Experiment no 11

Name:- Nehal D. Ughade PRN no :- 21520009

Course name:- ADSL Lab Date- 27/04/2023

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.
3. Load this data using Neo4j Data Browser
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)

Introduction:

A graph database stores nodes and relationships instead of tables, or documents. Data is stored just like you might sketch ideas on a whiteboard. Your data is stored without restrictingit to a pre-defined model, allowing a very flexible way of thinking about and using it.

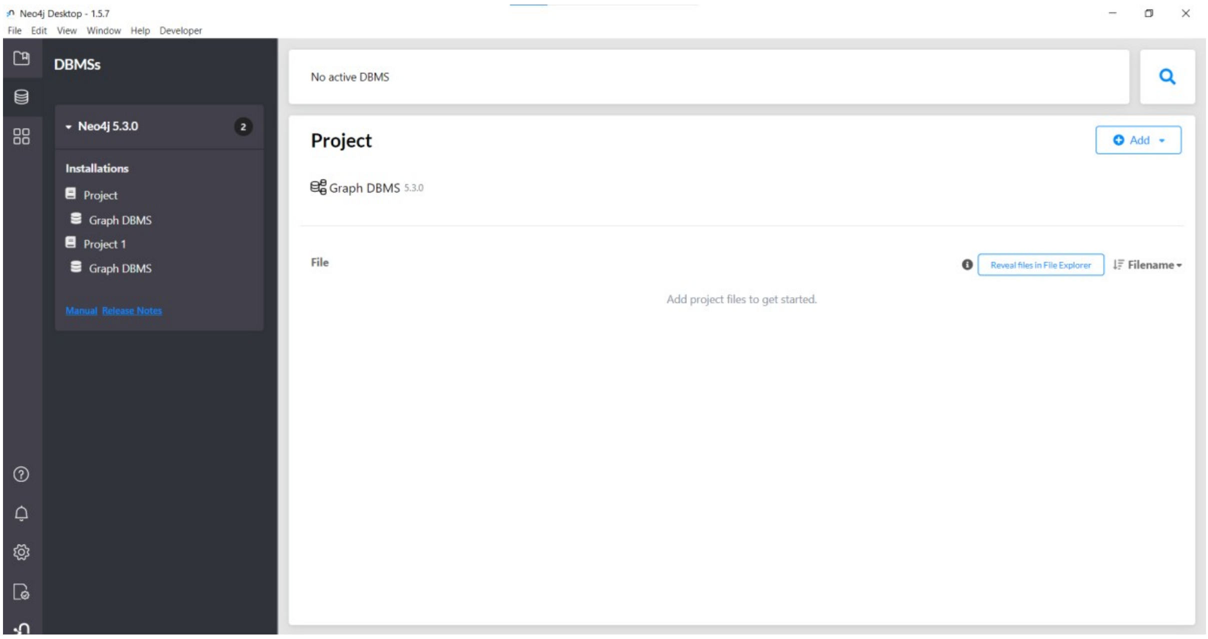
Theory:

Neo4j is the world's leading graph database. The architecture is designed for optimal management, storage, and traversal of nodes and relationships. The graph database takes a property graph approach, which is beneficial for both traversal performance and operations runtime.

Cypher

Cypher is Neo4j's graph query language that allows users to store and retrieve data from the graph database. It is a declarative, SQL-inspired language for describing visual patterns in graphs using ASCII-art syntax. The syntax provides a visual and logical way to match patterns of nodes and relationships in the graph. Cypher has been designed to be easy tolearn, understand, and use for everyone, but also incorporate the power and functionality of other standard data access languages

Downloaded neo4j Desktop



No active DBMS

Add project files to get started.

