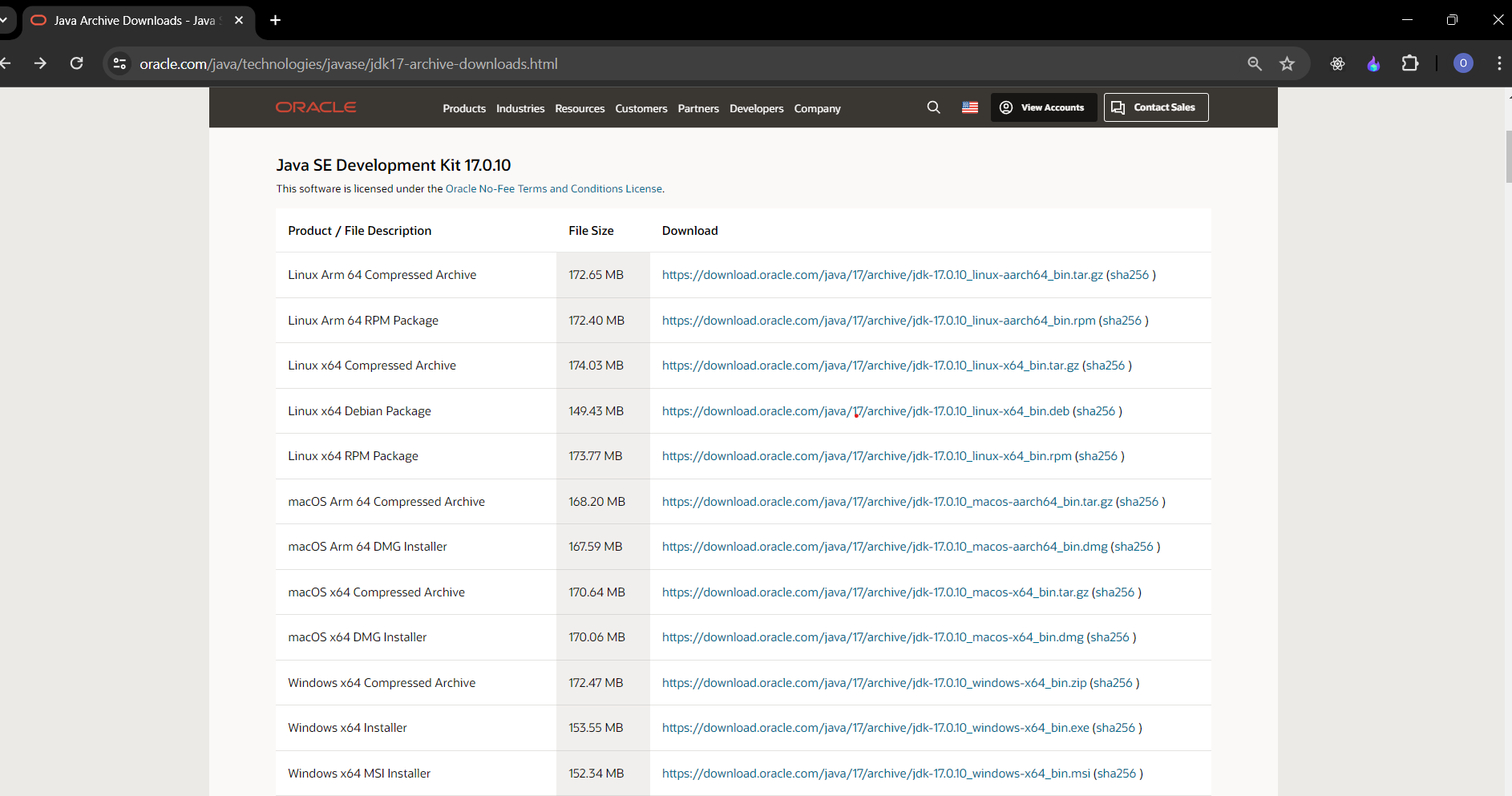
1. Designing the Database: First, we model the research papers scenario by defining node types for papers, authors, and classifications, and relationships between them.
2. Loading Data: Next, we download the raw data from the Cora Research Paper Classification Project and import it into Neo4j Data Browser.
3. Developing the Desktop Application: We create a Python-based desktop application that connects to the Neo4j database and allows users to perform queries. The application is designed to answer questions like whether paper A cites paper B directly or indirectly and to display the full classification of a paper.

