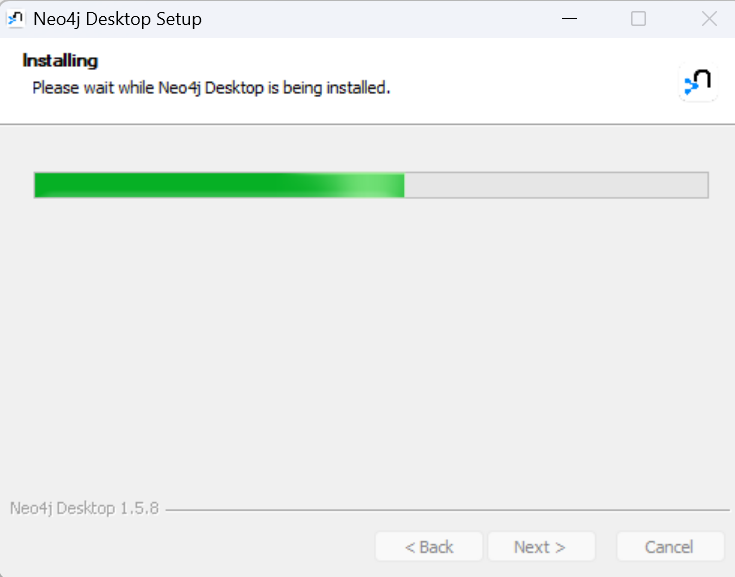
**ADS ASSIGNMENT - 11**

**Consider the “Research Papers Database” scenario as follows :**

**The research papers have authors (often more than one). Most papers have a classification (what the paper is about). The classifications form a hierarchy in several levels (for example, the classification “Databases” has the sub-classifications “Relational” and “Object-Oriented”). A paper usually has a list of references, which are other papers. These are called citations.**

**1. Design/model the graph database using Neo4j for above scenario.**

First install neo4j desktop



Nodes:

Paper: represents a research paper with attributes such as title, abstract, publication date, DOI, and paper ID.

Author: represents an author of a research paper with attributes such as name, affiliation, and email.

Classification: represents a classification of a research paper with attributes such as name and classification ID.

Citation: represents a citation of a research paper.

Relationships:

PUBLISHED\_BY: connects an Author node to a Paper node indicating who published the paper.

WRITTEN\_BY: connects a Paper node to one or more Author nodes indicating who wrote the paper.

HAS\_CLASSIFICATION: connects a Paper node to a Classification node indicating the classification of the paper.

BELONGS\_TO: connects a Classification node to another Classification node indicating the hierarchical relationship between classifications.

CITED\_BY: connects a Paper node to one or more Citation nodes indicating which papers cite the current paper.

CITATION\_OF: connects a Citation node to a Paper node indicating which paper the citation refers to.

Cypher(it is Neo4j’s graph query language that allows users to store and retrieve data from the graph database) query to retrieve all papers with their authors, classifications, and citations:

MATCH (p:Paper)-[:WRITTEN\_BY]->(a:Author), (p)-[:HAS\_CLASSIFICATION]->(c:Classification), (p)<-[:CITATION\_OF]-(ci:Citation)

RETURN p.title, COLLECT(DISTINCT a.name) as authors, COLLECT(DISTINCT c.name) as classifications, COLLECT(DISTINCT ci.paper\_id) as citations

**2. Download the raw data from Cora Research Paper Classification Project : http://people.cs.umass.edu/~mccallum/data.html The database contains approximately 25,000 authors, 37,000 papers and 220,000 relationships.**

