**5CS372: Advanced Database System Lab.**

**Assignment No. 11**

Name: Divya Milind Kekade

PRN: 2020BTECS00038

----------------------------------------------------------------------------------------------------

**Neo4j Graph Database**

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.

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.

**Installation of Neo4j:**

**Download the zip file of Neo4j Community edition from official website.**

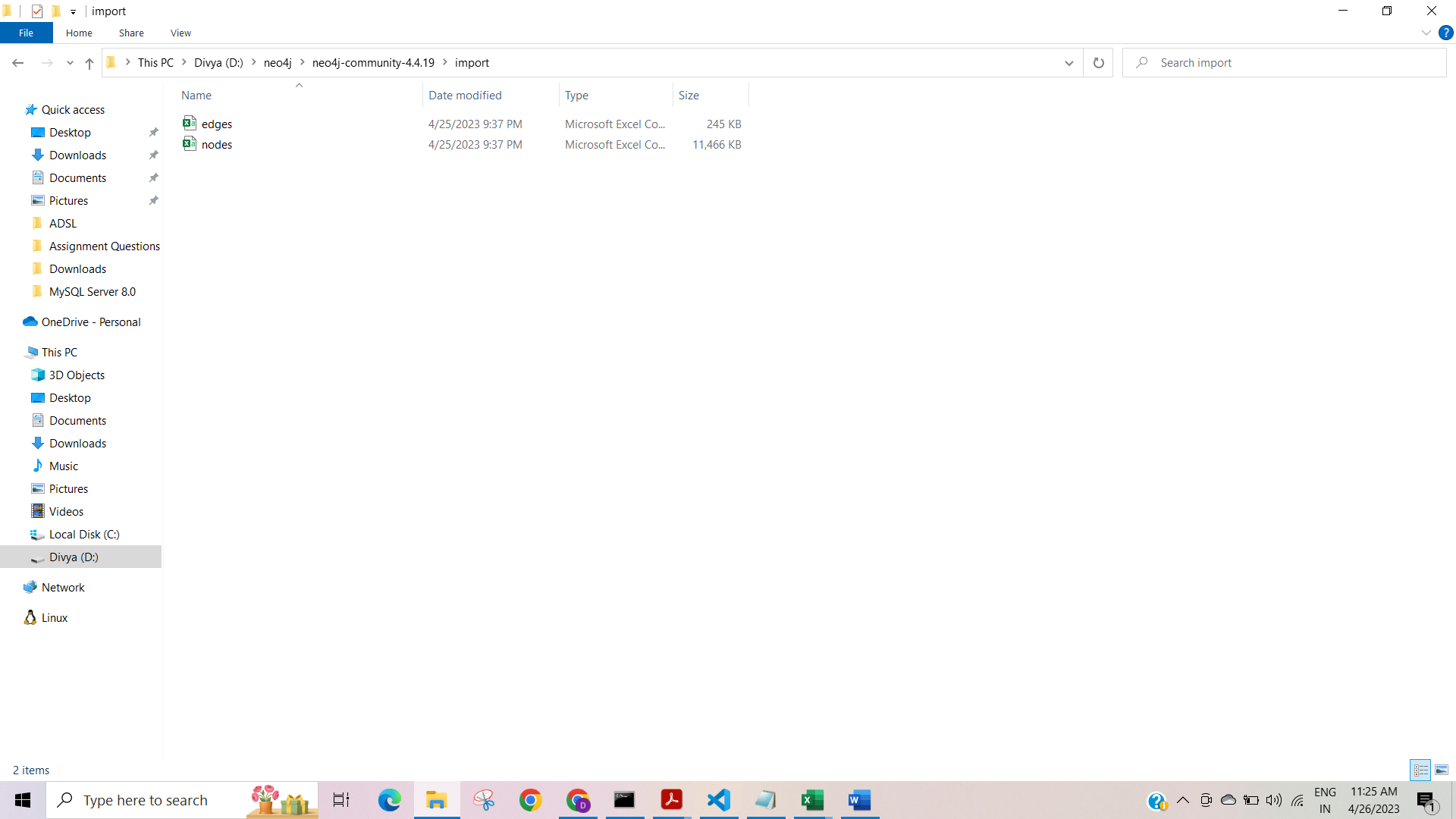
**Extract the zip file and add the path of the bin folder in environment variables of the system.**