**Download Neo4j:**

**Installation Steps:**

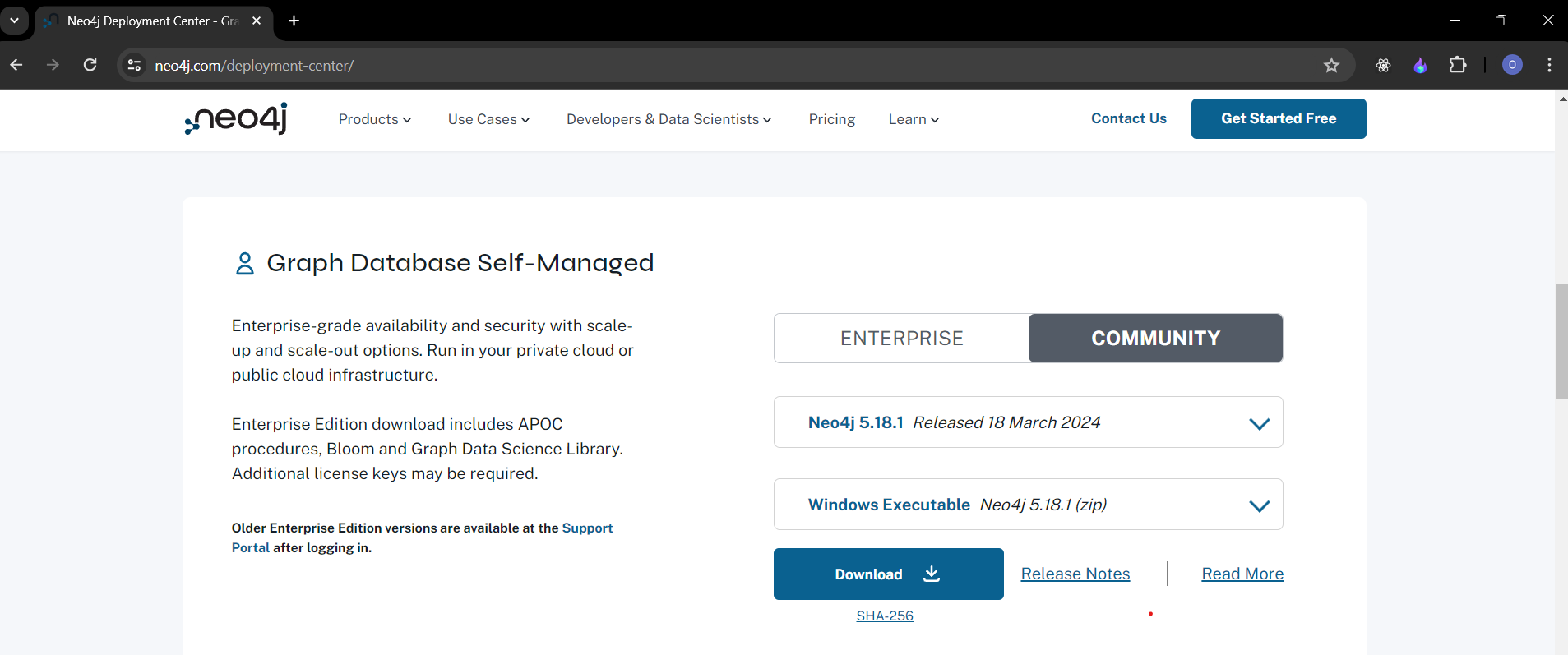
**a. Install** **Java SE Development Kit 17.0.10**

**Link** - <https://www.oracle.com/java/technologies/javase/jdk17-archive-downloads.html>



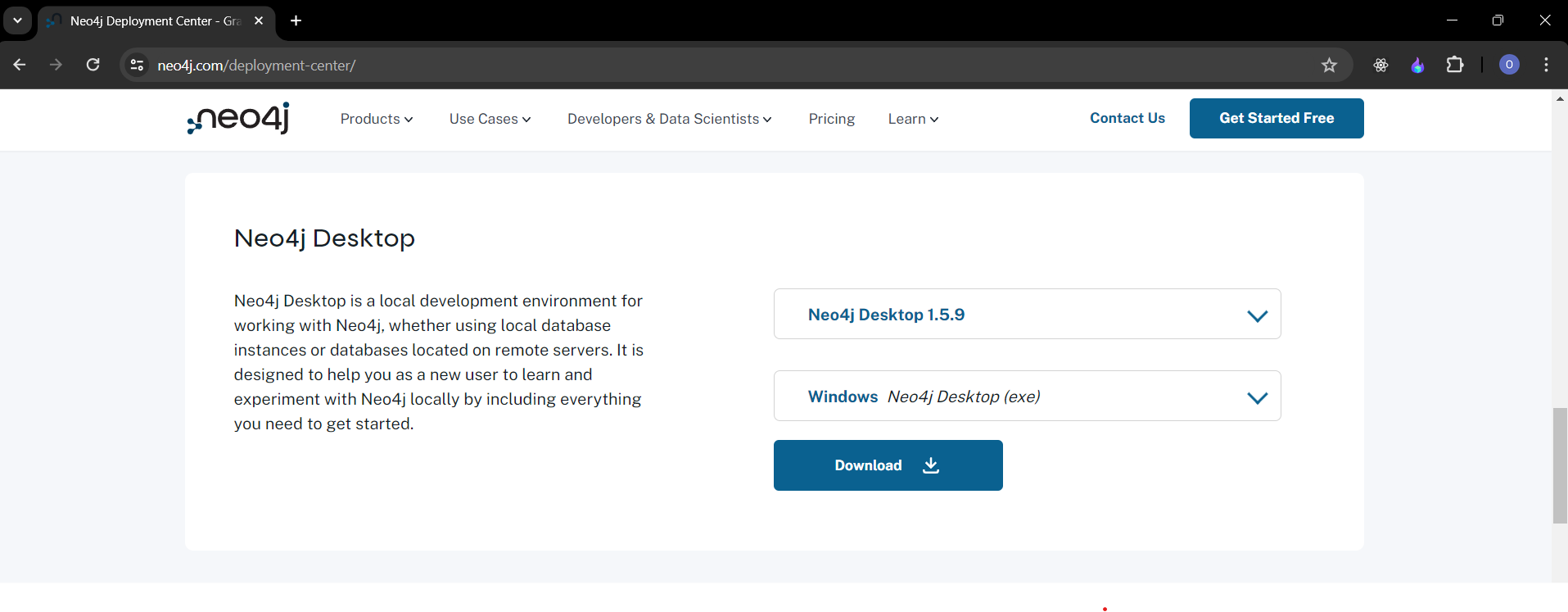
**b. Download the neo4j community server (.zip for windows)**