**3. Load this data using Neo4j Data Browser**

LOAD CSV WITH HEADERS FROM

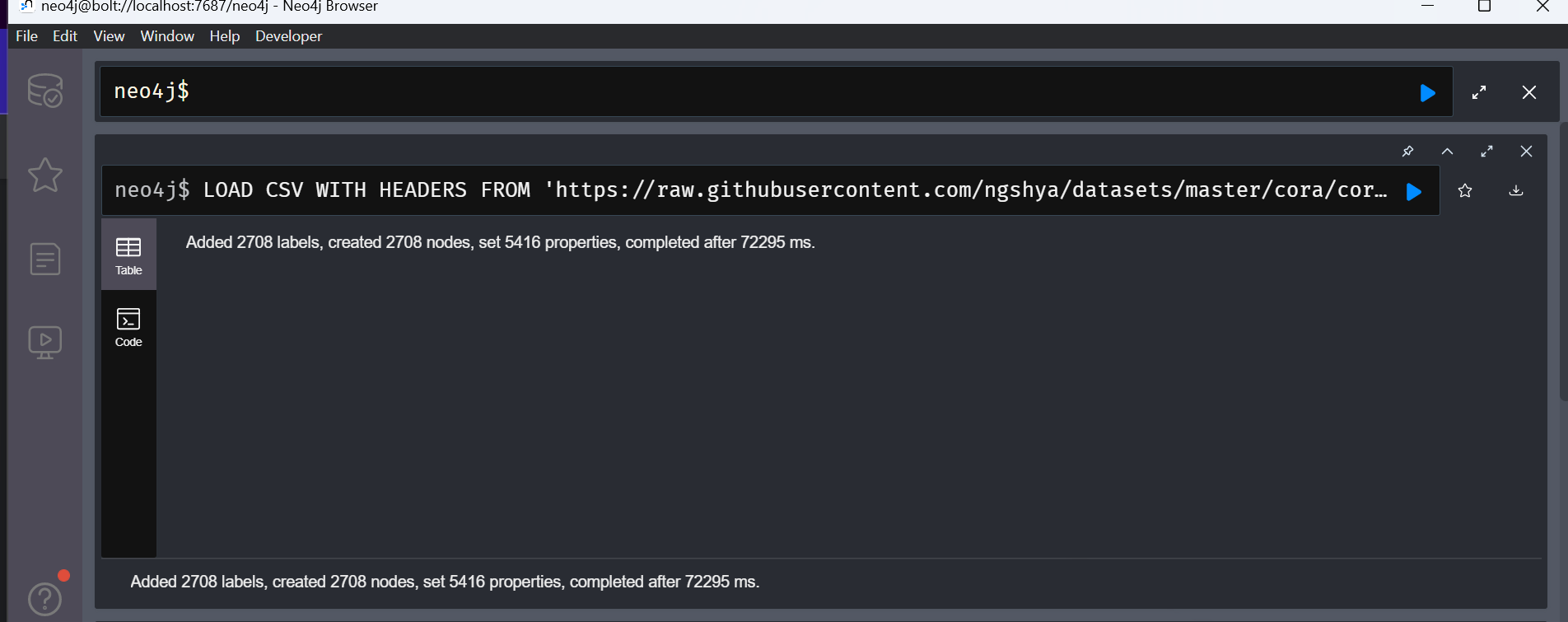
'https://raw.githubusercontent.com/ngshya/datasets/master/cora/cora\_content.csv

'

AS line FIELDTERMINATOR ','

CREATE (:Paper {id: line.paper\_id, class: line.label})

The above Cypher query loads data from a CSV file hosted on the specified URL and creates a Paper node for each row in the CSV file.

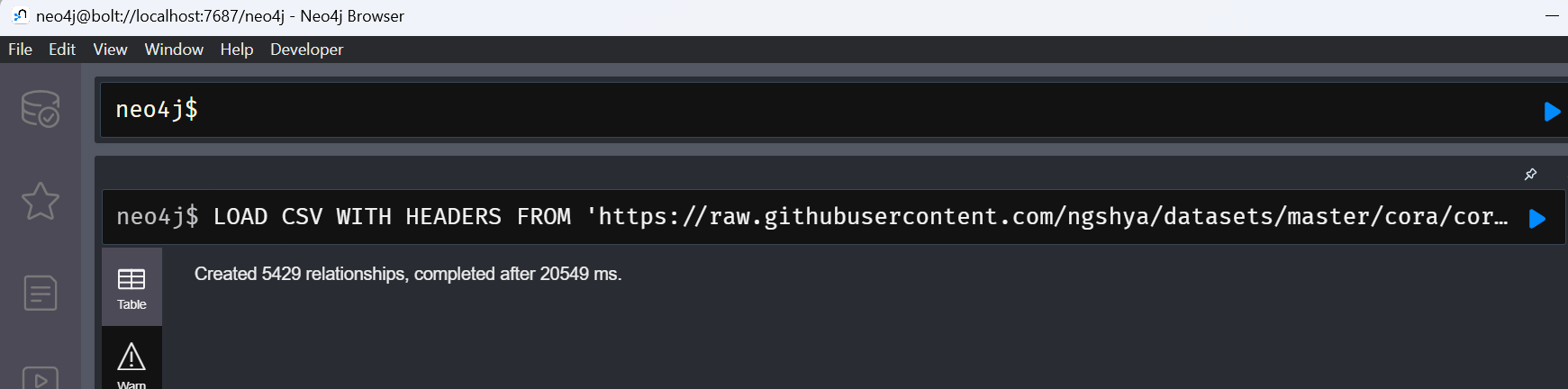


LOAD CSV WITH HEADERS FROM

'https://raw.githubusercontent.com/ngshya/datasets/master/cora/cora\_cites.csv' AS line FIELDTERMINATOR ',' MATCH (citing\_paper:Paper {id:

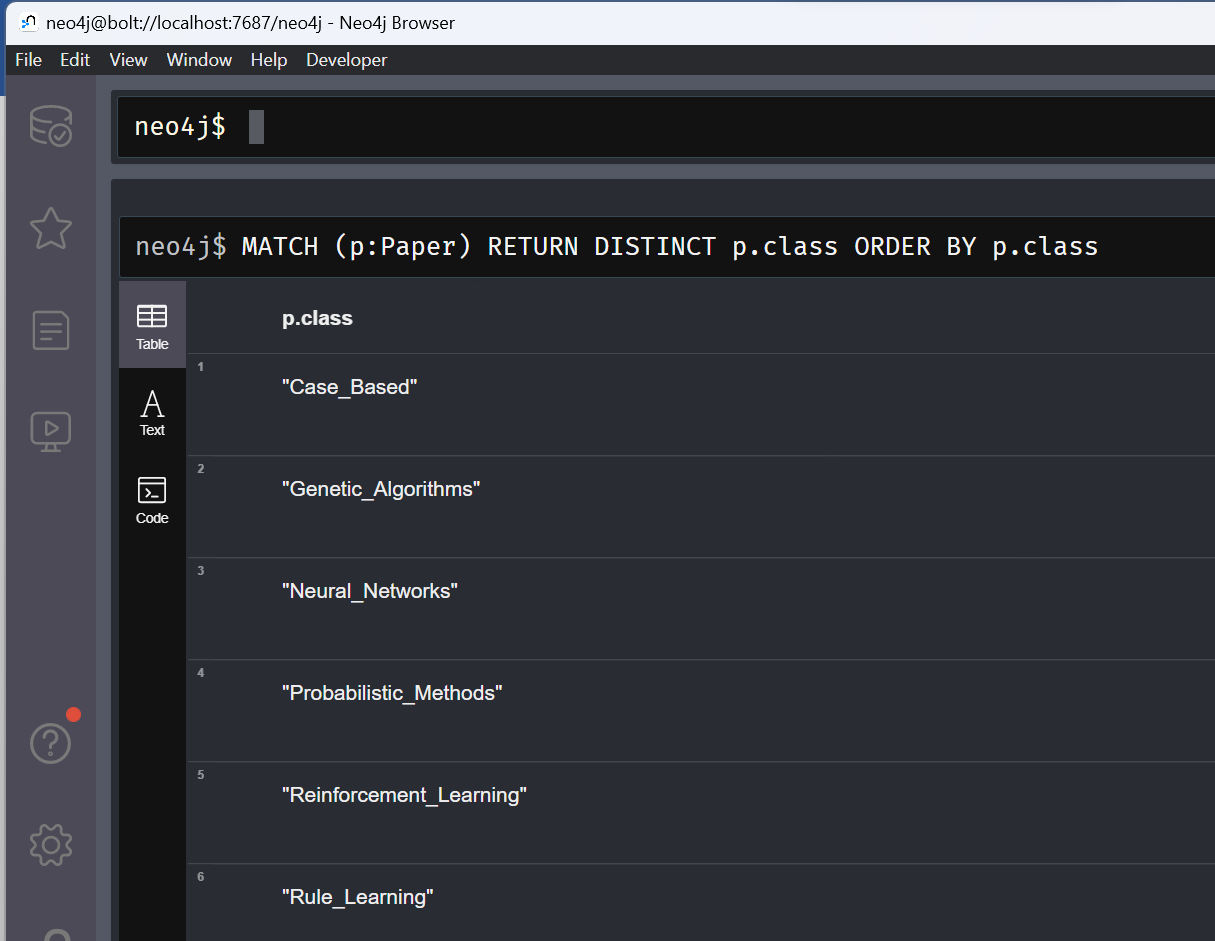
line.citing\_paper\_id}),(cited\_paper:Paper {id: line.cited\_paper\_id}) CREATE (citing\_paper)-[:CITES]->(cited\_paper)

The above Cypher query loads data from a CSV file hosted on the specified URL and creates a CITES relationship between Paper nodes based on the data in the CSV file.

