Title: Understanding configuration and working of Neo4J.

Aim: Installation and configuration of Neo4J Graph Database and Designing Research Papers Database

Procedure:

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 restricting it to a pre-defined model, allowing a very flexible way of thinking about and using it.

Theory:

Neo4j

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

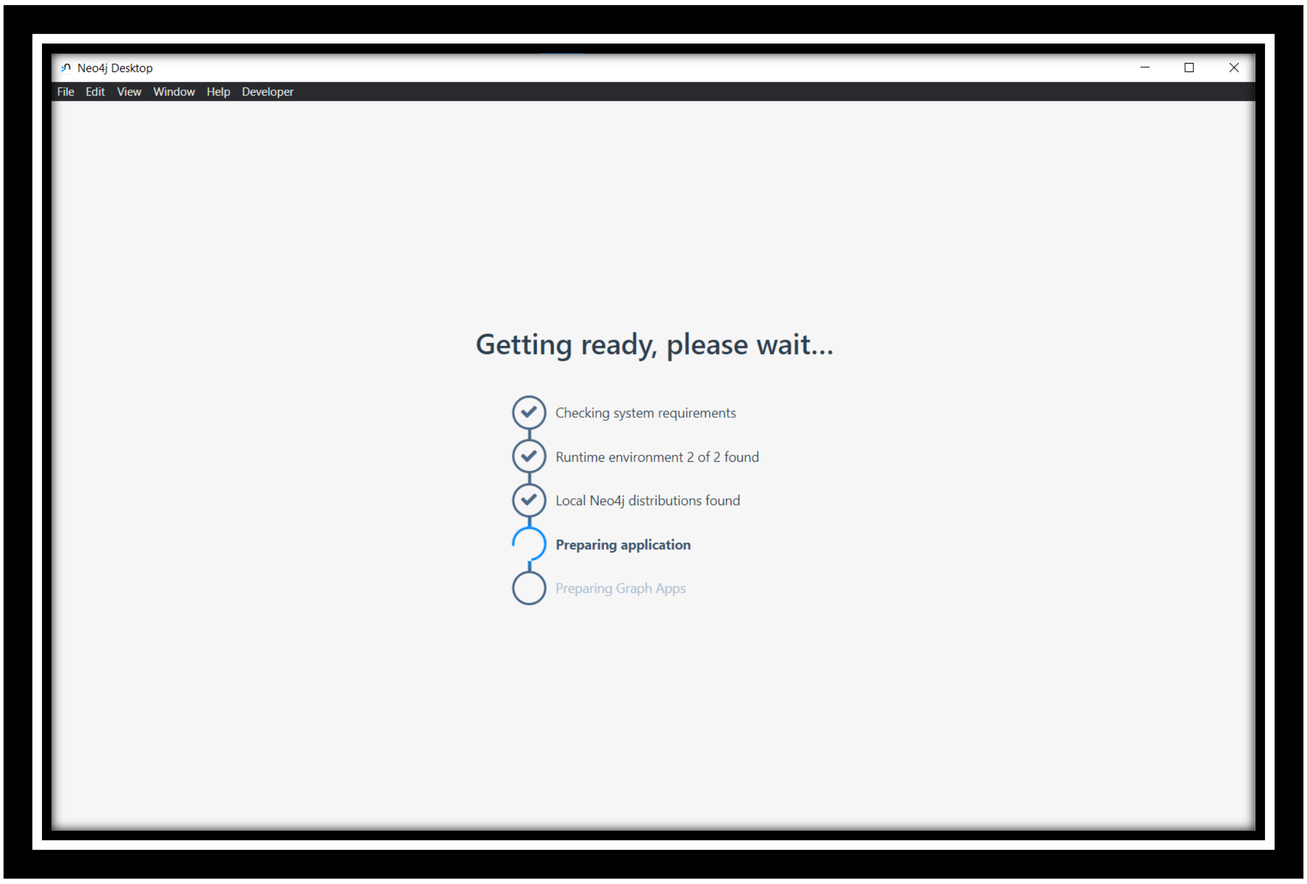
Cypher is Neo4j’s graph query language that allows users to store and retrieve data from the graph database. It is a declarative, SQL-inspired language for describing visual patterns in graphs using ASCII-art syntax. The syntax provides a visual and logical way to match patterns of nodes and relationships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

Procedure:

Step 1: Download JDK.