**Now, we can start the neo4j database using the command “neo4j console”.**

**We can see the interface of neo4j browser at** [**http://localhost:7474/**](http://localhost:7474/) **.**



**Now dataset is downloaded from the Cora Research Paper Classification Project. The csv files that are to be imported to the neo4j database are placed in the import folder of neo4j.**



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

**To import the data in neo4j:**

LOAD CSV WITH HEADERS FROM 'file:///nodes.csv' AS row

Create (n:Paper{id:row["nodeId"], subject:row["subject"], features:row["features"]

});

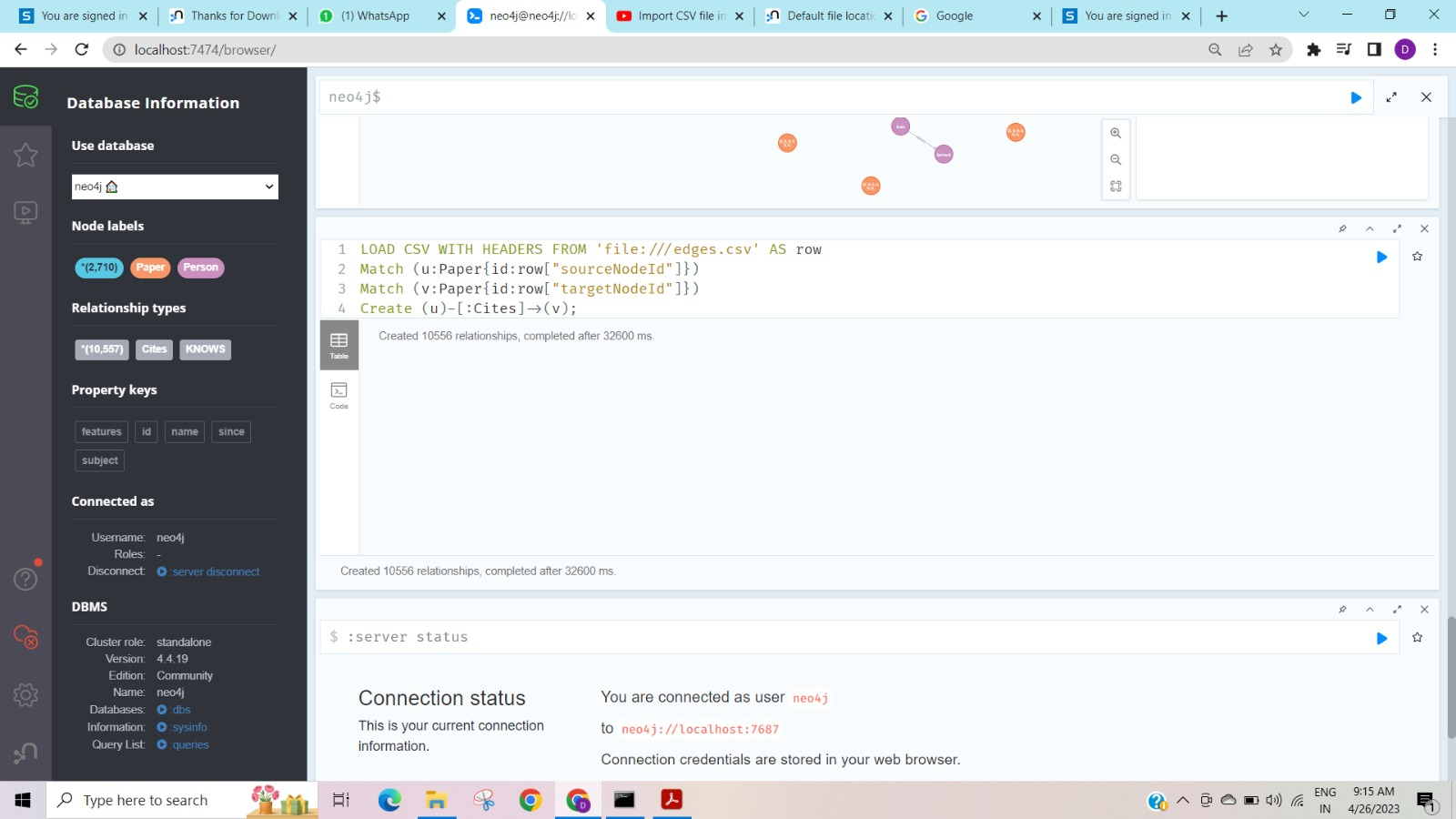


LOAD CSV WITH HEADERS FROM 'file:///edges.csv' AS row

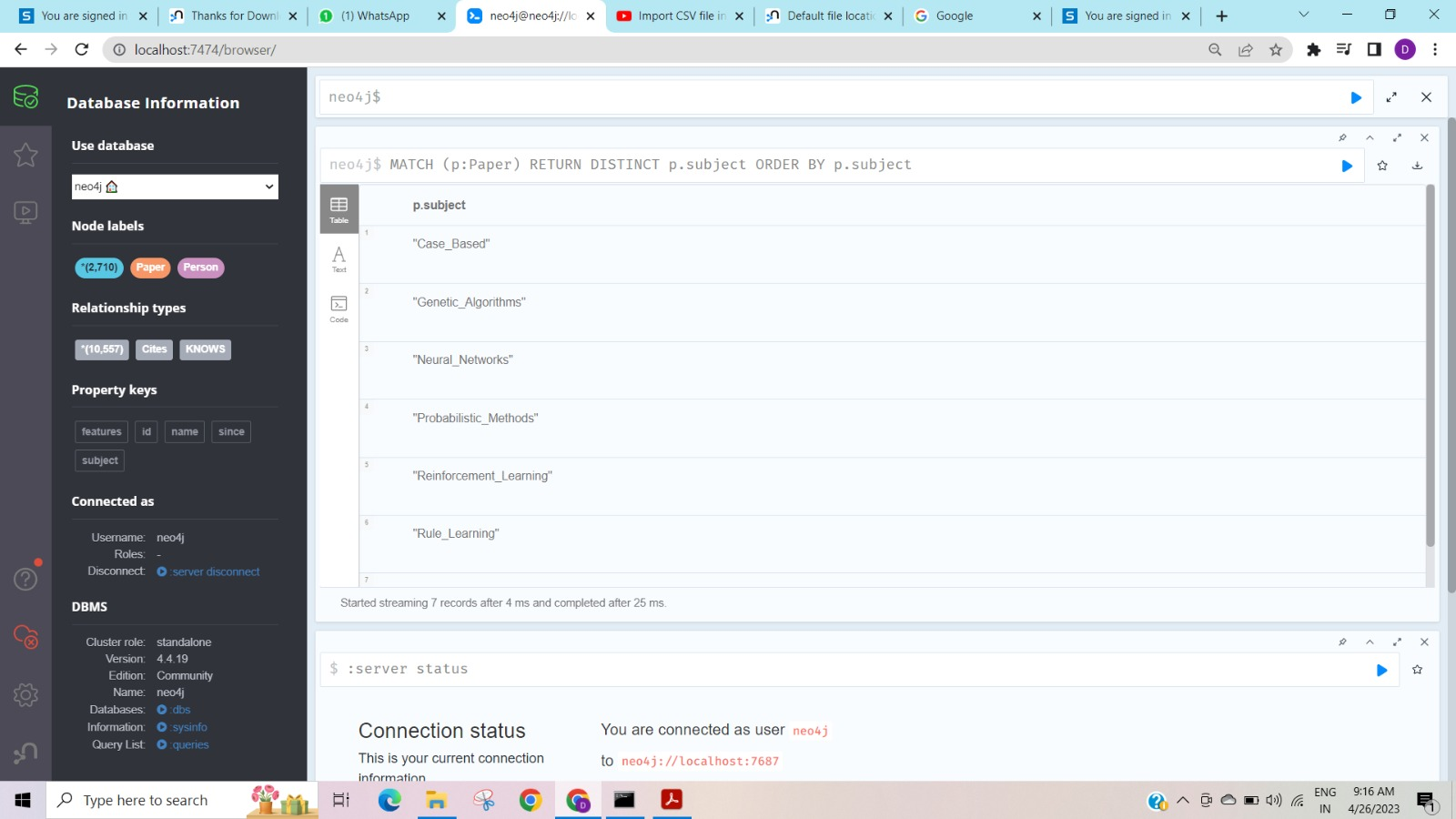
Match (u:Paper{id:row["sourceNodeId"]})