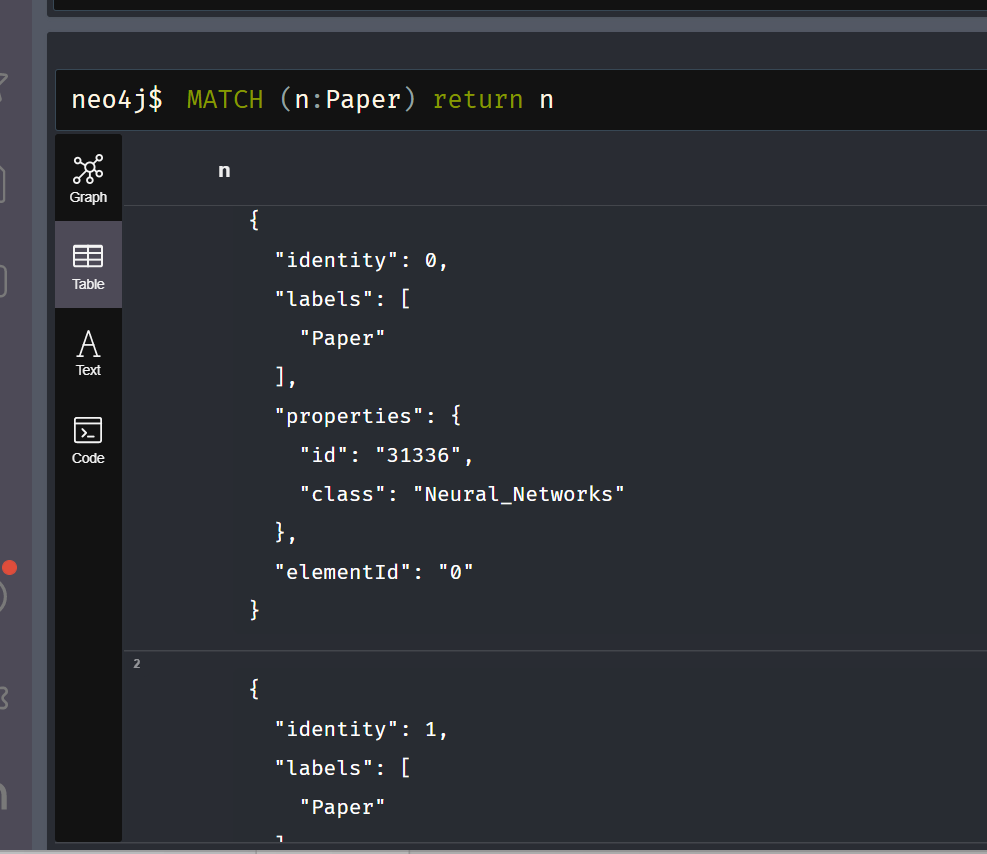


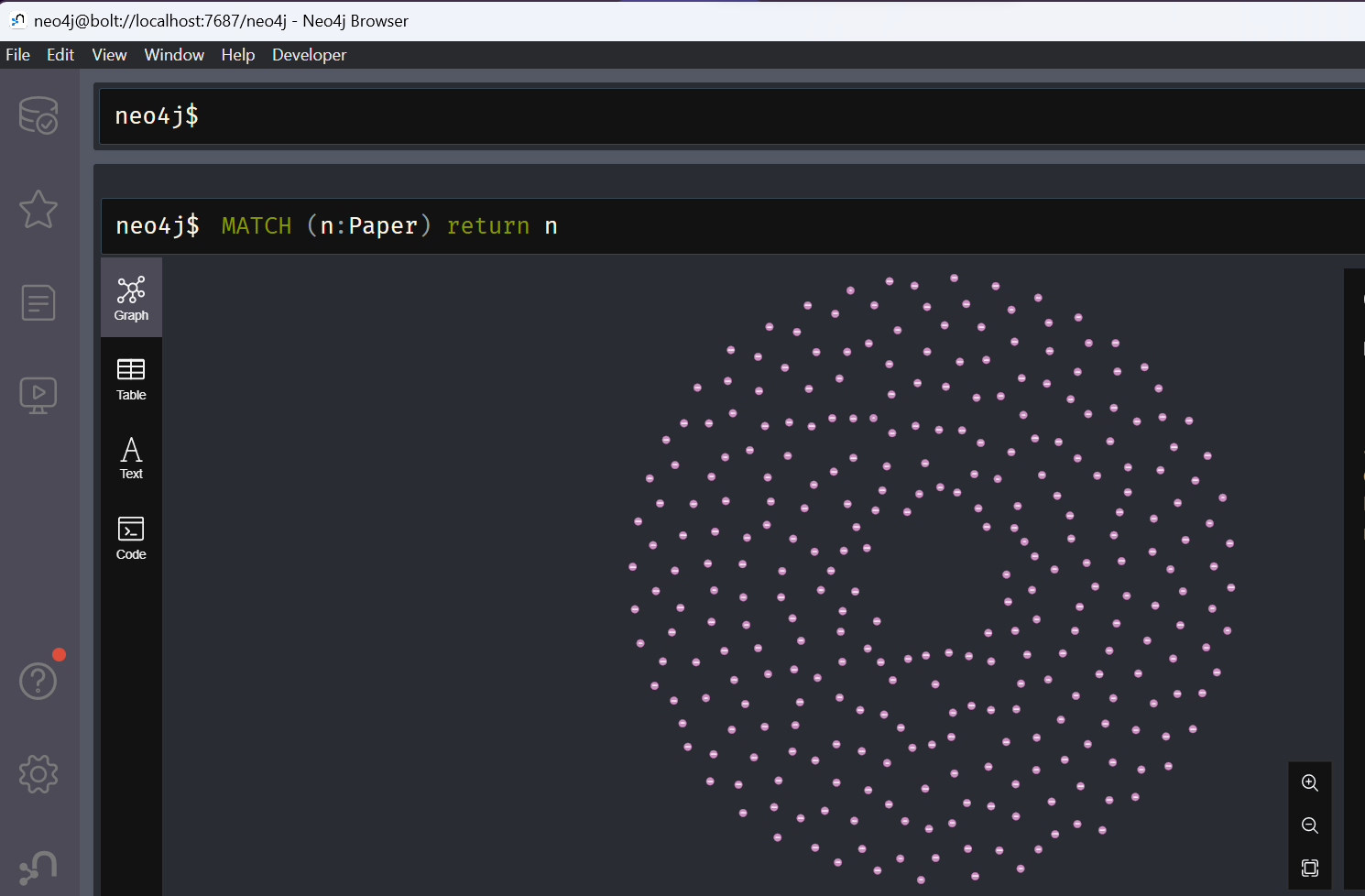
MATCH (p:Paper) RETURN DISTINCT p.class ORDER BY p.class

