Gael Blanchard

Lloyd Massiah

Required libraries

Import csv -for csv file manipulation

Import pandas as pd

Import sys,getopt,pprint

Import matplotlib.pyplot as plt - for pie graphs

Import collabheader- defined the help function call in the main Python file

Import collabgraphs - holds code for the pie graph returning functions

From pymongo import MongoClient, ASCENDING- to initialize mongoDB

pymongo.ASCENDING deprecated

From py2neo import G,a,import cyphererror - To authorize neo4j database and utilize it cypher error allows us to detect errors during the runtime of giving data to the neo4j

Program Workflow

\_\_\_\_\_\_\_\_\_\_\_\_\_\\_\_\_CSV\_\_\_/\_\_\_\_\_\_\_\_\_\_\_\_\_\_

User given csv files give file names

To testcollab.py which allows us to

Generate the data into the database

A.Neo4j. B.mongoDB

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

| |

| |

| |

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

The relations in the | The user info

Collab.net | handles the primarily

Handles the. |entity(username)

Relation. | based queries

Based queries. |ex.organization name Trusted colleagues |search, username etc

The first function. |

When queries are called in the python's

Command Line interface we access the respective database associated with it then

Return the data in a legible format to the CLI

Used queries:

Mongodb primary query

cnDB.files\_data.find({"address":{"$regex":searchaddress}})

Using the standard query syntax along with regex we can access the address in the database and therefore return it. Just by replacing the address field with interest.name we can access even su document fields

gettheuser = cypher.execute("MATCH (user:User {username:{a}})-[:interestedin]->(interest)<-[getvalue:interestedin]-(otheruser:User) MATCH (otheruser:User)-[:workedat]->(org:Org) MATCH (anotheruser:User)-[:workedat]->(b:Org) MATCH (org:Org)<-[r:distance]->(b:Org) WHERE r.miles <= 10 MATCH (b:Org)<-[:workedat]-(user:User) WITH otheruser,collect(b.name) as organization,collect(interest.name) as matchedinterest,collect(r.miles) as distance,collect(getvalue.level) as interestlevel RETURN otheruser.username, otheruser.first\_name, otheruser.last\_name, organization, matchedinterest, distance, interestlevel", a=userinput)

print(gettheuser)

The preceding query is the query designed to answer the first function. We basically reaccess each relation from one user to the other user using a relation chain. This is the standard syntax. The most unique we do in this query is add the get value to define the weight of the interest. The collect statements were used for readability