Step 2: Downloaded neo4j community server edition

Step 3: Downloaded neo4j Desktop



Getting ready, please wait...

Checking system requirements

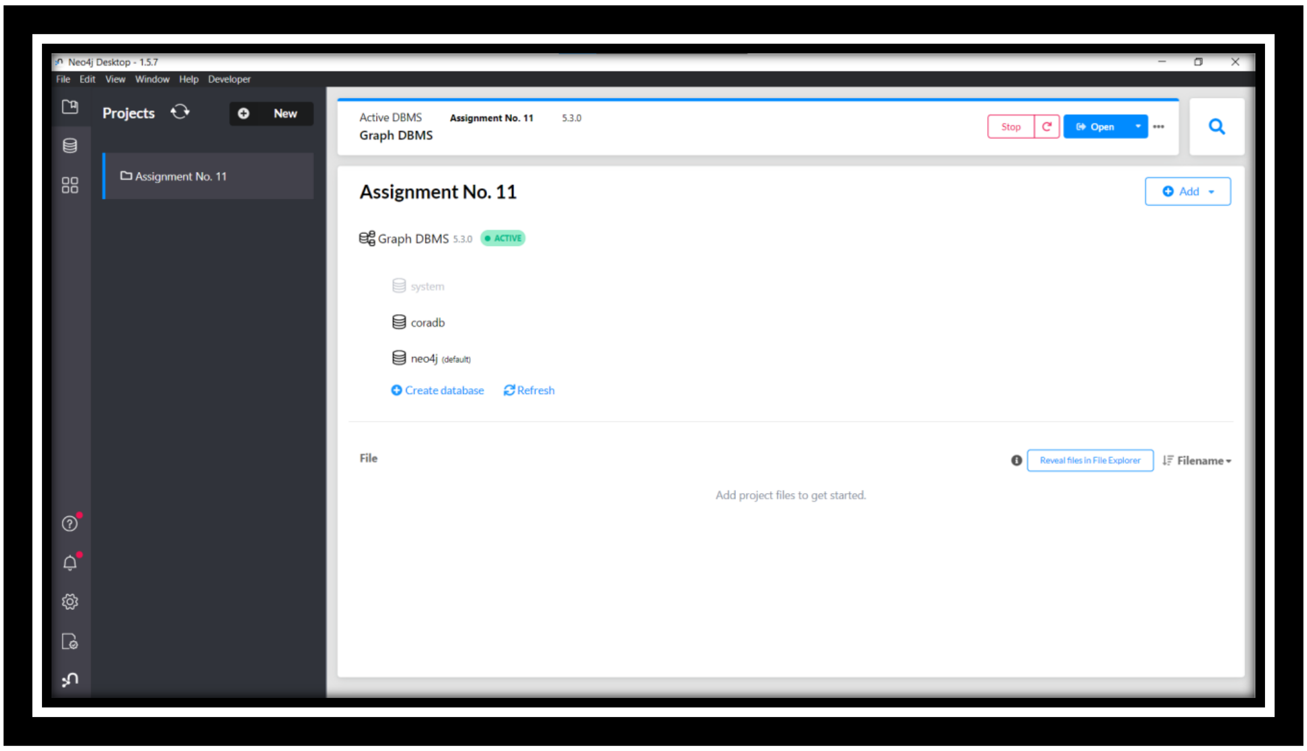
Runtime environment 2 of 2 found

Local Neo4j distributions found

Preparing application

Preparing Graph Apps

Creating a new project named Assignment11



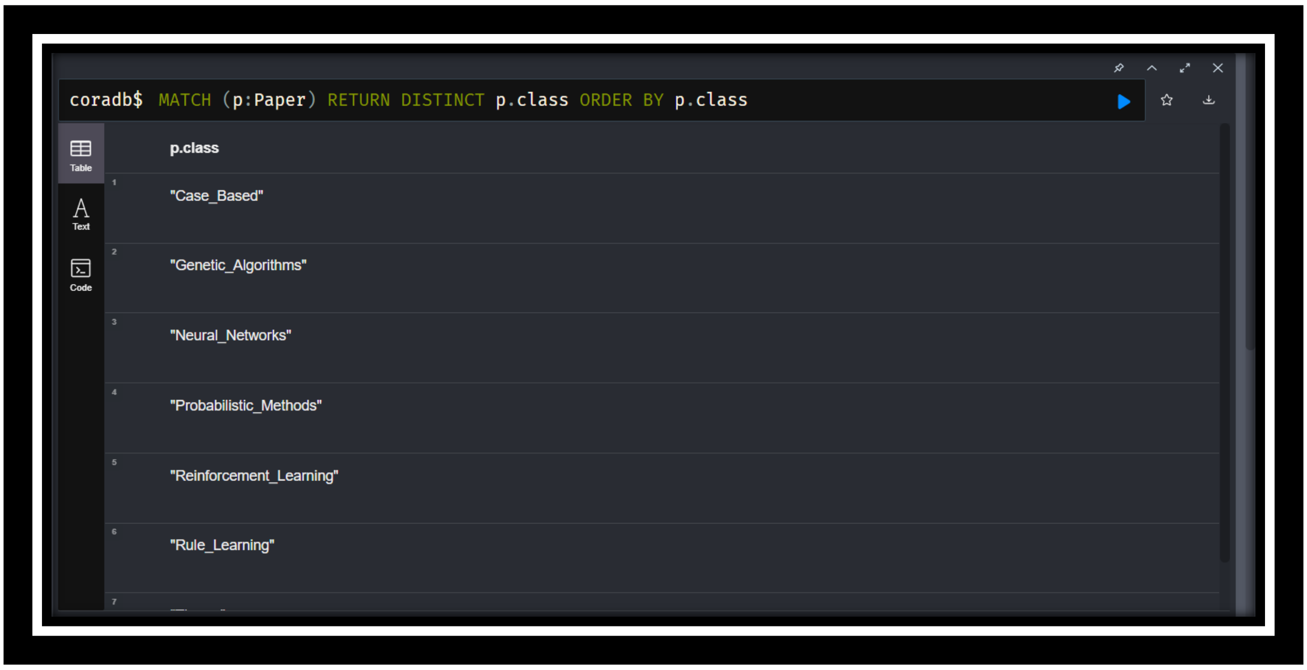
Python Application

```
from neo4j import GraphDatabase

class Neo4jConnection:

    def __init__(self, uri, user, pwd):
        self.__uri = uri
        self.__user = user
        self.__pwd = pwd
        self.__conn = None
        try:
            self.__conn = GraphDatabase.driver(self.__uri, auth=(self.__user, 
self.__pwd))
        except Exception as e:
            print("Failed to create the conn:", e)
    def close(self):
        if self.__conn is not None:
            self.__conn.close()

    def query(self, query, db=None):
        assert self.__conn is not None, "conn not initialized!"
        session = None
        response = None
        try:
            session = self.__conn.session(database=db) if db is not None else
self.__conn.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="Mahesh@123")


# res = conn.query("show databases")
# print(res)
# conn.query("CREATE OR REPLACE DATABASE coradb")


query_string = '''
CALL {
LOAD CSV WITH HEADERS FROM
'https://raw.githubusercontent.com/ngshya/datasets/master/cora/cora_content.cs
v'
AS line FIELDTERMINATOR ','
CREATE (:Paper {id: line.paper_id, class: line.label}) }
'''
conn.query(query_string, db='coradb')

query_string = '''
CALL{
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)}
'''
```

```
conn.query(query_string, db='coradb')

# a = '1152448'
# b = '2354'

query_string = " MATCH (p1:Paper { id: '" + a + "' }),(p2:Paper { id: '"+b+"'
}), path = shortestPath((p1)-[*..15]->(p2)) return path "

print(query_string)

res = conn.query(query_string,db='coradb')

print(res)
```