Load this data using Neo4j Data Browser

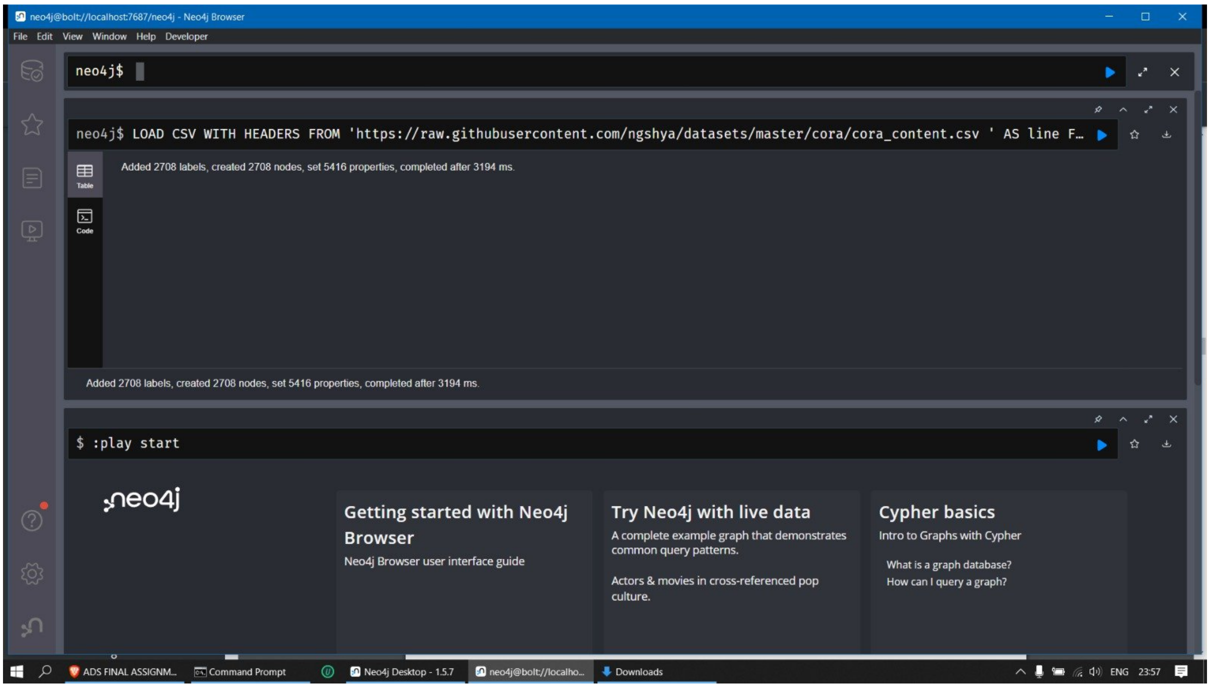
LOAD CSV WITH HEADERS FROM

'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 createsa Paper node for each row in the CSV file.



neo4j@bolt://localhost:7687/neo4j - Neo4j Browser
File Edit View Window Help Developer
Table
Code
neo4j
Getting started with Neo4j Browser
Neo4j Browser user interface guide
Try Neo4j with live data
A complete example graph that demonstrates common query patterns.
Actors & movies in cross-referenced pop culture.
Cypher basics
Intro to Graphs with Cypher
What is a graph database?
How can I query a graph?
ADS FINAL ASSIGNM...
Command Prompt
Neo4j Desktop - 1.5.7
neo4j@bolt://localho...
Downloads
ENG
23:57

```
neo4j$
```

```
neo4j$ LOAD CSV WITH HEADERS FROM 'https://raw.githubusercontent.com/ngshya/datasets/master/cora/cora_content.csv ' AS line F…
```

```
Added 2708 labels, created 2708 nodes, set 5416 properties, completed after 3194 ms.
```