Match (v:Paper{id:row["targetNodeId"]})

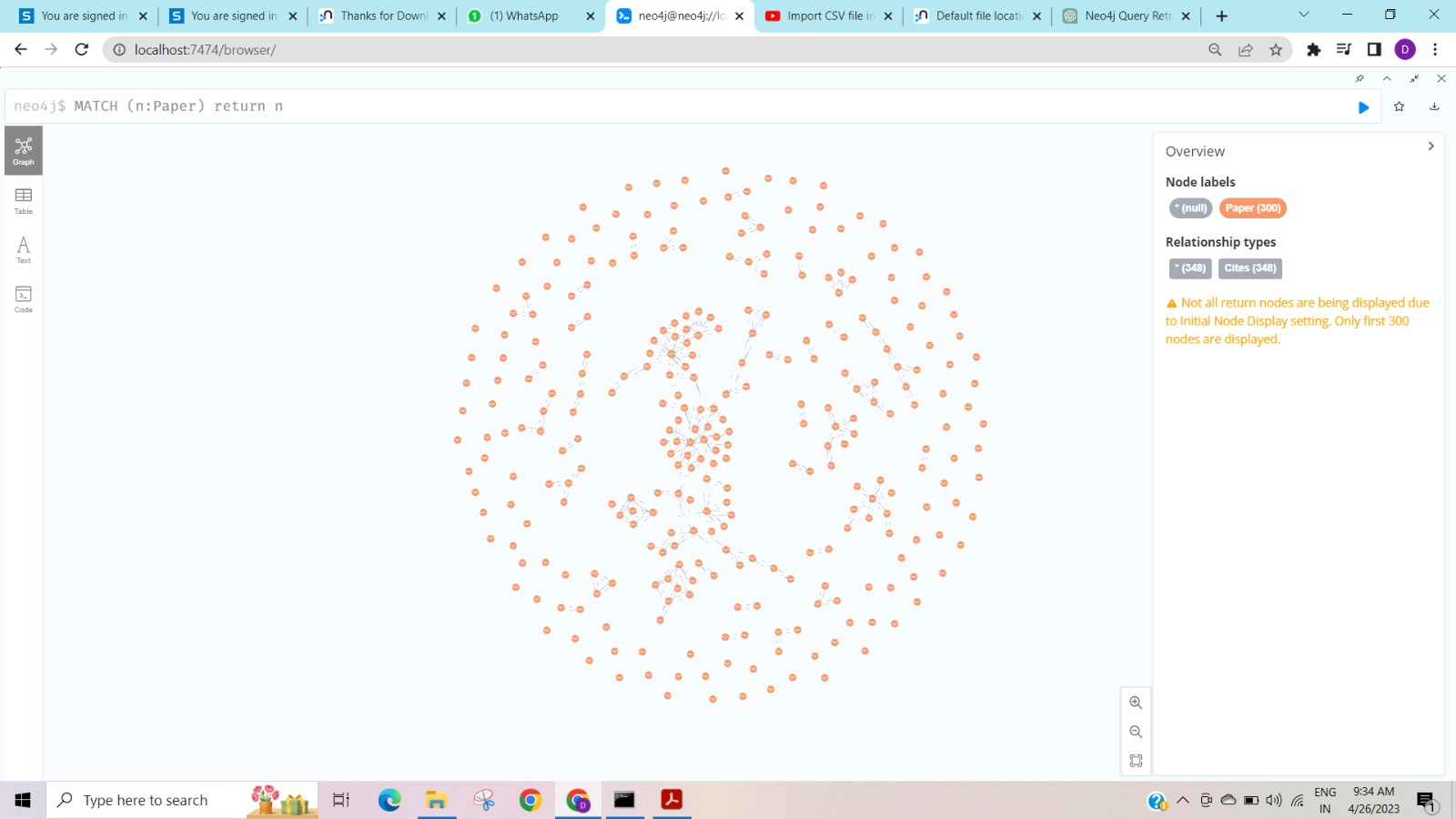
Create (u)-[:Cites]->(v);