Opened using Neo4j browser

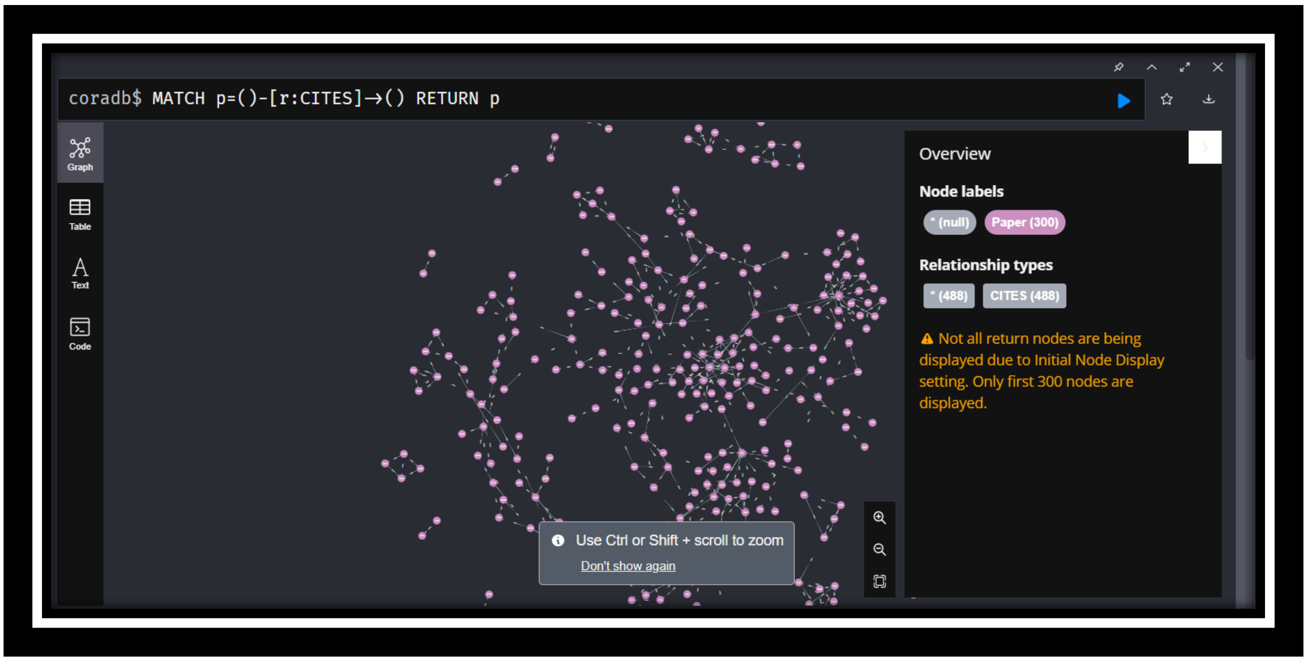
MATCH (p:Paper) RETURN DISTINCT p.class ORDER BY p.class



```
coradb$ 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	

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



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

Overview

Node labels

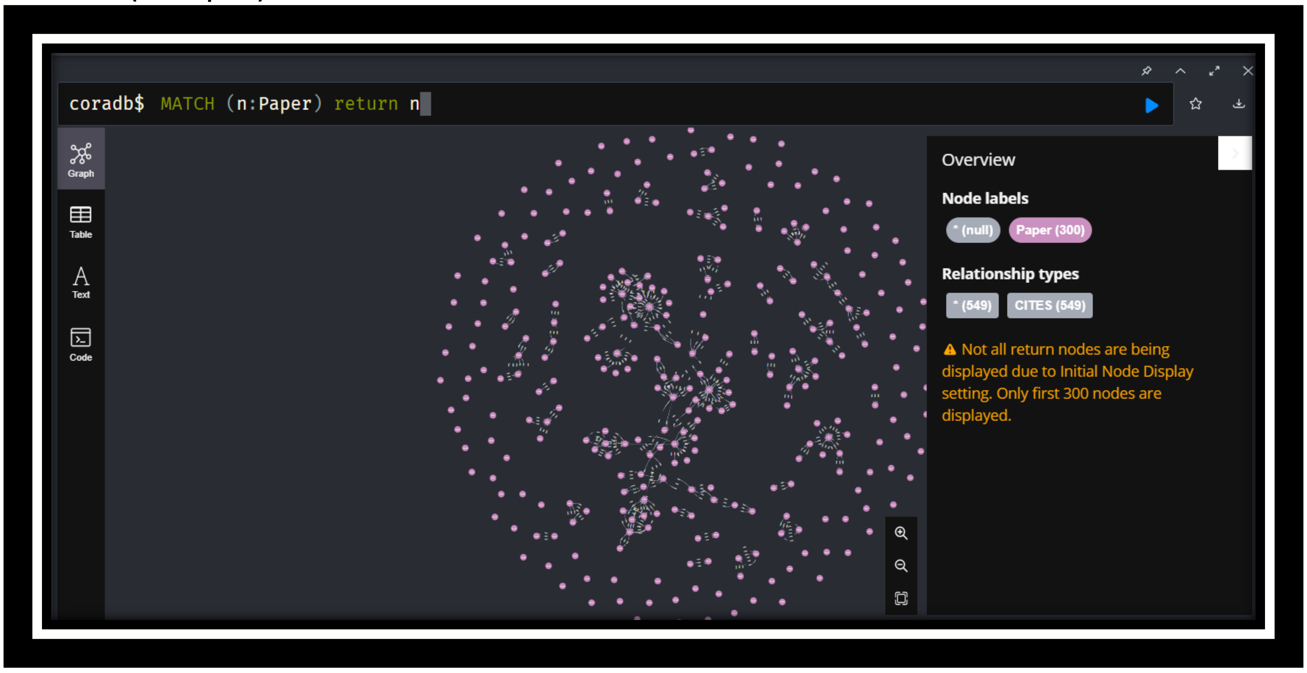
* (null) Paper (300)

Relationship types

* (488) CITES (488)

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

MATCH (n:Paper) return n



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

Overview

Node labels

* (null) Paper (300)

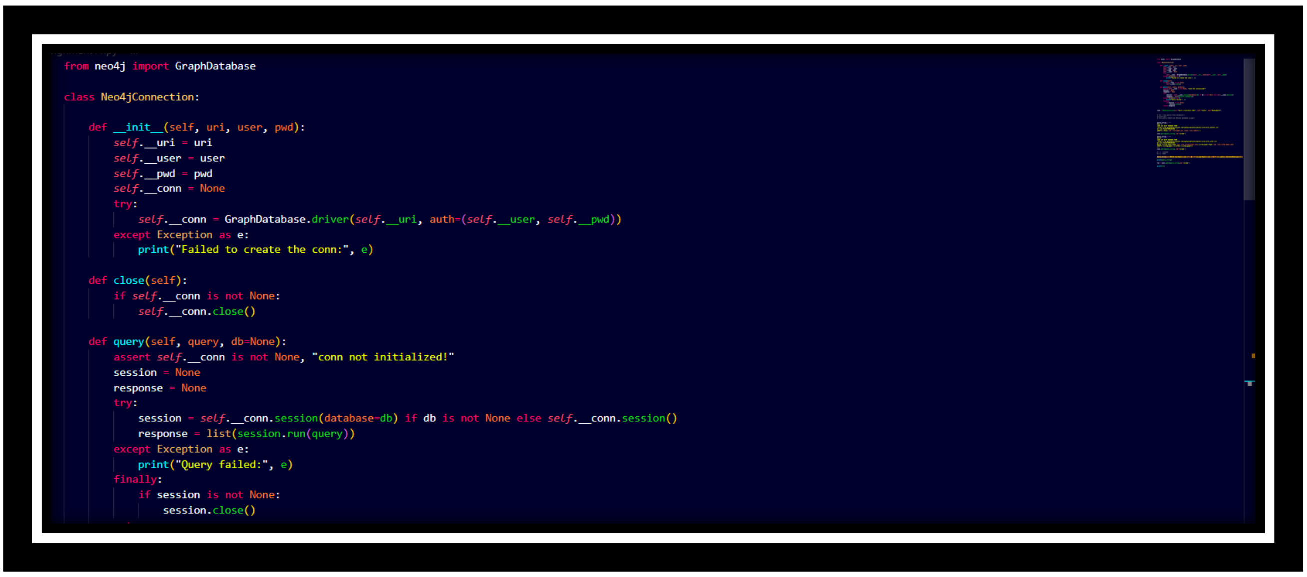
Relationship types

* (549) CITES (549)

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

Python Desktop Application:

Output:



```
from neo4j import GraphDatabase

class Neo4jConnection:

    def __init__(self, uri, user, pwd):
        self.__uri = uri
        self.__user = user
        self.__pwd = pwd
        self.__conn = None
        try:
            self.__conn = GraphDatabase.driver(self.__uri, auth=(self.__user, self.__pwd))
        except Exception as e:
            print("Failed to create the conn:", e)

    def close(self):
        if self.__conn is not None:
            self.__conn.close()

    def query(self, query, db=None):
        assert self.__conn is not None, "conn not initialized!"
        session = None
        response = None
        try:
            session = self.__conn.session(database=db) if db is not None else self.__conn.session()
            response = list(session.run(query))
        except Exception as e:
            print("Query failed:", e)
        finally:
            if session is not None:
                session.close()
```

<u>Conclusion:</u> Successfully installed and configured neo4j graph database and run python desktop application for the given dataset to achieve required aim.

<u>References:</u>

https://neo4j.com/docs/cypher-manual/current/