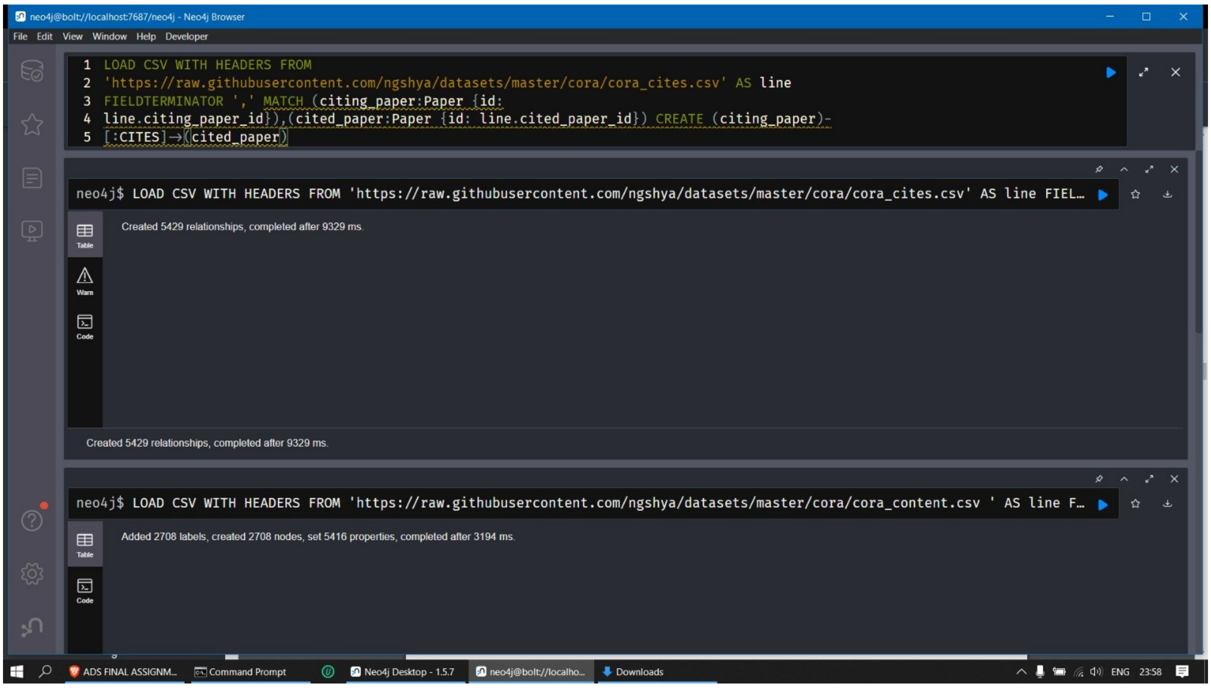
```
Added 2708 labels, created 2708 nodes, set 5416 properties, completed after 3194 ms.
```

```
$ :play start
```

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 createsa CITES relationship between Paper nodes based on the data in the CSV file.


neo4j@bolt://localhost:7687/neo4j - Neo4j Browser
File Edit View Window Help Developer
Table
Warn
Code

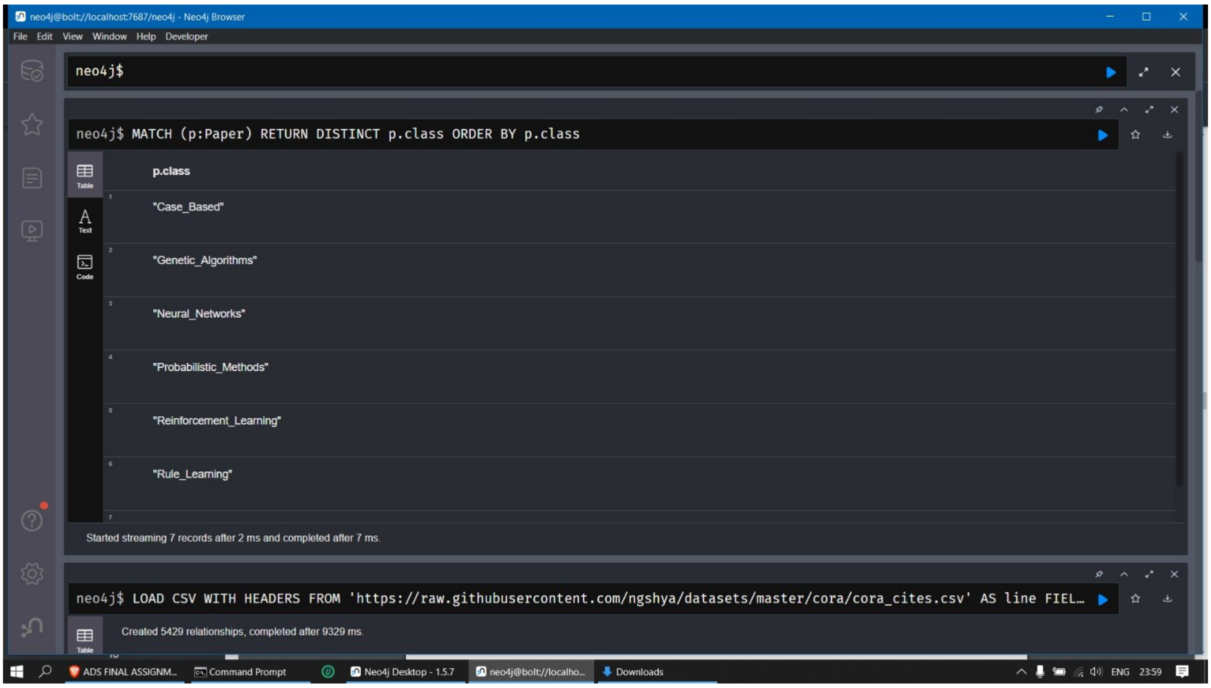
```
LOAD CSV WITH HEADERS FROM
'https://raw.githubusercontent.com/ngshya/datasets/master/cora/cora_cites.csv' AS line
FIELDTERMINATOR ',' MATCH (citing_paper:Paper {id:
line.citing_paper_id}),(cited_paper:Paper {id: line.cited_paper_id}) CREATE (citing_paper)-
[:CITES]→(cited_paper)
```

```
neo4j$ LOAD CSV WITH HEADERS FROM 'https://raw.githubusercontent.com/ngshya/datasets/master/cora/cora_cites.csv' AS line FIEL…
Created 5429 relationships, completed after 9329 ms.
Created 5429 relationships, completed after 9329 ms.
```

```
neo4j$ LOAD CSV WITH HEADERS FROM 'https://raw.githubusercontent.com/ngshya/datasets/master/cora/cora_content.csv ' AS line F…
Added 2708 labels, created 2708 nodes, set 5416 properties, completed after 3194 ms.
```

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 sortsthem in ascending order.


neo4j@bolt://localhost:7687/neo4j - Neo4j Browser
File Edit View Window Help Developer
Table
Text
Code

```
neo4j$ MATCH (p:Paper) RETURN DISTINCT p.class ORDER BY p.class
```

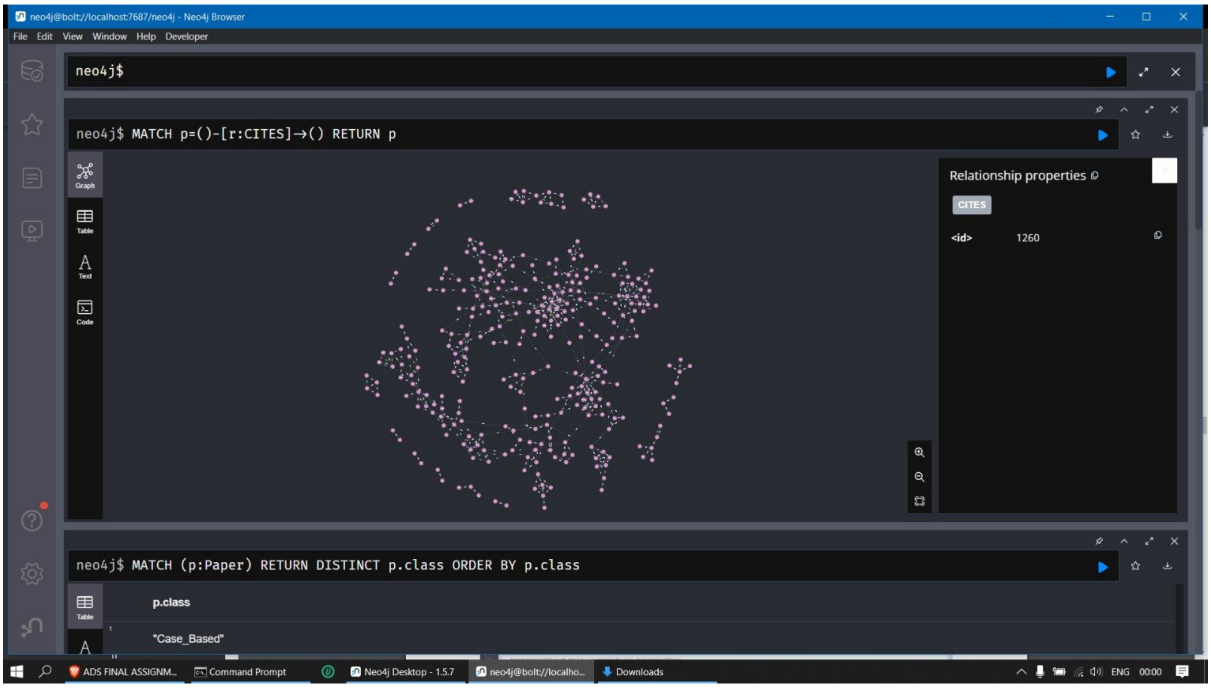
	p.class
1	"Case_Based"
2	"Genetic_Algorithms"
3	"Neural_Networks"
4	"Probabilistic_Methods"
5	"Reinforcement_Learning"
6	"Rule_Learning"
7	

Started streaming 7 records after 2 ms and completed after 7 ms.

```
neo4j$ LOAD CSV WITH HEADERS FROM 'https://raw.githubusercontent.com/ngshya/datasets/master/cora/cora_cites.csv' AS line FIEL…
Created 5429 relationships, completed after 9329 ms.
```

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

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



```
neo4j$
```

```
neo4j$ MATCH p=()-[r:CITES]→() RETURN p
```

Relationship properties

CITES

<id> 1260

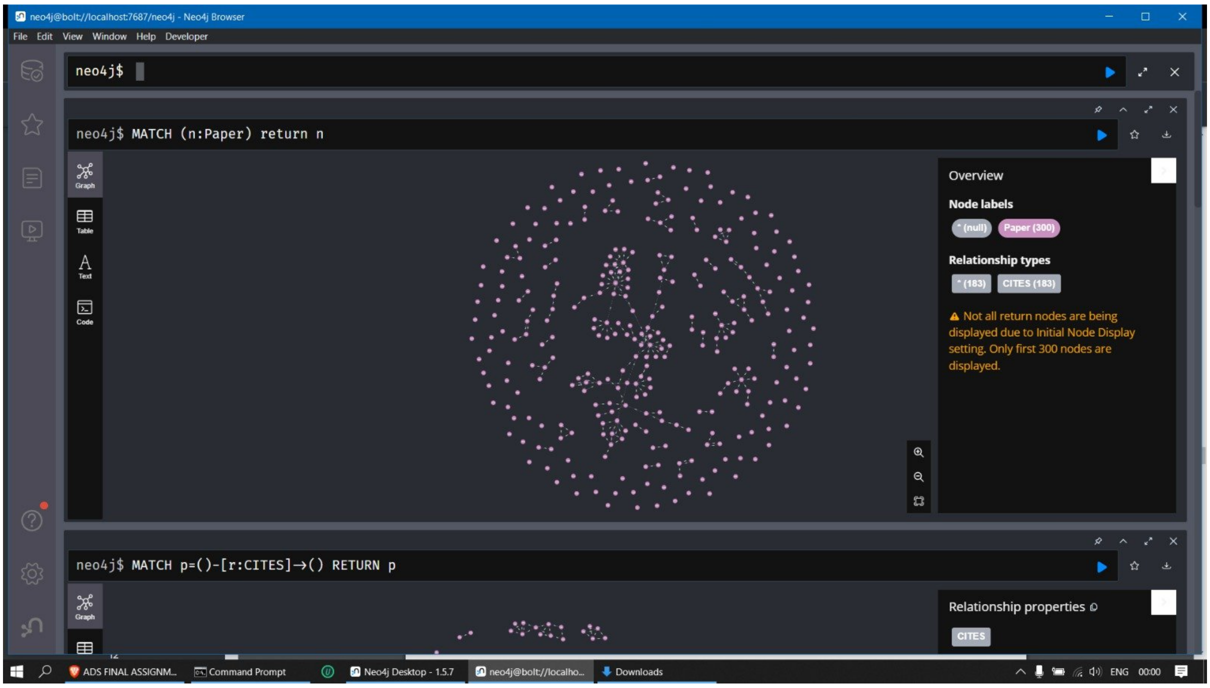
```
neo4j$ MATCH (p:Paper) RETURN DISTINCT p.class ORDER BY p.class
```

p.class

"Case_Based"

MATCH (n:Paper) return n

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



```
neo4j$
```

```
neo4j$ MATCH (n:Paper) return n
```

Overview

Node labels

* (null) Paper (300)

Relationship types

* (183) CITES (183)

⚠ Not all return nodes are being displayed due to Initial Node Display setting. Only first 300 nodes are displayed.

```
neo4j$ MATCH p=()-[r:CITES]→() RETURN p
```

Relationship properties

CITES

Design the python based desktop application for any kind of search on above database. Theapplication 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 = Nonetry:

            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 elseself.
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 resultblog.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 executesthe 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 tothe Neo4j database.