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



MATCH (n:Paper) return n



MATCH (n:Paper)-[r:Cites]->(m:Paper)

RETURN n.id as sourceNodeId, m.id as targetNodeId, n.subject, n.features



**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)**

**Installing neo4j-driver to connect neo4j database with python GUI:**



Python GUI code:

import tkinter as tk

from neo4j import GraphDatabase

# Connect to the Neo4j database

uri = "neo4j://localhost:7687"

username = "neo4j"

password = "123456"

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

# Define a function to add a new paper node

def add\_paper\_node(node\_id, subject, features):

     with driver.session() as session:

      session.run("CREATE (:Paper {id: $node\_id, subject: $subject, features:$features})",

      node\_id=node\_id, subject=subject, features=features)

# Define a function to delete a paper node

def delete\_paper\_node(node\_id):

    with driver.session() as session:

     session.run("MATCH (n:Paper {id: $node\_id}) DETACH DELETE n",

                  node\_id=node\_id)

# Define a function to update a paper node

def update\_paper\_node(node\_id, subject, features):

    with driver.session() as session:

     session.run("MATCH (n:Paper {id: $node\_id}) SET n.subject = $subject,n.features = $features",

                 node\_id=node\_id, subject=subject, features=features)

# Define a function to view all paper nodes

def view\_paper\_nodes():

    with driver.session() as session:

     result = session.run("MATCH (n:Paper) RETURN n.id, n.subject, n.features")

     return [dict(record) for record in result]

# Define the GUI window and its components

root = tk.Tk()

root.title("Paper Database")

# Define the labels and text fields for the input form

node\_id\_label = tk.Label(root, text="Node ID:")

node\_id\_entry = tk.Entry(root)

subject\_label = tk.Label(root, text="Subject:")

subject\_entry = tk.Entry(root)

features\_label = tk.Label(root, text="Features:")

features\_entry = tk.Entry(root)

# Define the buttons for the CRUD operations

add\_button = tk.Button(root, text="Add", command=lambda:

add\_paper\_node(node\_id\_entry.get(),

 subject\_entry.get(),

features\_entry.get()))

delete\_button = tk.Button(root, text="Delete", command=lambda:

delete\_paper\_node(node\_id\_entry.get()))

update\_button = tk.Button(root, text="Update", command=lambda:

update\_paper\_node(node\_id\_entry.get(),

 subject\_entry.get(),

features\_entry.get()))

view\_button = tk.Button(root, text="View", command=lambda:

display\_results(view\_paper\_nodes()))

# Define the text area for the query results

results\_text = tk.Text(root, height=100, width=100)

# Define a function to display the query results in the text area

def display\_results(results):

     results\_text.delete(1.0, tk.END)

     for record in results:

         results\_text.insert(tk.END, f"Node ID: {record['n.id']}\nSubject:{record['n.subject']}\nFeatures: {record['n.features']}\n\n")

# Pack the components into the GUI window

node\_id\_label.pack()

node\_id\_entry.pack()

subject\_label.pack()

subject\_entry.pack()

features\_label.pack()

features\_entry.pack()

add\_button.pack(side=tk.LEFT)

delete\_button.pack(side=tk.LEFT)