The above Cypher query retrieves all distinct classes of Paper nodes in the database and sorts them in ascending order.



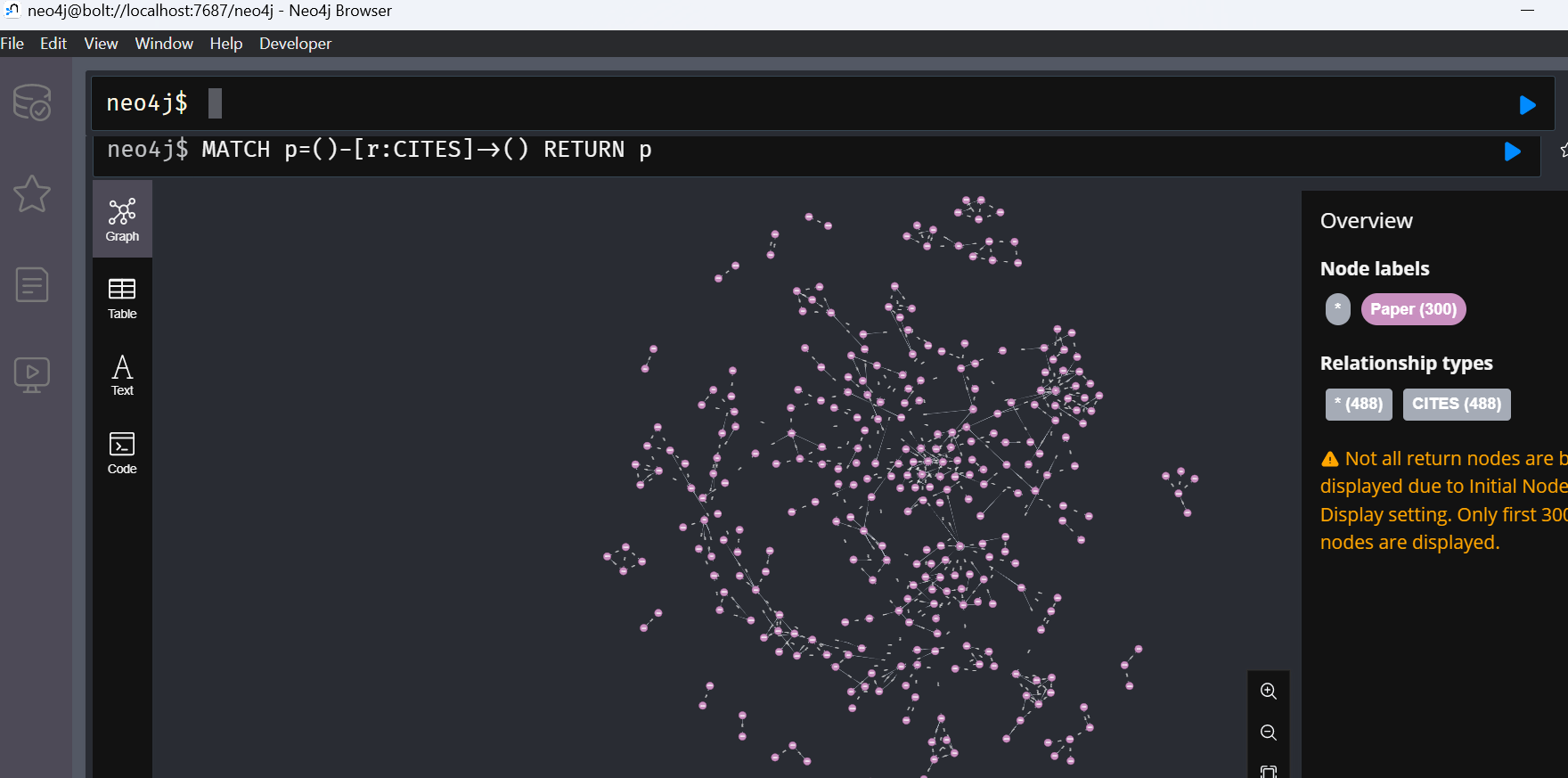
MATCH (n:Paper) return n





The above Cypher query retrieves all Paper nodes in the database and returns them.

MATCH p=()-[r:CITES]->() RETURN p



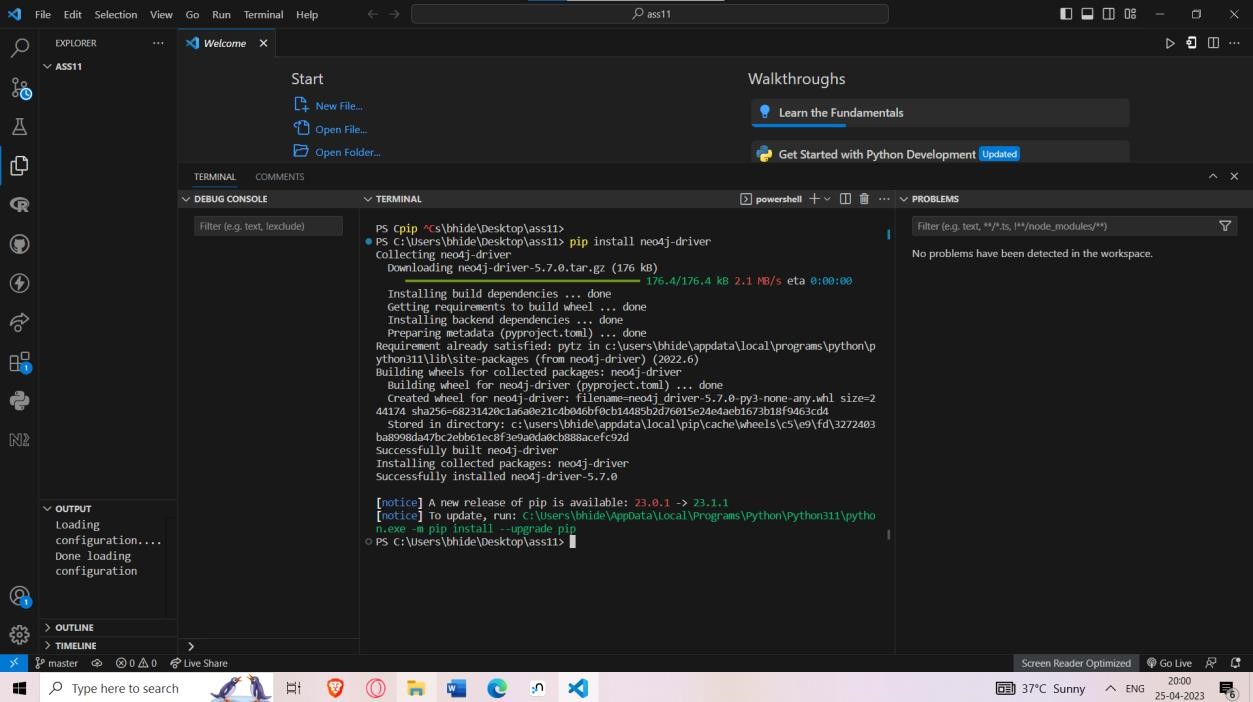
The above Cypher query retrieves all CITES relationships in the database and returns them as paths (p)

**4. Design the python based desktop application for any kind of search on above database. The application should able to answer queries like**

**a) Does paper A cite paper B? If not directly, does paper A cite a paper which in its turn cites paper B? And so on, in several levels.**

**b) Show the full classification of a paper (for example, Databases / Relational)**

Installed neo4j-driver to run python application



Python Desktop Application:

Code :

import sys import os import tkinter as tk from tkinter import \*

import tkinter.messagebox

# For Neo4j Connection from neo4j import GraphDatabase class Neo4jConnection:

def \_\_init\_\_(self, uri, user, pwd):

self.\_\_uri = uri self.\_\_user = user self.\_\_pwd = pwd self.\_\_driver = None try:

self.\_\_driver = GraphDatabase.driver(self.\_\_uri, auth=(self.\_\_user, self.\_\_pwd)) except Exception as e:

print("Failed to create the driver:", e)

def close(self): if self.\_\_driver is not None:

self.\_\_driver.close()

def query(self, query, db=None):

assert self.\_\_driver is not None, "Driver not initialized!" session = None response = None try:

session = self.\_\_driver.session(database=db) if db is not None else self.\_\_driver.session()

response = list(session.run(query)) except Exception as e:

print("Query failed:", e) finally: if session is not None:

session.close() return response conn = Neo4jConnection(uri="bolt://localhost:7687", user="neo4j", pwd="newpass123")

# ^ Neo4j Connected

window = tk.Tk() window.title("Neo4j Desktop App") window.geometry("700x500") window.configure(bg="grey") blog=tk.StringVar() blog\_title=tk.StringVar() direct\_id1=tk.StringVar() direct\_id2=tk.StringVar() recur\_id1=tk.StringVar() recur\_id2=tk.StringVar()

#submitting query def submit(): query\_string = blog\_title.get()

result = conn.query(query\_string, db='neo4j') result\_label.config(text=result) # Update the label text with the query result blog.set("") # Clear the blog\_title entry widget

def direct\_check(): id1=direct\_id1.get() id2=direct\_id2.get()

query\_string = '''MATCH p=(:Paper{id:"'''+id1+'''"})-[r:CITES]->(:Paper{id:"'''+id2+'''"}) RETURN p'''

result = conn.query(query\_string, db='neo4j') if(result):

Label(window,text="YES", fg="blue",font=("Arial", 15),width=37).grid(row=160) else:

Label(window,text="NO", fg="RED",font=("Arial", 15),width=37).grid(row=160) blog.set("")

def indirect\_check(): id1=recur\_id1.get() id2=recur\_id2.get()

query\_string = '''MATCH p=(:Paper{id:"'''+id1+'''"})-[r:CITES]->() MATCH q=(:Paper{id:"'''+id2+'''"}) RETURN q''' result = conn.query(query\_string, db='neo4j') if(result):

Label(window,text="YES", fg="blue",font=("Arial", 15),width=37).grid(row=220) else:

Label(window,text="NO", fg="RED",font=("Arial", 15),width=37).grid(row=220) blog.set("")

#tkinter window

title\_label = tk.Label(window,text="Neo4j Python Desktop Application", fg="black",font=("Arial", 25, 'bold'),width=37)

title\_label.grid(row=0,column=0, pady=10)

name\_label = tk.Label(window, text='Query', font=('calibre',10, 'bold')) name\_label.grid(row=70, pady=10)

name\_entry = tk.Entry(window, textvariable=blog\_title, font=('calibre',10,'normal'), width=70)

name\_entry.grid(row=80, pady=5)

sub\_btn = tk.Button(window, text='Run Query', command=submit) sub\_btn.grid(row=110, pady=10)

result\_label = tk.Label(window, text='', font=('calibre', 12, 'normal')) result\_label.grid(row=90, pady=20)

name\_label = tk.Label(window, text='Does Paper with id1 cite id2 directly?', font=('calibre',10,'bold')).grid(row=120)

name\_entry1 = tk.Entry(window, textvariable=direct\_id1, font=('calibre',10,'normal')) name\_entry1.grid(row=130, pady=5)

name\_entry2 = tk.Entry(window, textvariable=direct\_id2, font=('calibre',10,'normal')) name\_entry2.grid(row=140, pady=5) sub\_btn = tk.Button(window, text='Check', command=direct\_check).grid(row=150, pady=10)

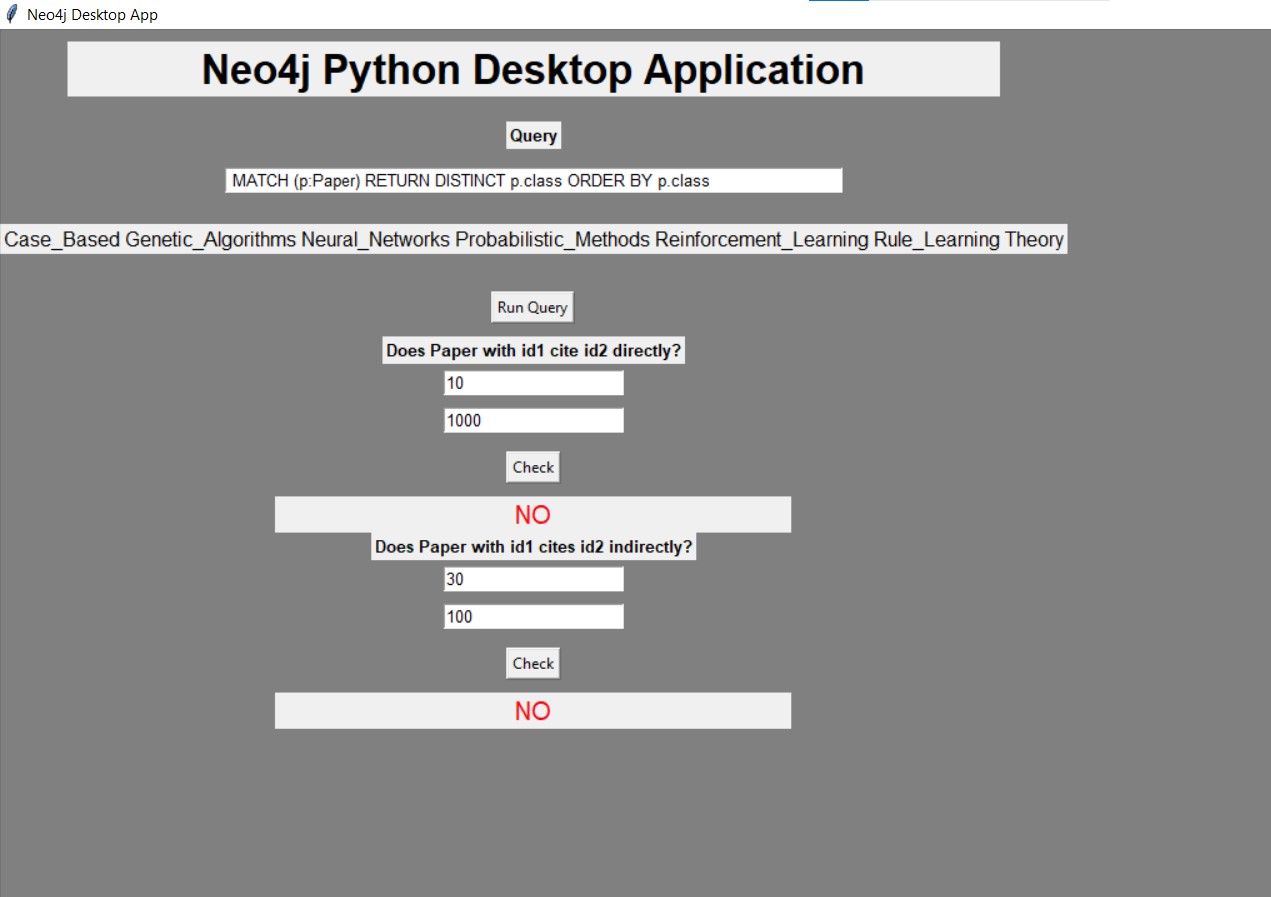
name\_label = tk.Label(window, text='Does Paper with id1 cites id2 indirectly?', font=('calibre',10,'bold')).grid(row=180)

name\_entry1 = tk.Entry(window, textvariable=recur\_id1, font=('calibre',10,'normal')) name\_entry1.grid(row=190, pady=5)

name\_entry2 = tk.Entry(window, textvariable=recur\_id2, font=('calibre',10,'normal')) name\_entry2.grid(row=200, pady=5)

sub\_btn = tk.Button(window, text='Check', command=indirect\_check).grid(row=210, pady=10)

window.mainloop()



This is a Python desktop application that provides a GUI for interacting with a Neo4j database. It allows the user to submit queries to the database and retrieve results. The application has two functions:

direct\_check(): This function checks whether a paper with a given ID (direct\_id1) directly cites another paper with a given ID (direct\_id2). If the citation exists, it displays "YES" in blue, and if not, it displays "NO" in red.

indirect\_check(): This function checks whether a paper with a given ID (recur\_id1) cites another paper with a given ID (recur\_id2) indirectly. If the citation exists, it displays "YES" in blue, and if not, it displays "NO" in red.

The submit() function is called when the user clicks the "Run Query" button, and it executes the query entered by the user (blog\_title) and displays the result in a label (result\_label).

The application uses the tkinter library to create the GUI and the neo4j library to connect to the Neo4j database.