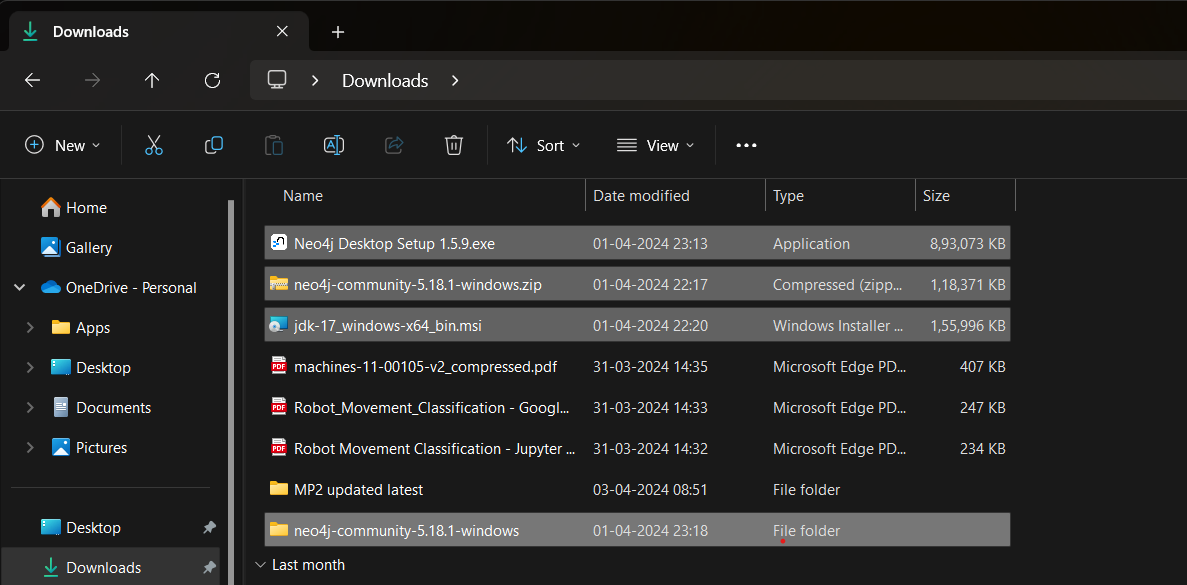
**Link-** <https://neo4j.com/deployment-center/>