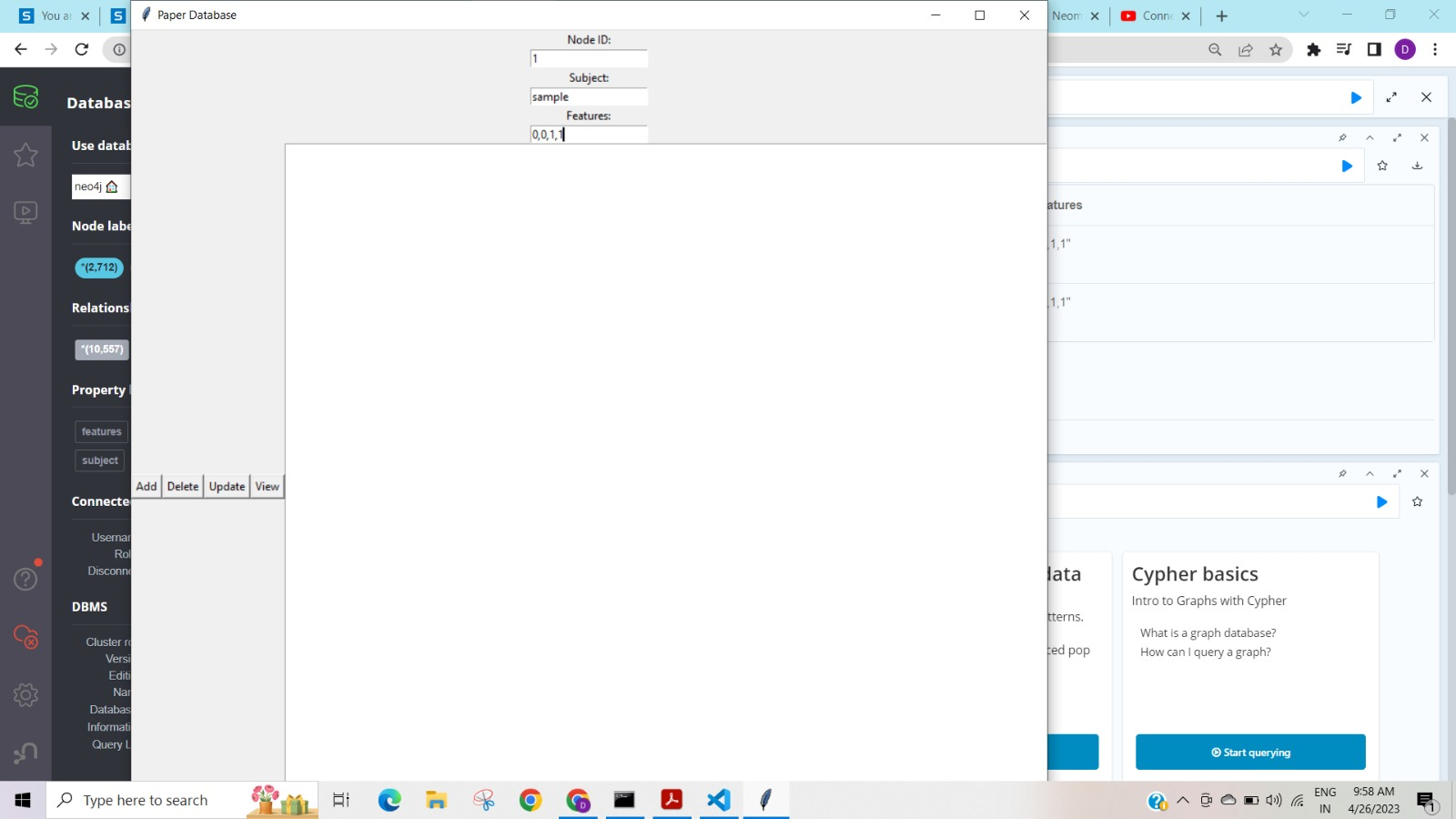
update\_button.pack(side=tk.LEFT)

