2020BTECS00040

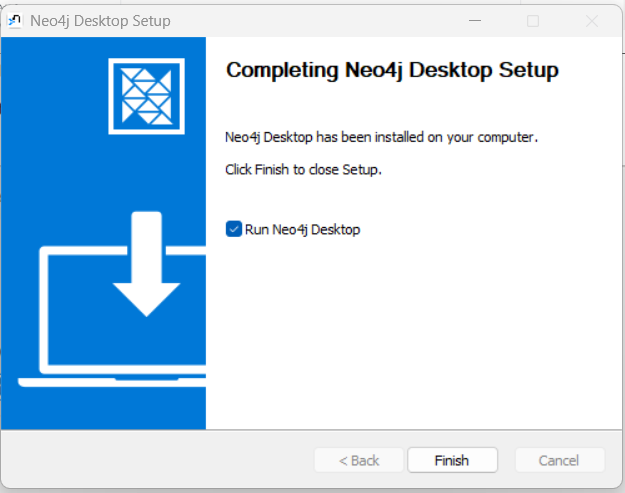
Pranav Dilip Hingankar

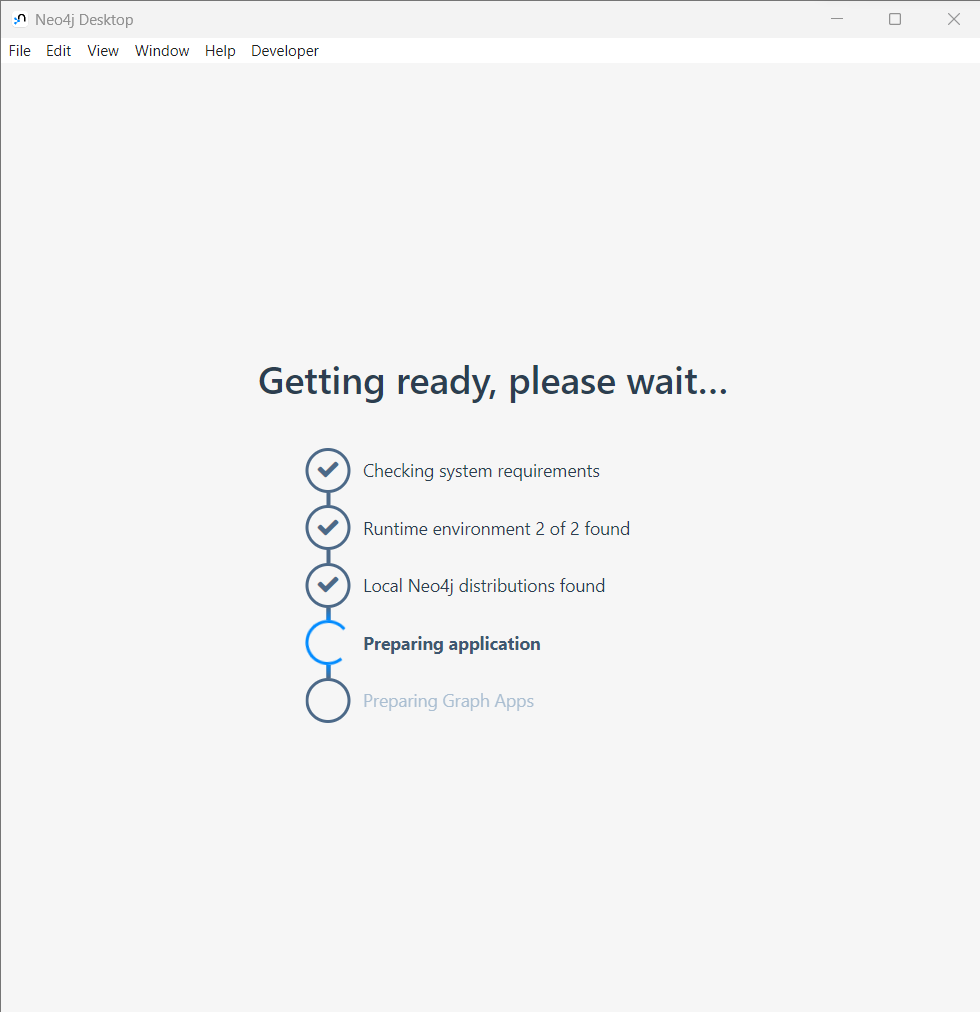
Batch: T6

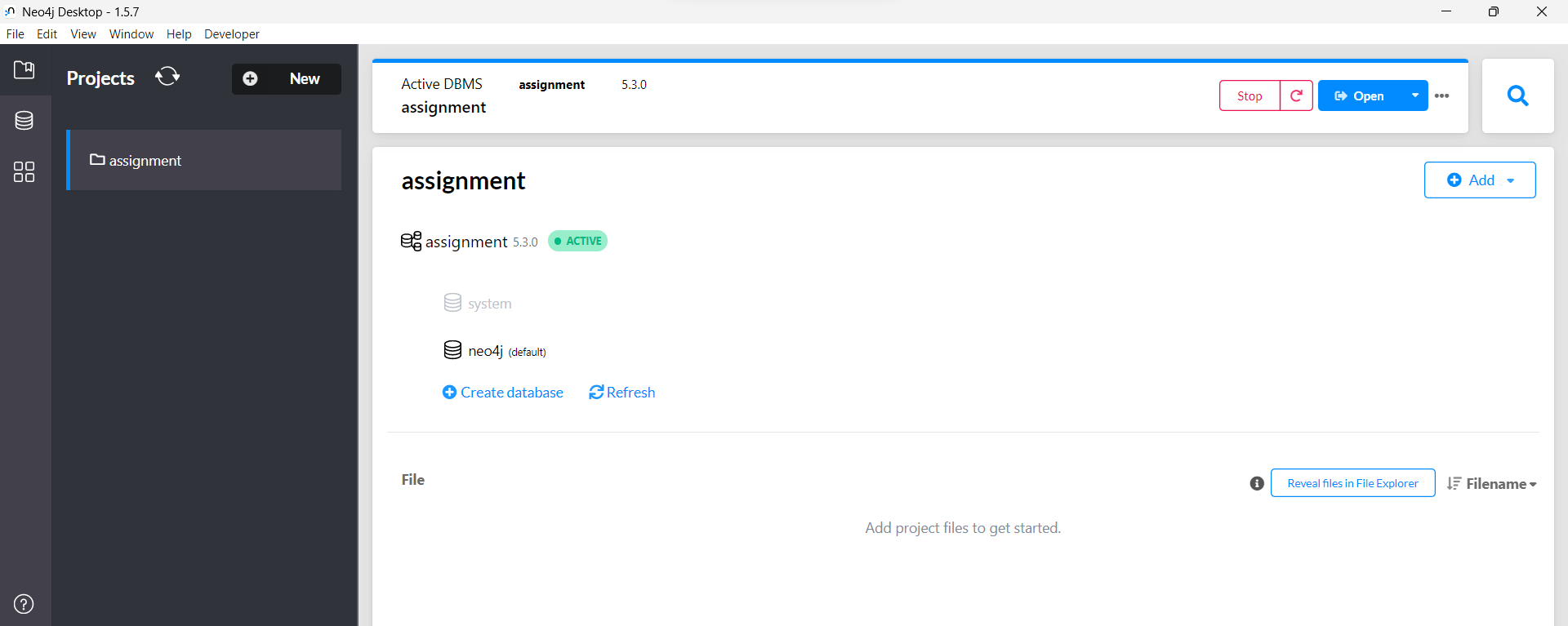
**Assignment 11**

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 subclassifications “Relational” and “Object-Oriented”). A paper usually has a list of references, which are other papers. These are called citations.

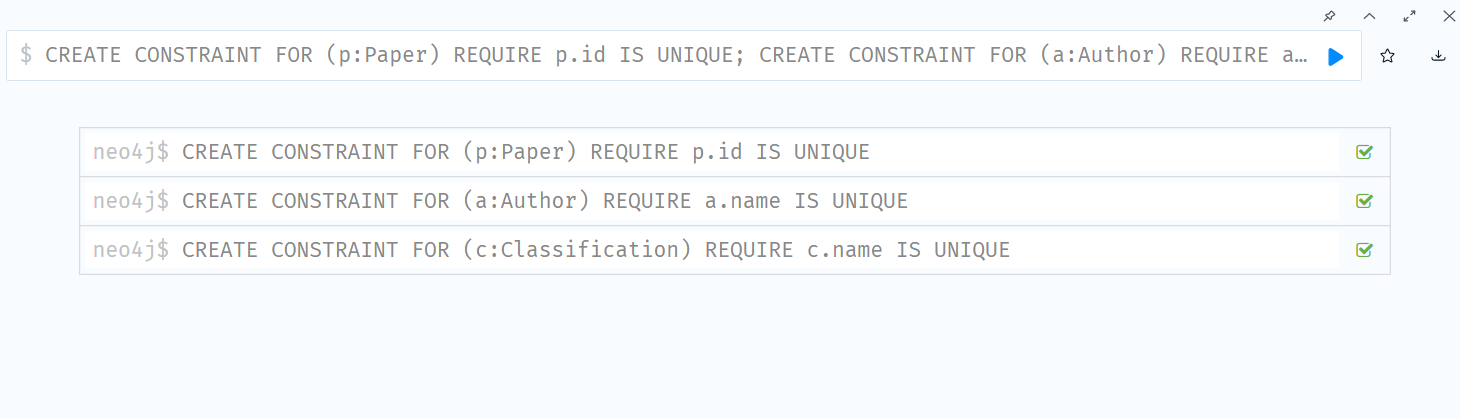


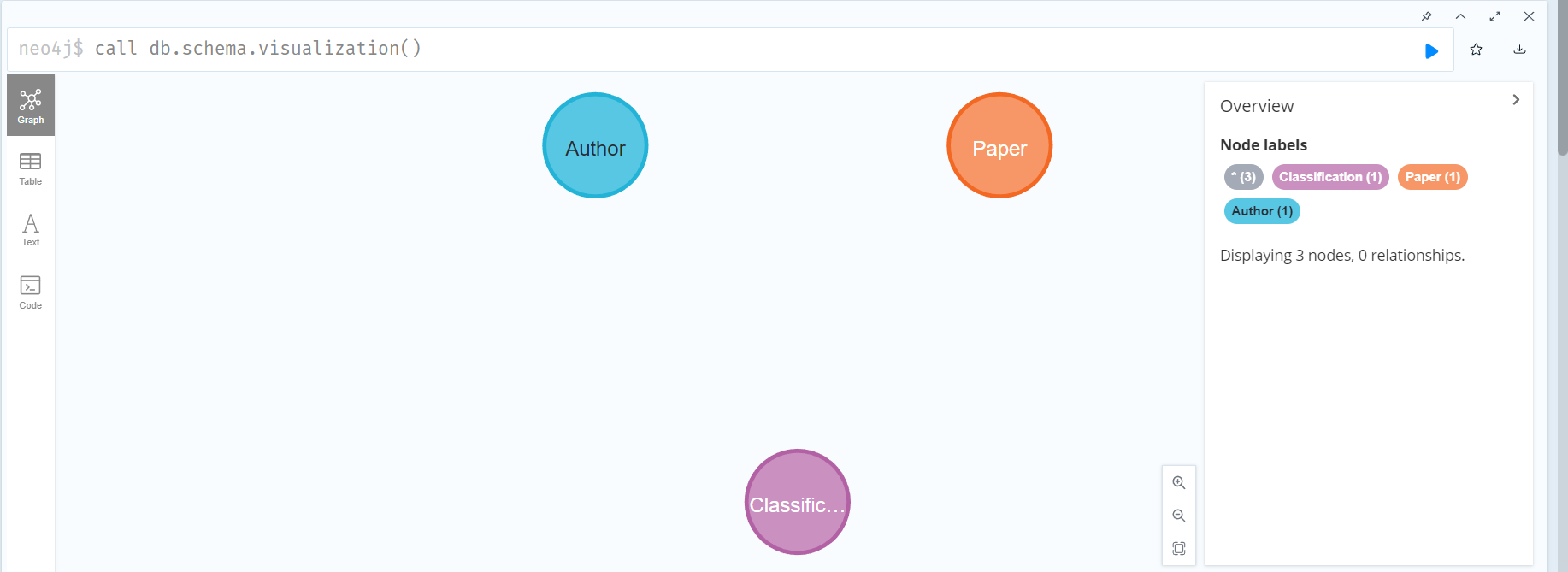




1. Design/model the graph database using Neo4j for above scenario.

1. 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.