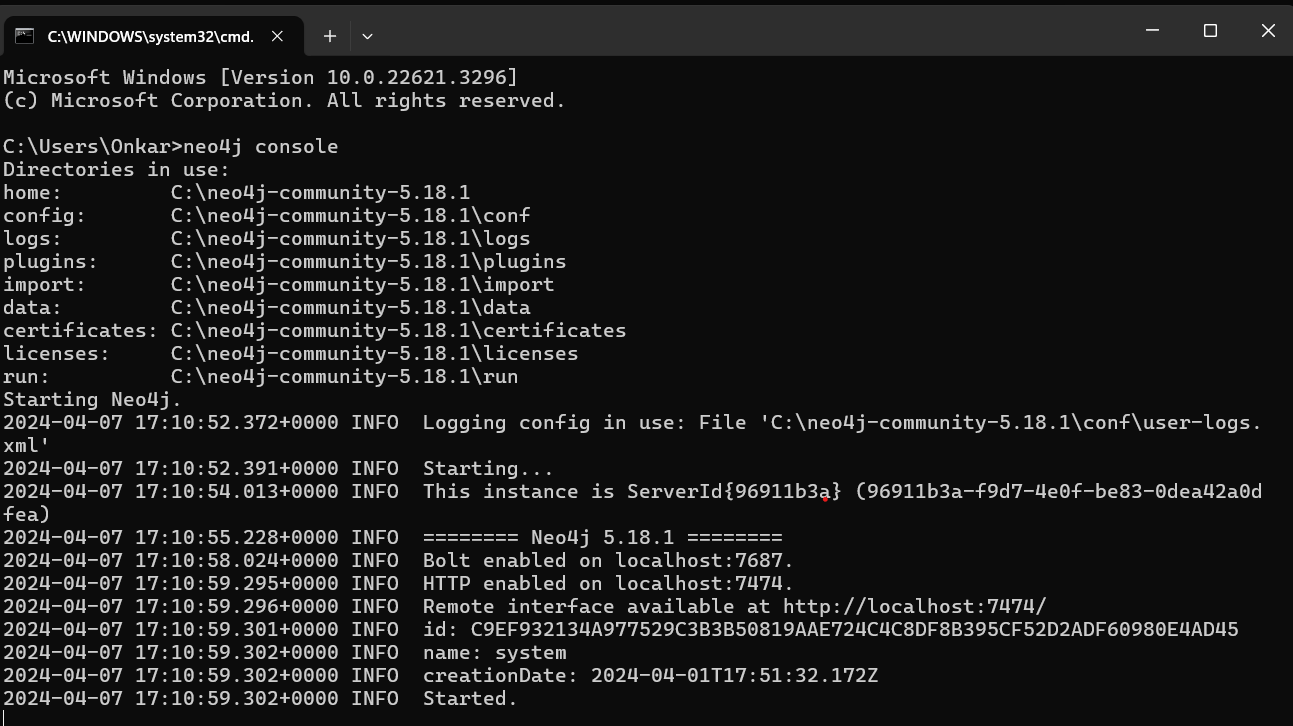
**c. Download the neo4j desktop**

**Link-** <https://neo4j.com/deployment-center/>



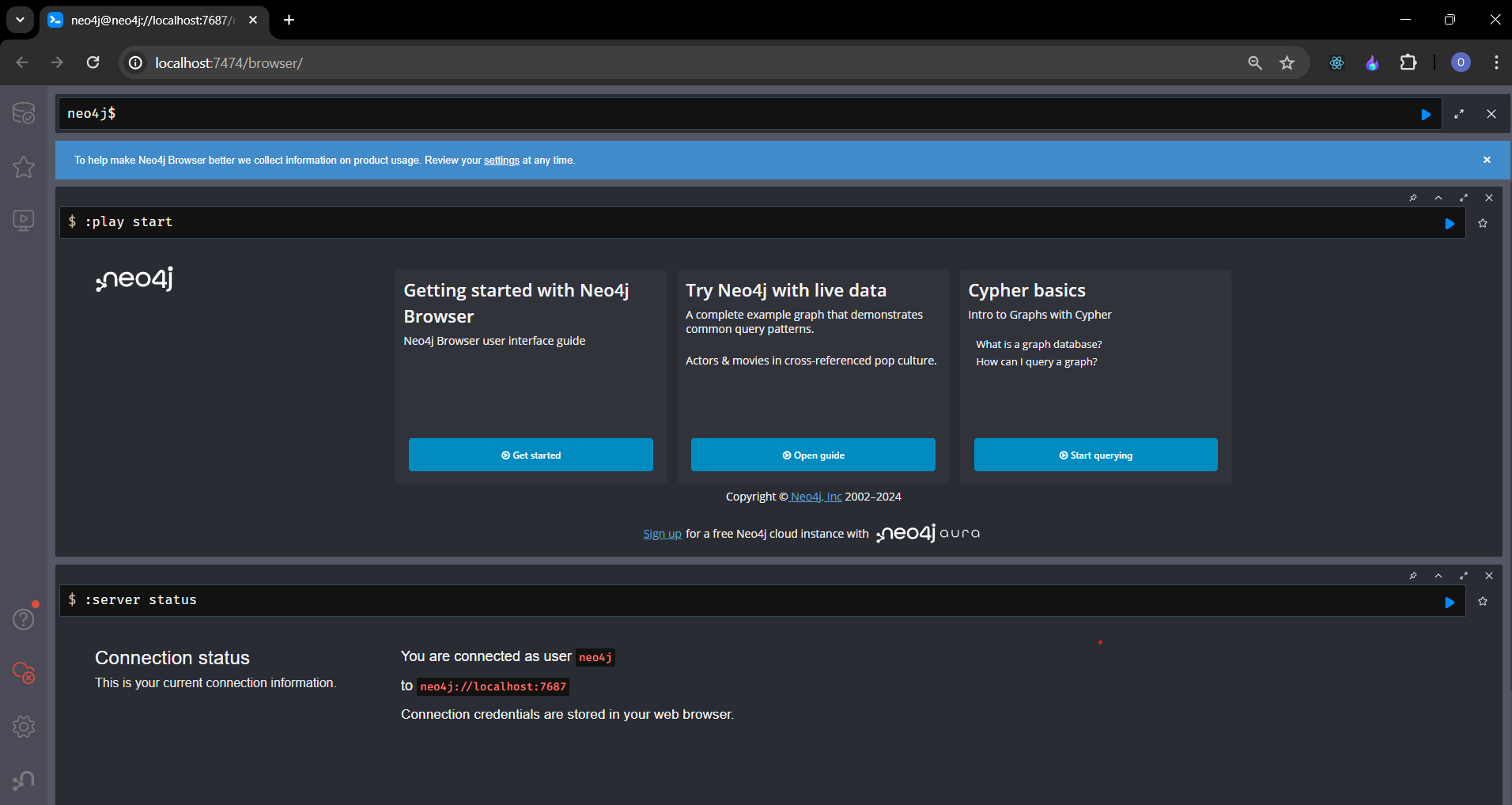


1. **Start Neo4j Server**



**2. Access Neo4j Browser**

a. Access neo4j browser from <http://localhost:7474/browser/>



**3. Login and Configure**

a. For the first time, the following credentials are to be use:

**Username:** neo4j

**Password:** neo4j

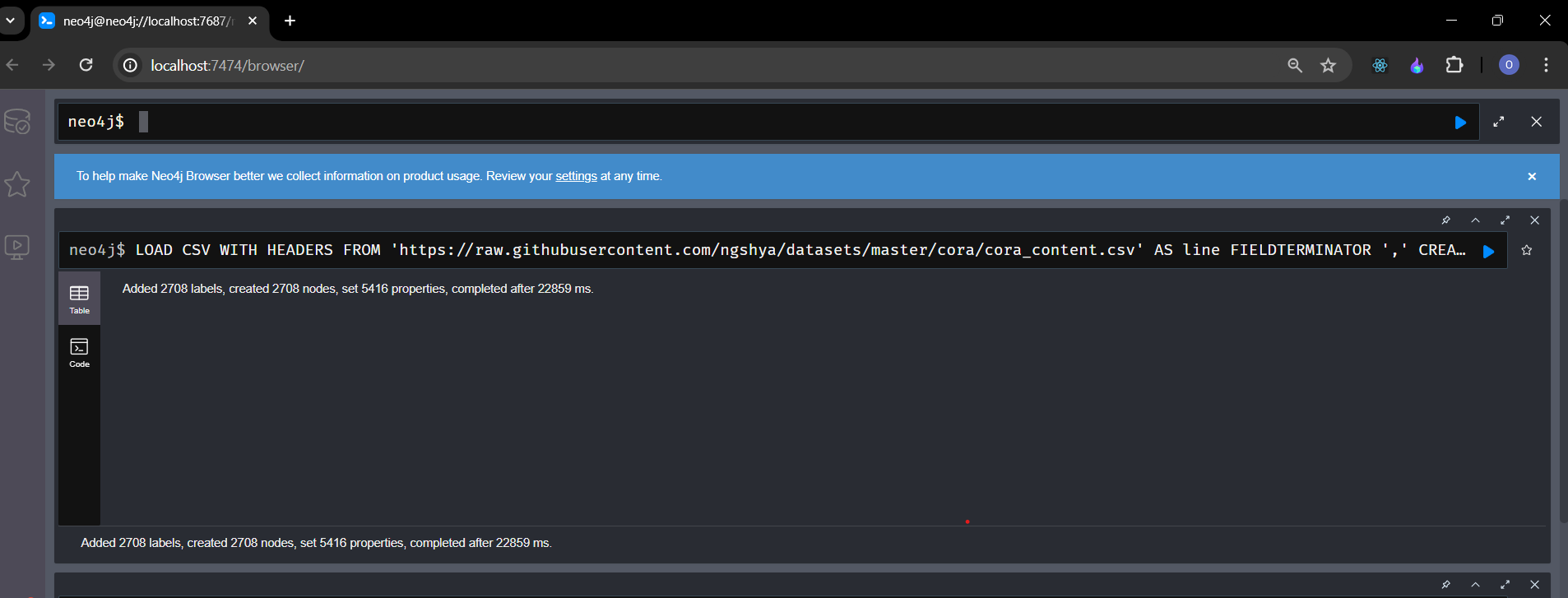
b. On the first login, password has to be changed.

**4. Load data for research paper**

**a. Load the data using the following commands:**

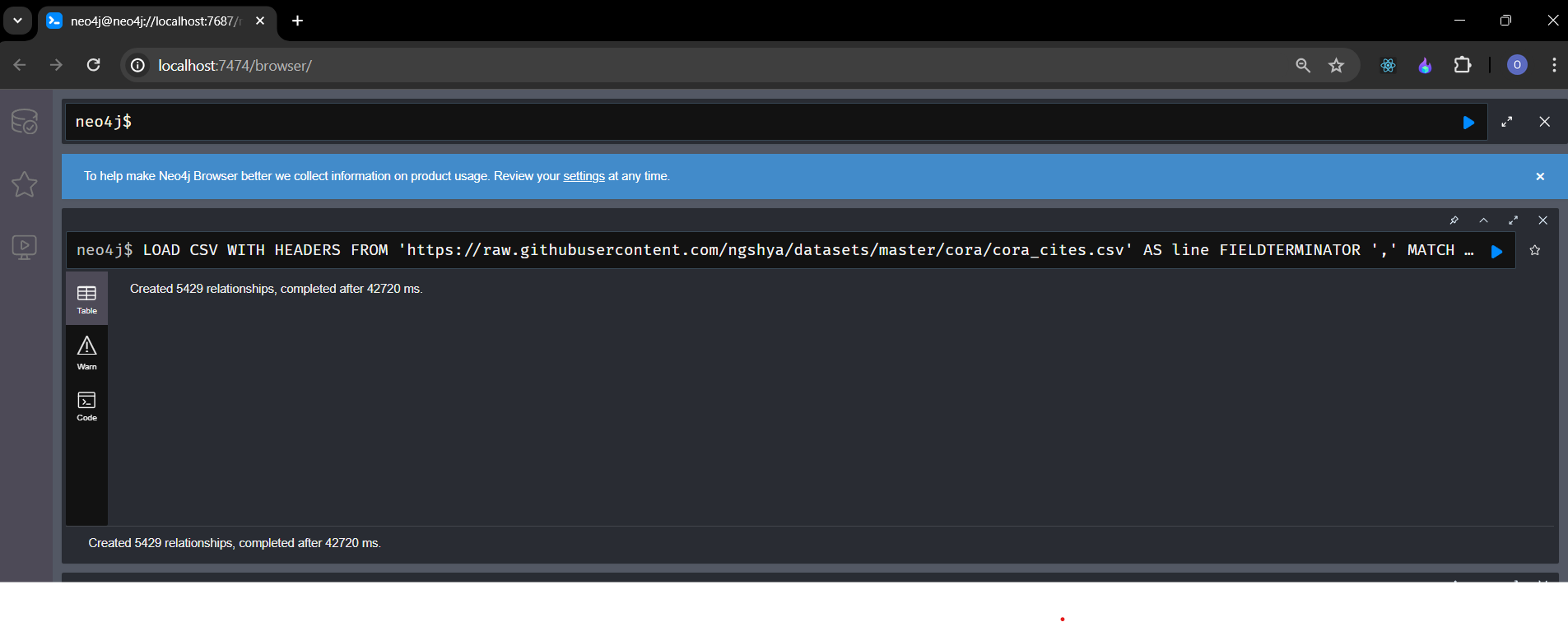
1. **Load the nodes:**

LOAD CSV WITH HEADERS FROM 'https://raw.githubusercontent.com/ngshya/datasets/master/cora/cora\_content.csv' AS line FIELDTERMINATOR ',' CREATE (:Paper {id: line.paper\_id, class: line.label})

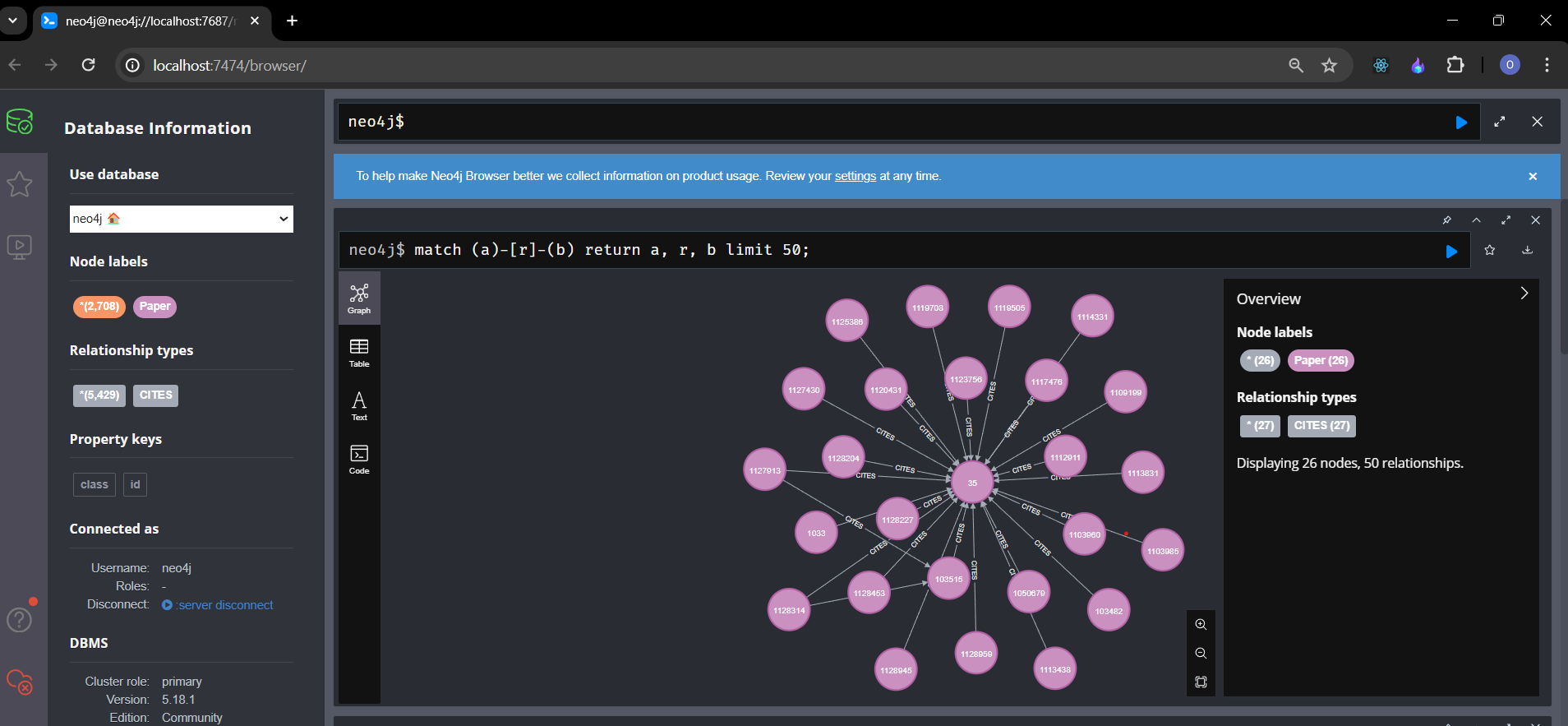


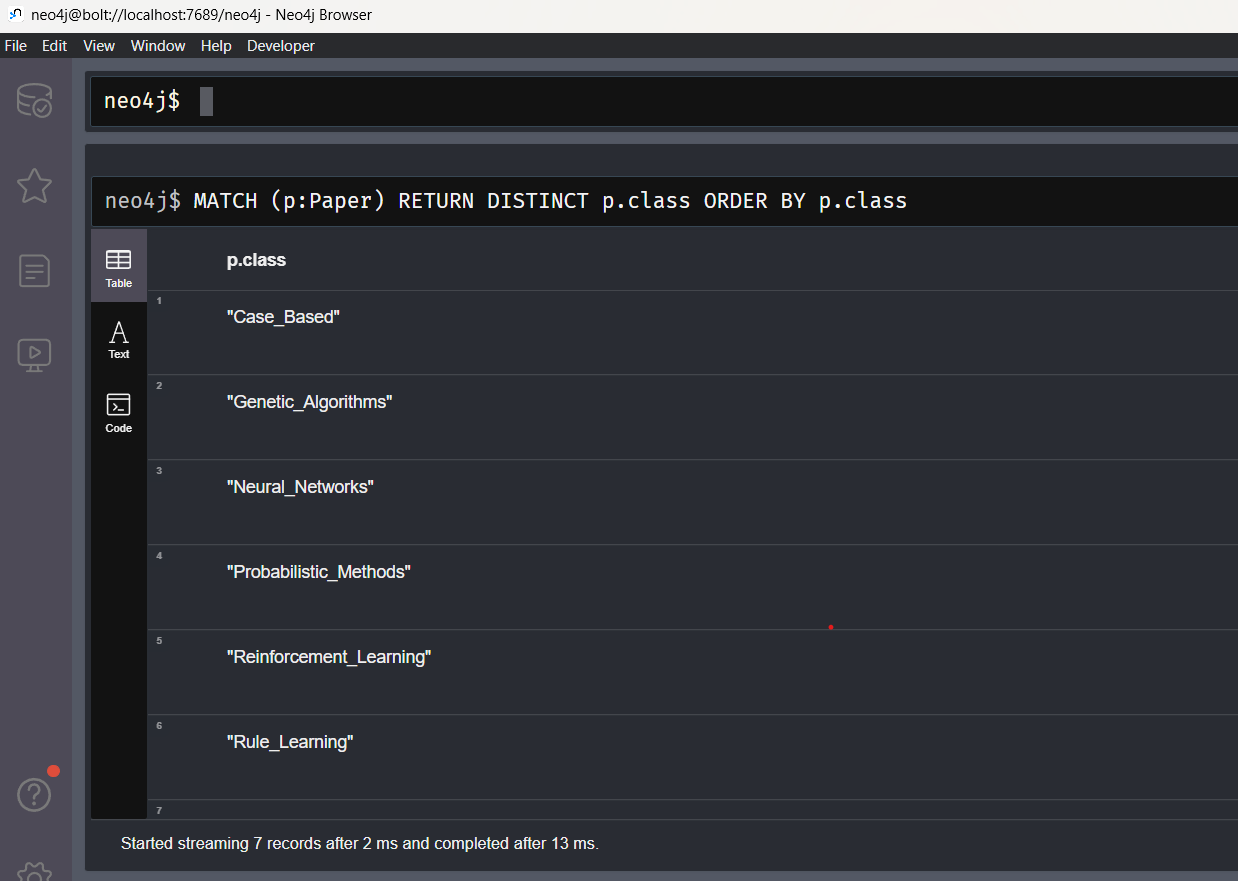
1. **Load the relationships:**

LOAD CSV WITH HEADERS FROM 'https://raw.githubusercontent.com/ngshya/datasets/master/cora/cora\_cites.csv' AS line FIELDTERMINATOR ',' MATCH (citing\_paper:Paper {id: line.citing\_paper\_id}),(cited\_paper:Paper {id: line.cited\_paper\_id}) CREATE (citing\_paper)-[:CITES]->(cited\_paper)

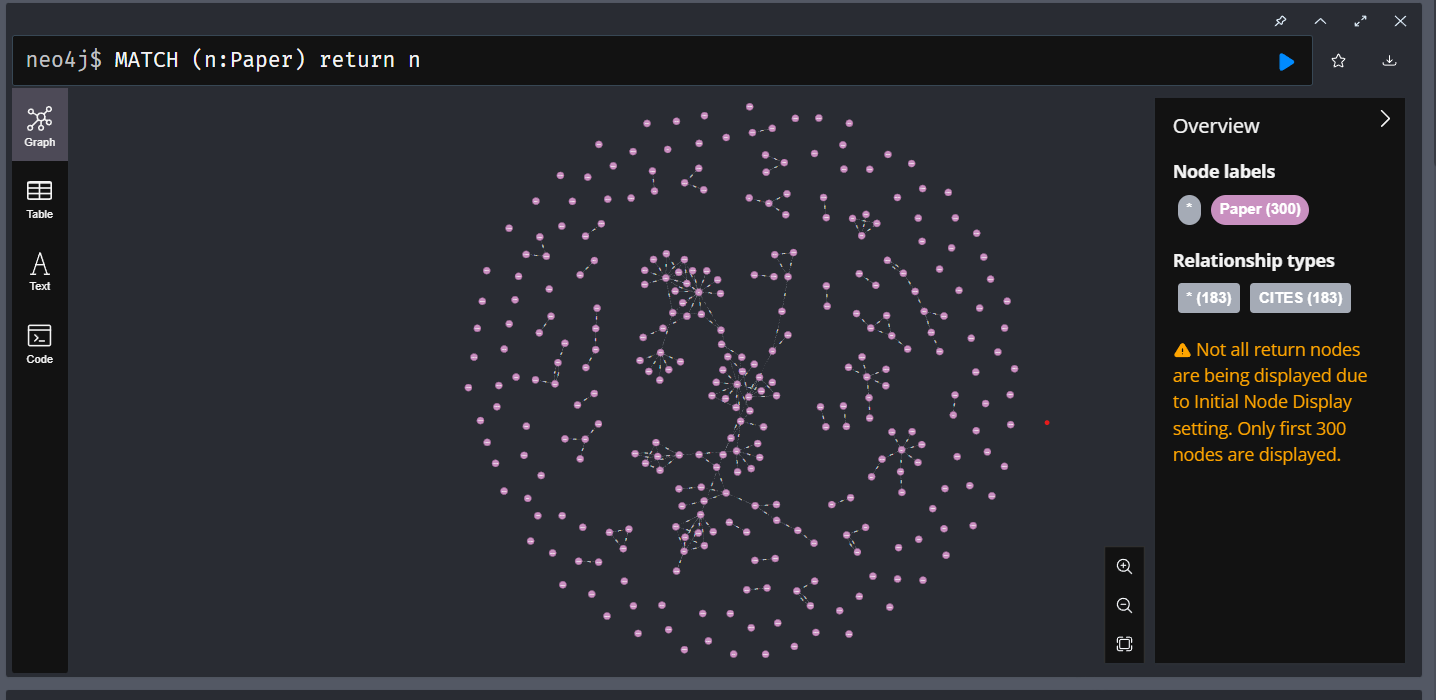
****









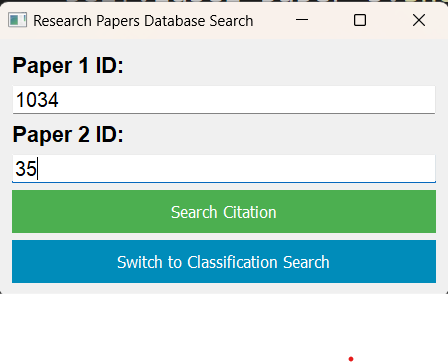


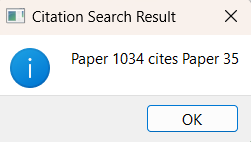
1. **Create a python desktop application using tkinter and neo4j library:**

For this application, we have created a new user with the username `neo4j` and password `admin123`.

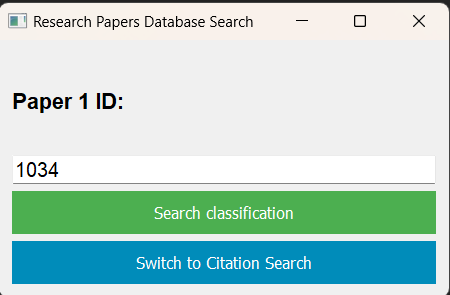
**Result / Observations:**

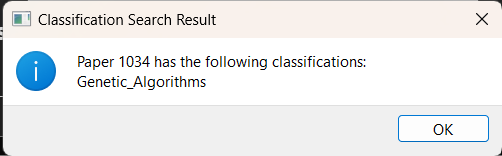
1. **Citation Search:**





1. **Classification Search:**





**Using Neo4j Console vs Installing Neo4j Service:**

**Neo4j Console:**

1. **Starting Neo4j:**
   * **Command:** **neo4j console**
   * **Importance:** Launches Neo4j in the foreground from the command line, providing real-time logs and feedback. Useful for debugging and development purposes where immediate feedback is needed.
2. **Stopping Neo4j:**
   * **Command:** Press **Ctrl + C** in the console window where Neo4j is running.
   * **Importance:** Halts the Neo4j instance gracefully, allowing it to shut down all processes and release resources properly.
3. **Accessing Neo4j Browser:**
   * **URL:** <http://localhost:7474/>
   * **Importance:** Provides a web interface for interacting with the Neo4j database, allowing users to execute queries, visualize data, and manage the database.

**Installing Neo4j Service:**

1. **Installing the Service:**
   * **Command:** **neo4j windows-service install**
   * **Importance:** Installs Neo4j as a Windows service, allowing it to run in the background even after closing the command prompt or logging out of the system.
2. **Starting the Service:**
   * **Command:** **net start Neo4j**
   * **Importance:** Initiates the Neo4j service, enabling it to run in the background. The service automatically starts when the system boots up.
3. **Stopping the Service:**
   * **Command:** **net stop Neo4j**
   * **Importance:** Halts the Neo4j service, terminating all associated processes and releasing system resources. This is the recommended way to stop Neo4j when it's installed as a service.

**Summary:**

* **Neo4j Console:** Ideal for development and debugging purposes, providing real-time feedback and easy access to logs.
* **Neo4j Service:** Suitable for production environments or scenarios where Neo4j needs to run continuously in the background, ensuring persistent availability and reliability.

By understanding and utilizing these commands effectively, users can manage Neo4j instances according to their specific needs, whether for development, testing, or production purposes.

The following two queries are different in terms of the relationships they match and the patterns they represent in the Neo4j database. Let's break down each query and discuss their importance:

1. **Direct Citation Check Query:**

**MATCH p=(:Paper{id:"{paper\_a\_id}"})-[r:CITES]->(:Paper{id:"{paper\_b\_id}"}) RETURN p**

* This query checks if one paper directly cites another paper.
* It looks for a direct connection labeled as "CITES" between the two papers.
* This is helpful for finding out if one paper directly mentions or refers to another paper.

**2. Indirect Citation Check Query:**

**MATCH path = (p1:Paper {id: $paper\_a\_id})-[:CITES\*]->(p2:Paper {id: $paper\_b\_id})**

**RETURN relationships(path) AS citations**

**LIMIT 1**

* This query checks if there's a chain of citation connections between two papers, even if they're not directly connected.
* It allows for any number of other papers in between.
* This is useful for discovering papers that influence each other indirectly through a series of citations.

- `MATCH path =`: This part initiates a pattern matching operation where `path` is a variable representing the path between two papers.

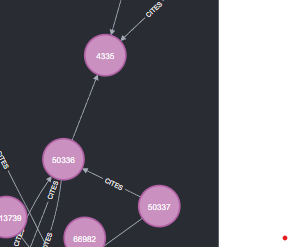
- `(p1:Paper {id: $paper\_a\_id})`: This specifies the starting node labeled as `Paper` with a property `id` equal to the value of `$paper\_a\_id`.

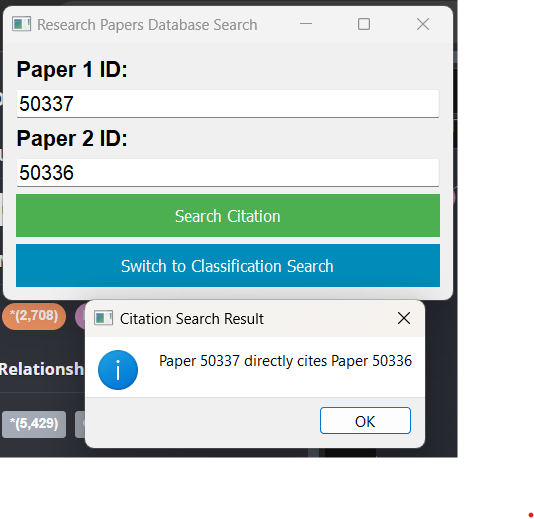
- `[:CITES\*]`: This part denotes a variable-length relationship pattern denoted by `\*`, which means there can be zero or more `CITES` relationships between `p1` and `p2`.

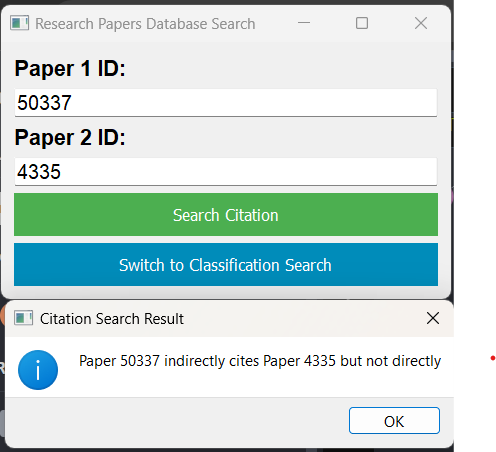
- `(p2:Paper {id: $paper\_b\_id})`: This specifies the ending node labeled as `Paper` with a property `id` equal to the value of `$paper\_b\_id`.

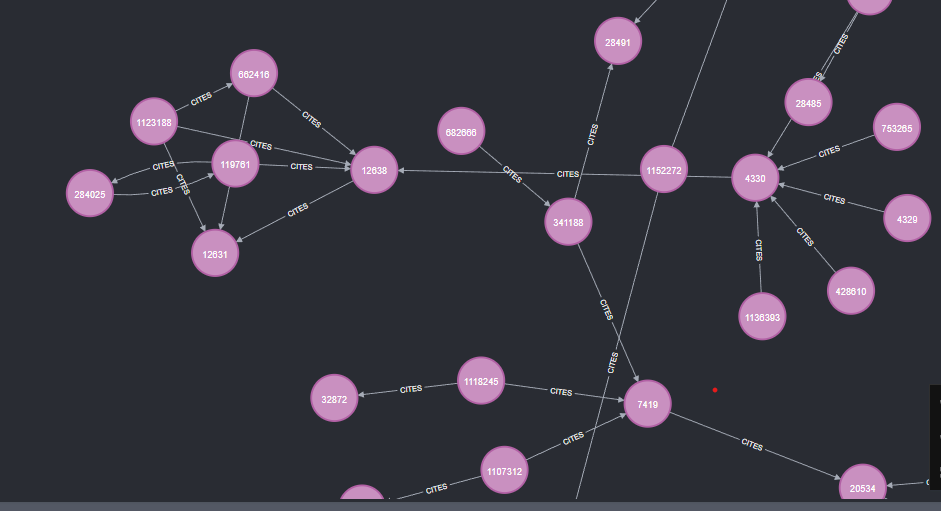
- `RETURN relationships(path) AS citations`: This returns the relationships along the matched path as `citations`. Since we're using `relationships(path)`, it will return all relationships along the path, indicating indirect citations between the papers.

- `LIMIT 1`: This limits the result to only one path, effectively limiting the result to one set of citations. However, since the relationships are returned, it might contain multiple relationships if there are multiple paths.









**Conclusion:**

In conclusion, this assignment demonstrates the effectiveness of using Neo4j for managing research paper data. By leveraging its graph database capabilities, we have designed a system that efficiently handles complex relationships between papers, authors, and classifications. The Python desktop application provides users with an intuitive interface to interact with the database, enabling seamless querying and exploration of research paper data.

**References:**

1. Neo4j Documentation:

<https://neo4j.com/docs/>

2. Cora Research Paper Classification Project**:**

[**http://people.cs.umass.edu/~mccallum/data.html**](http://people.cs.umass.edu/~mccallum/data.html)

1. Python Neo4j driver:

<https://pypi.org/project/neo4j/>

1. PyQt5 tutorial:

<https://www.tutorialspoint.com/pyqt5/index.htm>