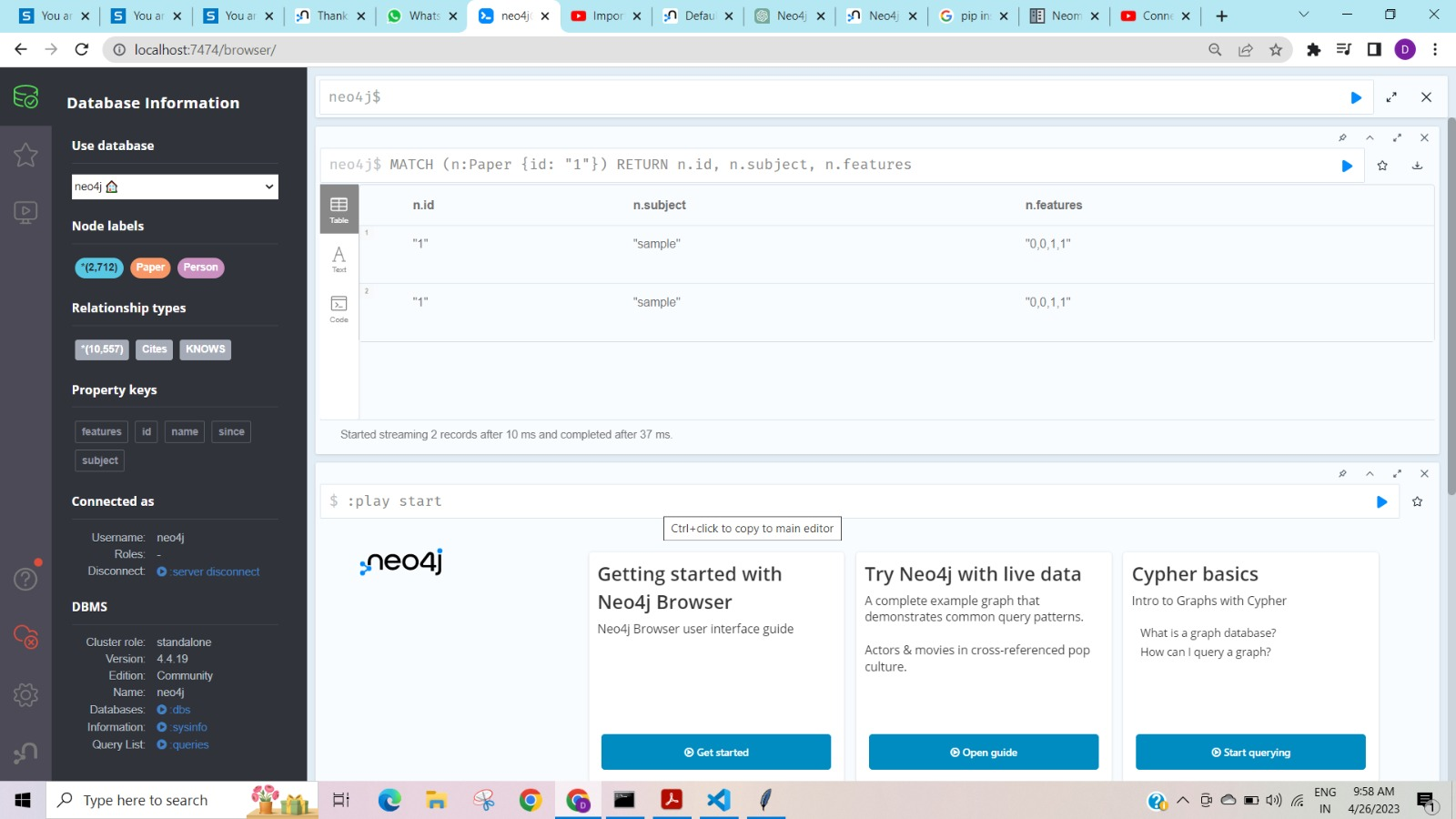
view\_button.pack(side=tk.LEFT)

results\_text.pack()

root.mainloop()