1. Load this data using Neo4j Data Browser

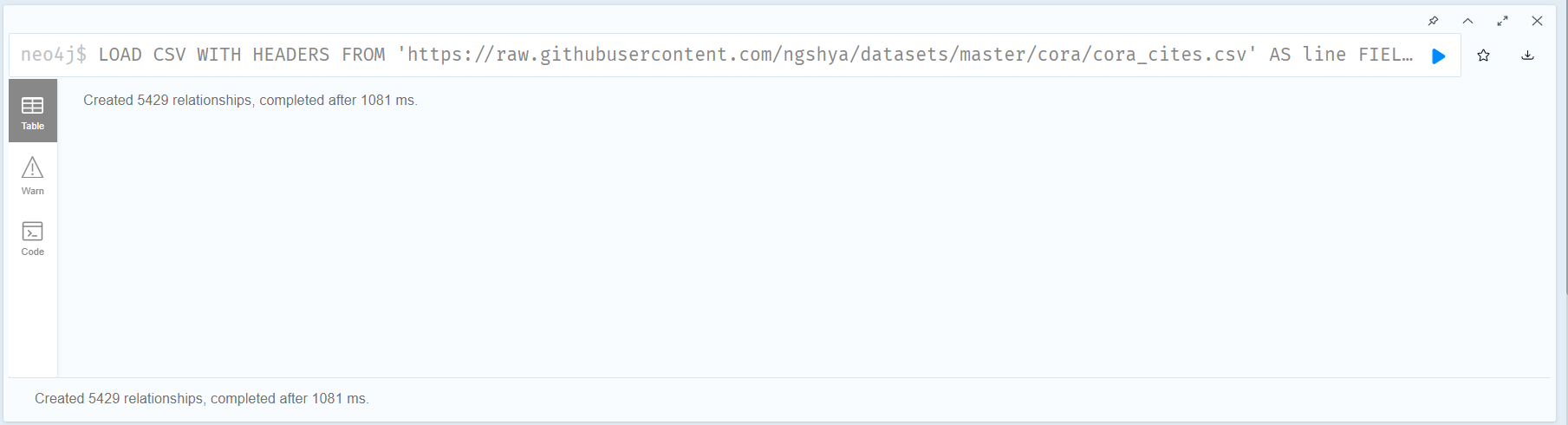


LOAD CSV WITH HEADERS FROM

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

AS line FIELDTERMINATOR ','

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



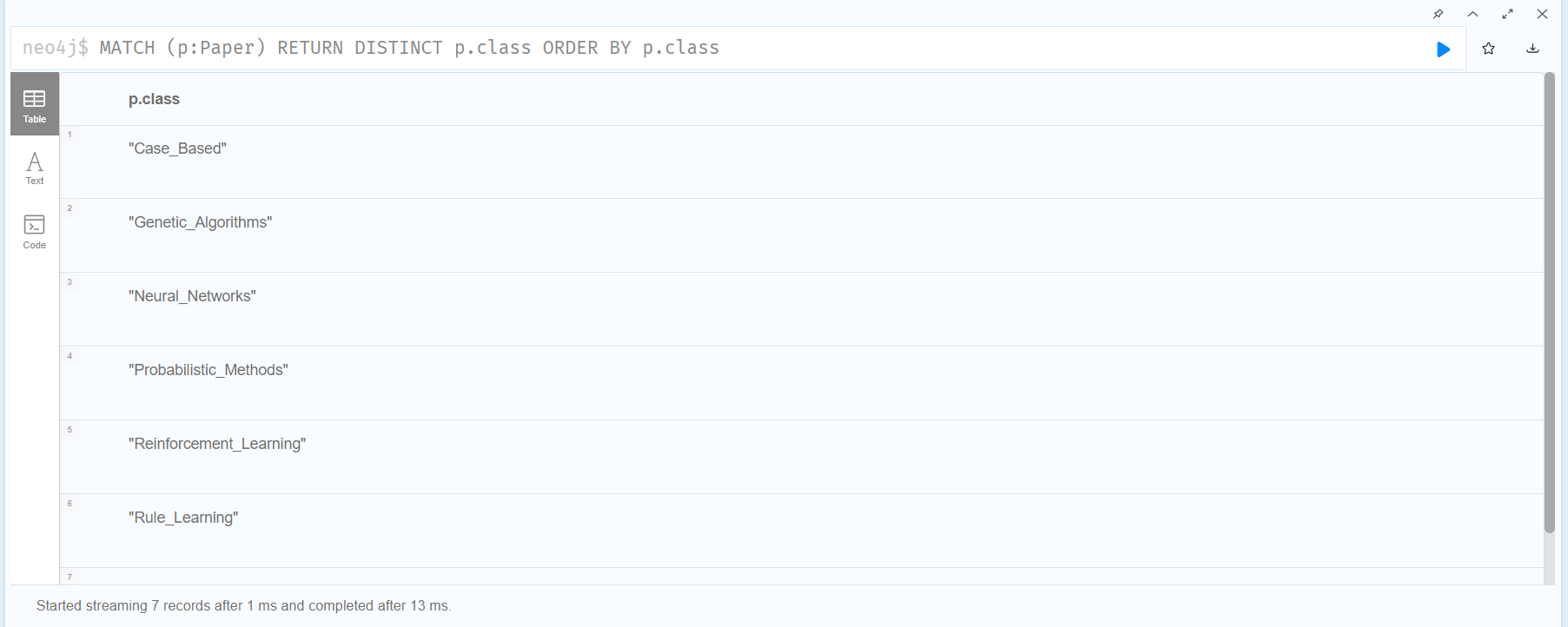
LOAD CSV WITH HEADERS FROM

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

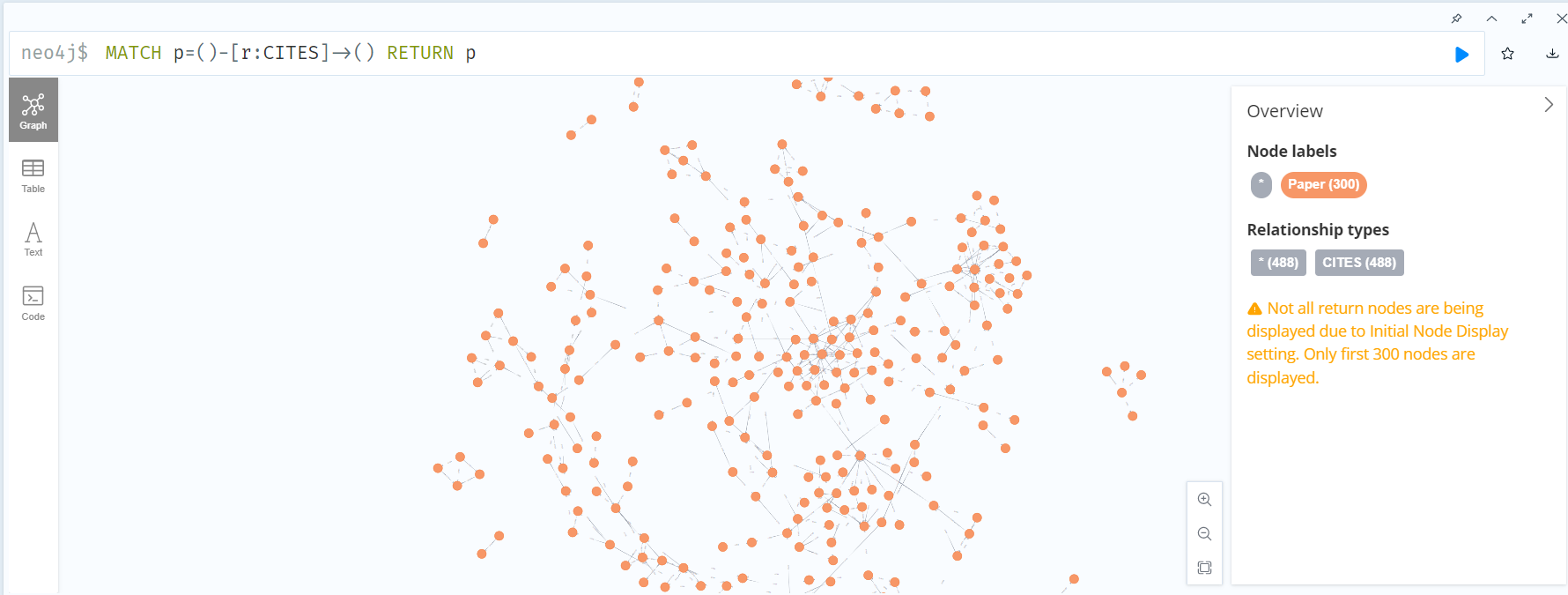
AS line FIELDTERMINATOR ','

MATCH (citing\_paper:Paper {id: line.citing\_paper\_id}),(cited\_paper:Paper {id: line.cited\_paper\_id})

CREATE (citing\_paper)-[:CITES]->(cited\_paper)

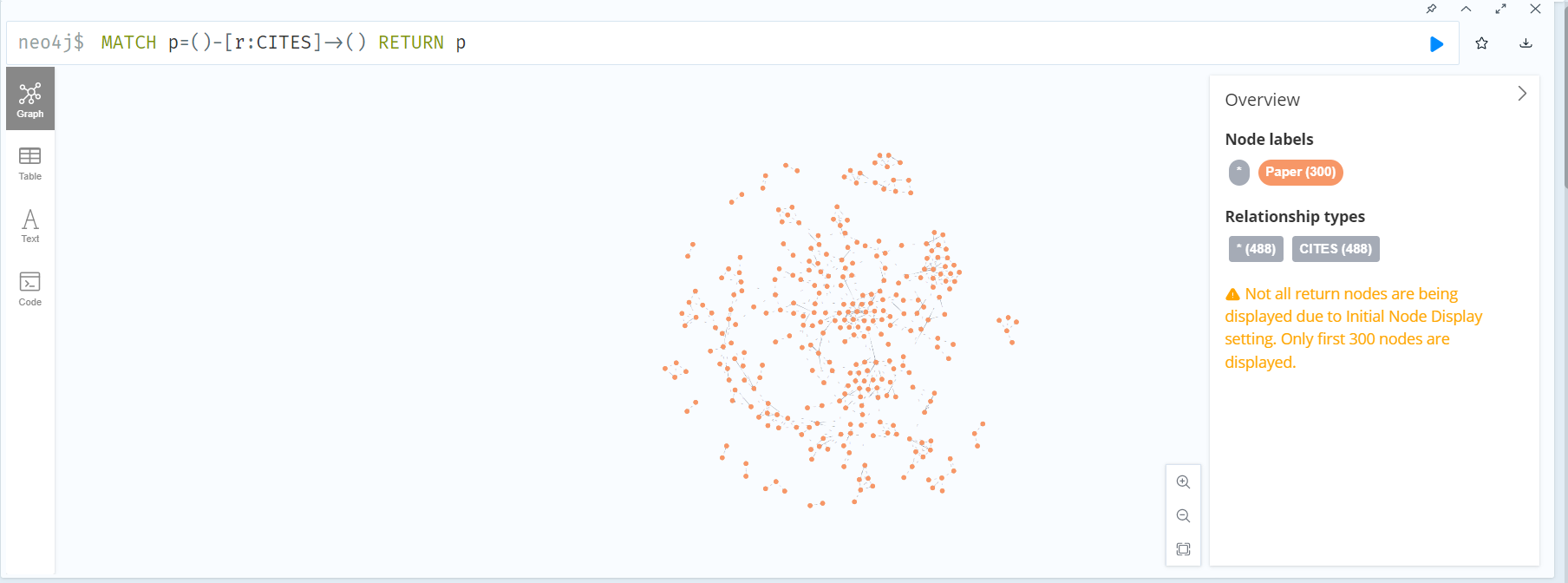


MATCH (p:Paper) RETURN DISTINCT p.class ORDER BY p.class



MATCH p=()-[r:CITES]->() RETURN p

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



PYTHON APPLICATION:

# 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="neo4j")

# # ^ Neo4j Connected

# window = tk.Tk()

# window.title("Desktop App by Sagar")

# 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')

#     print(result)

#     blog.set("")

# 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

# Label(window, text="Neo4j Application", fg="black",

#       font=("Arial", 25, 'bold'), width=37).grid(row=0, column=0)

# name\_label = tk.Label(window, text='Query', font=(

#     'calibre', 10, 'bold')).grid(row=70)

# name\_entry = tk.Entry(window, textvariable=blog\_title, font=(

#     'calibre', 10, 'normal'), width=70).grid(row=80)

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

# name\_label = tk.Label(window, text='Does Paper with id1 cites id2 directly?', font=(

#     'calibre', 10, 'bold')).grid(row=120)

# name\_entry = tk.Entry(window, textvariable=direct\_id1,

#                       font=('calibre', 10, 'normal')).grid(row=130)

# name\_entry = tk.Entry(window, textvariable=direct\_id2,

#                       font=('calibre', 10, 'normal')).grid(row=140)

# sub\_btn = tk.Button(window, text='Check', command=direct\_check).grid(row=150)

# name\_label = tk.Label(window, text='Does Paper with id1 cites id2 indirectly?', font=(

#     'calibre', 10, 'bold')).grid(row=180)

# name\_entry = tk.Entry(window, textvariable=recur\_id1,

#                       font=('calibre', 10, 'normal')).grid(row=190)

# name\_entry = tk.Entry(window, textvariable=recur\_id2,

#                       font=('calibre', 10, 'normal')).grid(row=200)

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

# window.mainloop()

import tkinter as tk

from neo4j import GraphDatabase

# Neo4j database connection details

uri = "bolt://localhost:7687"

username = "neo4j"

password = "root1234"

# Initialize the Neo4j driver

driver = GraphDatabase.driver(uri, auth=(username, password))

# Function to check if paper A cites paper B

def does\_paper\_a\_cite\_paper\_b(tx, paper\_a\_id, paper\_b\_id):

    query = (

        "MATCH (a:Paper {paper\_id: $paper\_a\_id})-[:CITATION\*..3]->(b:Paper {paper\_id: $paper\_b\_id}) "

        "RETURN count(\*) > 0"

    )

    result = tx.run(query, paper\_a\_id=paper\_a\_id, paper\_b\_id=paper\_b\_id)

    return result.single()[0]

# Function to get the full classification of a paper

def get\_classification\_of\_paper(tx, paper\_id):

    query = (

        "MATCH (p:Paper {paper\_id: $paper\_id})-[:CLASSIFICATION\*]->(c:Classification) "

        "RETURN c.name"

    )

    result = tx.run(query, paper\_id=paper\_id)

    return [record['c.name'] for record in result]

# Function to handle search button click

def search():

    # Get the paper IDs from the entry widgets

    paper\_a\_id = entry\_paper\_a\_id.get()

    paper\_b\_id = entry\_paper\_b\_id.get()

    paper\_id = entry\_paper\_id.get()

    # Open a new Neo4j session

    with driver.session() as session:

        # Check if paper A cites paper B

        result\_a\_b = does\_paper\_a\_cite\_paper\_b(session, paper\_a\_id, paper\_b\_id)

        label\_a\_b.config(text="Yes" if result\_a\_b else "No")

        # Check if paper A cites a paper that cites paper B

        result\_a\_cite\_b = False

        for i in range(3):

            result\_a\_cite\_b = does\_paper\_a\_cite\_paper\_b(

                session, paper\_a\_id, paper\_b\_id)

            if result\_a\_cite\_b:

                break

        label\_a\_cite\_b.config(text="Yes" if result\_a\_cite\_b else "No")

        # Get the full classification of the paper

        result\_classification = get\_classification\_of\_paper(session, paper\_id)

        label\_classification.config(text="/".join(result\_classification))

# Create the main window

window = tk.Tk()

window.title("Research Papers Database")

# Create the widgets

label\_paper\_a\_id = tk.Label(window, text="Paper A ID:")

entry\_paper\_a\_id = tk.Entry(window)

label\_paper\_b\_id = tk.Label(window, text="Paper B ID:")

entry\_paper\_b\_id = tk.Entry(window)

button\_search\_citations = tk.Button(

    window, text="Search Citations", command=search)

label\_a\_b = tk.Label(window, text="")

label\_a\_cite\_b = tk.Label(window, text="")

label\_paper\_id = tk.Label(window, text="Paper ID:")

entry\_paper\_id = tk.Entry(window)

button\_search\_classification = tk.Button(

    window, text="Search Classification", command=search)

label\_classification = tk.Label(window, text="")

# Pack the widgets

label\_paper\_a\_id.pack()

entry\_paper\_a\_id.pack()

label\_paper\_b\_id.pack()

entry\_paper\_b\_id.pack()

button\_search\_citations.pack()

label\_a\_b.pack()

label\_a\_cite\_b.pack()

label\_paper\_id.pack()

entry\_paper\_id.pack()

button\_search\_classification.pack()

label\_classification.pack()

# Run the main loop

window.mainloop()