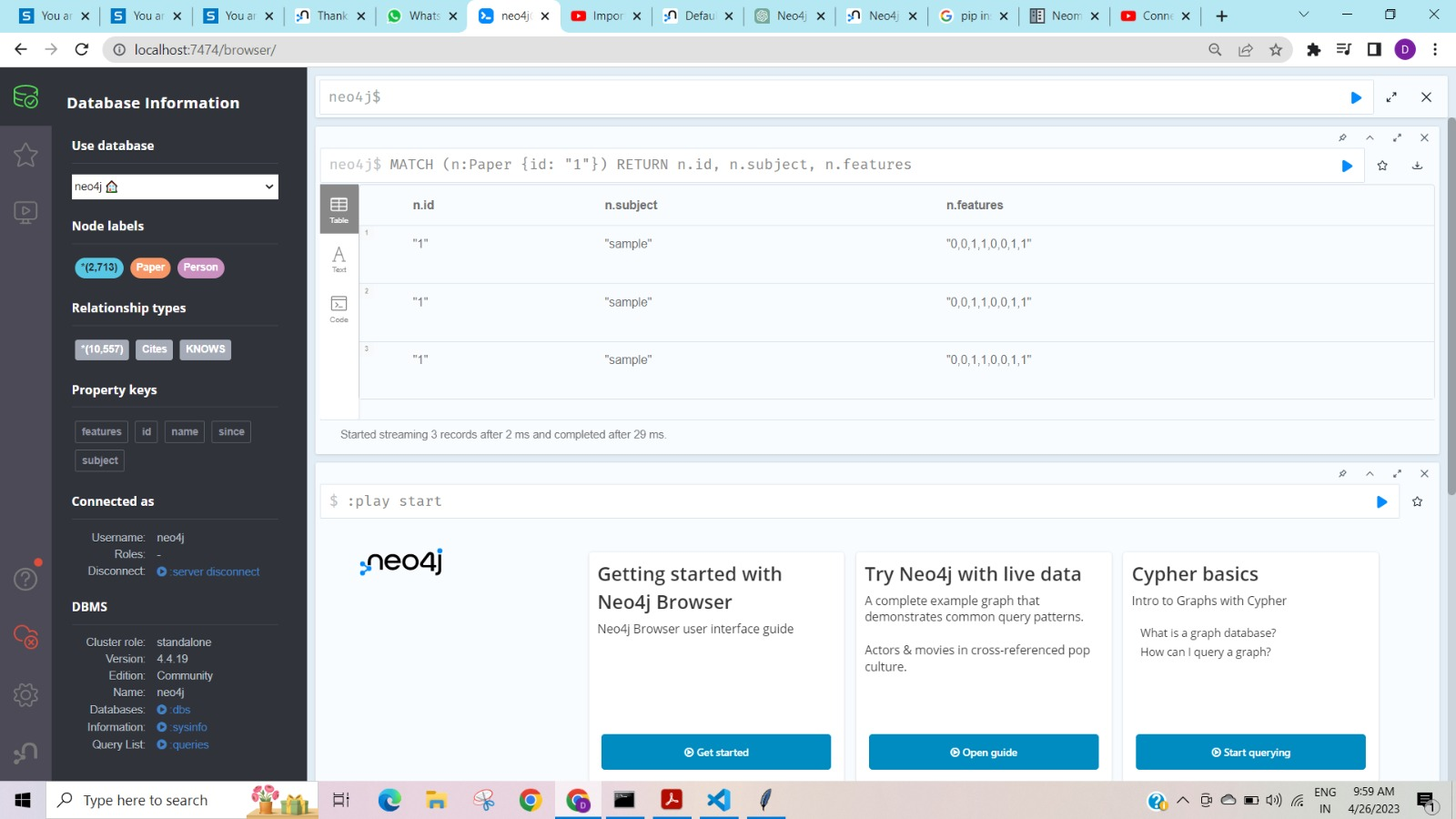
Adding data to database:





Updating data in database:





View data from database:

