**Graph Theory Project 2017:**

**GMIT Timetable.**

**Name:** Conor Tighe.

**Id:** G00314417.

Introduction

For a project in the module Graph Theory I was told to take the timetable found on the GMIT website and redesign it into my own schema that will allow users to interact with the elements within the timetabling system. The goal of this project is to hopefully improve the timetabling system and create an infrastructure that would allow detailed and fast searching using the design patterns we studied in the module and my own critical thinking skills. I have decided to cover the Software development and Digital Media semester 2 timetables as this will allow me to create a large schema, rich in data to interact with without making the scope of the project unrealistic. In this document, I will explain the process taken in creating the schema, the challenges encountered and the design and architecture of the model. I will also discuss the technologies/methods used to complete the task and how I used them.

Technologies Used

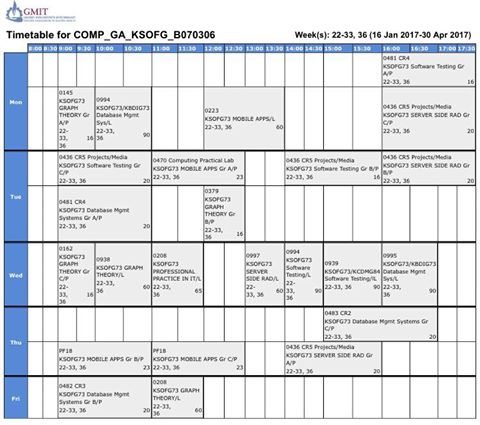
**Neo4j:** This is a NoSQL Graph database management system which can be used to display data as nodes and their edges as relationships. It is currently the most popular graph database. Neo4j displays its data-sets in eye-pleasing bubble layout that lets large amounts of information easy to read and interact with.

**Python:** This is a programming language first released in 1991 but still very popular. I decided to use this language to complete any programming parts of the project as it is easy to use yet powerful making it perfect for developing a web crawler.

**Microsoft Excel:** This is a program for developing spreadsheets and manipulating the data found in the rows and columns of the spreadsheet.

Planning and architecture

Before creating anything in Neo4j I wanted to have a design in mind of how my database would represent the timetable and the relationships between elements. To start designing I had to study the GMIT timetable and the page that displayed it first. After looking at the information displayed I decided the course codes where what I would base my database around. I included the lecturer name even though these where not necessary to a timetable system and can be found on other parts of the site. I focused on elements like room, module, time, groups and course codes as these are elements that would give my database powerful and effective search functionality when looking up a class. The data like level and year could be made into sub-information under course using labels. After reviewing the timetable below I decided the best way to go about extracting the information is dividing it up by the days along the left and the time along the top. The reason for this would enable by database to have a more in detailed search by not just looking for a matching time but also the day once I feed the information into my database.



Extracting data

After now figuring out what I needed to make up my timetable I started researching ways to take information off web sites. It wasn’t as easy as simply copying and pasting the data as all the timetables where made up of html table cells found across multiple pages. This meant that manually retrieving the data would be too time consuming. After using google chromes inspect source feature to review the code that made up the timetable and examined where the data I wanted was located. I noticed all the data was hardcoded the exact same way on each page making a perfect pattern for a web crawler.

Python seemed like a good choice for this job as it offers a clean and simple syntax with a wide variety of library’s. This would mean I could spend minimal time on writing an effective web scraping program as that is not the goal of the project but will certainly help me reach it. The clean syntax would also make dealing with large amounts of data easier. After looking online I read an article that discussed web scraping with a library called beautiful soup and then began reading the documentation on the library. I wrote the following program which has comments through-out explaining how it retrieves the data from the GMIT timetables html.

from urllib.request import urlopen as uReq

import null as null

from bs4 import BeautifulSoup as soup

from bs4 import Comment

gmitUrl = 'file:///C:/LINK-TO-URI-OF-TIMETABLE.html'

# connect to gmit and parse the timetables html

gmitTimetable = uReq(gmitUrl)

page\_html = gmitTimetable.read()

gmitTimetable.close()

# The name of the cvs the data will be stored in

filename="NAME OF TIMETABLE AND THE DAY.csv"

file = open(filename,"w")

# Hearders of the CSV file

headers = "Room, Module, Group, Capacity, Classes\n"

file.write(headers)

# html parse

page\_soup = soup(page\_html, "html.parser")

# Divide the timetable into blocks to target the day

blocks = page\_soup.findAll("table", {"class":"object-cell-args"})

i = 0

# Loop through each block on the timetable and extract its data

for block in blocks:

i = i % 3

block\_res = block.findAll("td", {"align": "left"})

#print(block\_res)

if i == 0:

room = block\_res

room = room[0].text

room = room.replace('[<td align="left">', "")

room = room.replace('</td>]', "")

file.write(room.replace(',', "-") + ",")

print("room", room)

elif i == 1:

module = block\_res

module = module[0].text

module = module.replace('[<td align="left">', "")

module = module.replace('</td>]', "")

if "Gr A/P" in module:

group = "A"

elif "Gr B" in module:

group = "B"

elif "Gr C" in module:

group = "C"

elif "Gr D" in module:

group = "D"

elif "Gr E" in module:

group = "E"

else:

group = "All Groups"

print("module", module)

file.write(module + ",")

file.write(group + ",")

print(group)

elif i == 2:

classes = block\_res

classes = classes[0].text

classes = classes.replace('[<td align="left">', "")

classes = classes.replace('</td>]', "")

print("classes", classes)

file.write(classes.replace(',', "-") + ",")

roomCap = block.findAll("td", {"align": "right"})

try:

roomCap = roomCap[0].text

roomCap = roomCap.replace('[<td align="right">', "")

roomCap = roomCap.replace('</td>]', "")

print("cap", roomCap)

file.write(roomCap + "\n")

except IndexError:

print("")

i += 1

#file.write(room.replace(',', "-") + "," + module + "," + classes.replace(',', "-") + roomCap + group)

The program grabs the information from the targeted row on the timetable and writes it to a created CSV file that is titled with the name of the course and day. Once everything is sorted into CSV files by the day I then created a master CSV file with a day header and dumped all the data into that and using a day column to track what day the data is from. To get the data from the html source I had to store the html page locally and then link the URI as the GMIT login blocked the web scraper from directly accessing it online. I then searched each module manually on learnonline.gmit to find the relevant lecturers. After I had their names paired them to the corresponding module using CTRL-F in Excel.

After having everything separated into a CSV file a began to use regular expression to extract information inside the module title to create headers such as the course code and Level. I then created a course csv file, this would hold all the related information to the course like the number of students enrolled and if it falls under Software development or Digital media. I then divided the csv containing all the timetable data into csv’s that would hold everything related to the rooms, times and modules etc. as I originally planned earlier in the document. I indexed each new csv to keep track what data was related and named the column the **ClassId** in each file. This indexing would make it easy to tie the relationships together later in Neo4j. The course node would connect to a department node to help distinguish between software and digital media visually on the graph.

Developmental Stage:

In Neo4j there is a feature that allows us to upload a CSV file and then neo4j will automatically create nodes and labels from the data within the CSV. This was the main reason I divided up the csv’s into rooms, times etc. and I wanted neo4j to treat each as a separate node and use the headers as labels for the data stored within each one. To upload each csv separately I had to place them in the **import** folder which I created in the Neo4j folder and enter the following query’s in Neo4j:

LOAD CSV WITH HEADERS FROM "file:///Course.csv" AS row

CREATE (n:Course)

SET n = row,

n.Enrolled = toInteger (row.Enrolled)

LOAD CSV WITH HEADERS FROM "file:///time.csv" AS row

CREATE (n:Time)

SET n = row

LOAD CSV WITH HEADERS FROM "file:///room.csv" AS row

CREATE (n:Room)

SET n = row

LOAD CSV WITH HEADERS FROM "file:///Modules.csv" AS row

CREATE (n:Module)

SET n = row

LOAD CSV WITH HEADERS FROM "file:///size.csv" AS row

CREATE (n:Class\_Size)

SET n = row

LOAD CSV WITH HEADERS FROM "file:///day.csv" AS row

CREATE (n:Day)

SET n = row

LOAD CSV WITH HEADERS FROM "file:///dept.csv" AS row

CREATE (Department)

SET n = row

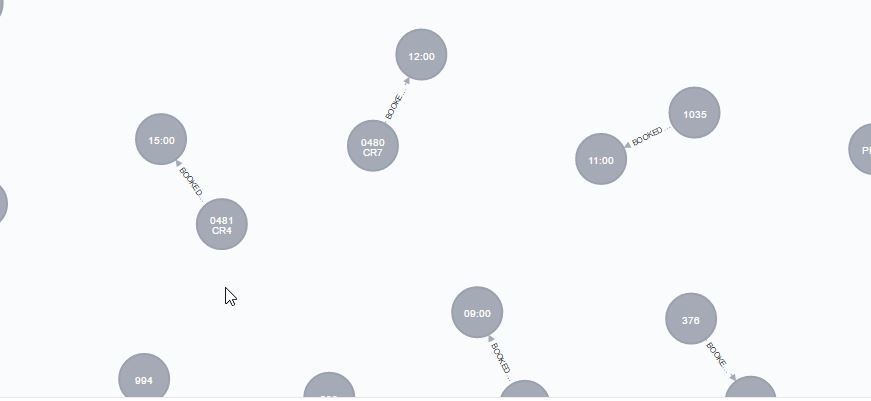
Once all the csv’s where uploaded and all the data was now available on Neo4j I began creating the relationship that would tie my information together. The first relationship I implemented was what time a room was booked for use. I implemented this by matching the **ClassId** found in each node.

MATCH (p:Room),(c:Time)

WHERE p.ClassId = c.ClassId

CREATE (p)-[:BOOKED\_AT]->(c)

After creating the relationships, I could retrieve all the rooms and the time there in use with the following line:

MATCH p=()-[r:BOOKED\_AT]->() RETURN p

I then used similar queries to set up the relationships between the other nodes. Here are some examples of relationships found in my database.

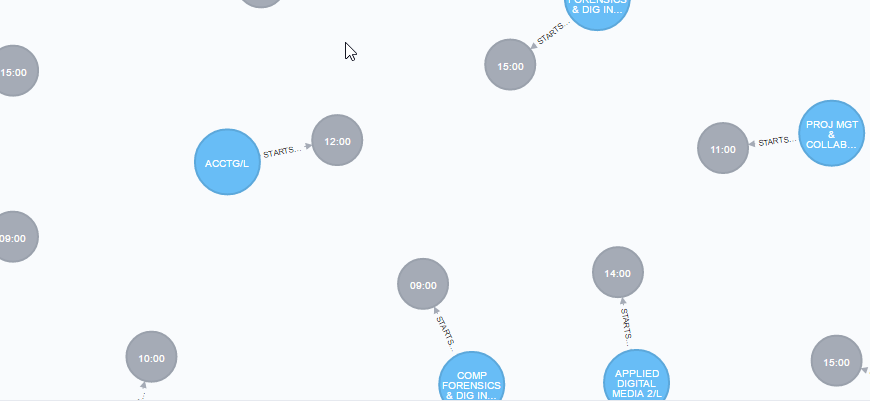
**Relationship between Module and Time:**

MATCH (p:Module),(c:Time)

WHERE p.ClassId = c.ClassId

CREATE (p)-[:STARTS\_AT]->(c)

MATCH p=()-[r:STARTS\_AT]->() RETURN p



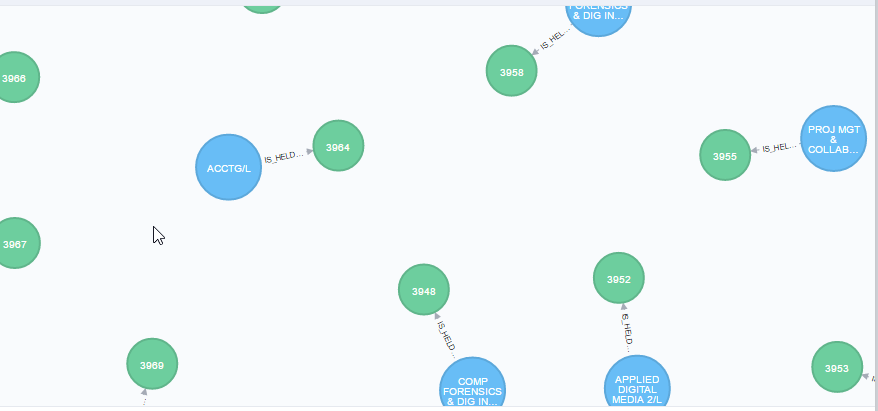
**Relationship between Module and Room:**

MATCH (p:Module),(c:Room)

WHERE p.ClassId = c.ClassId

CREATE (p)-[:HELD\_IN]->(c)

MATCH p=()-[r:HELD\_IN]->() RETURN p



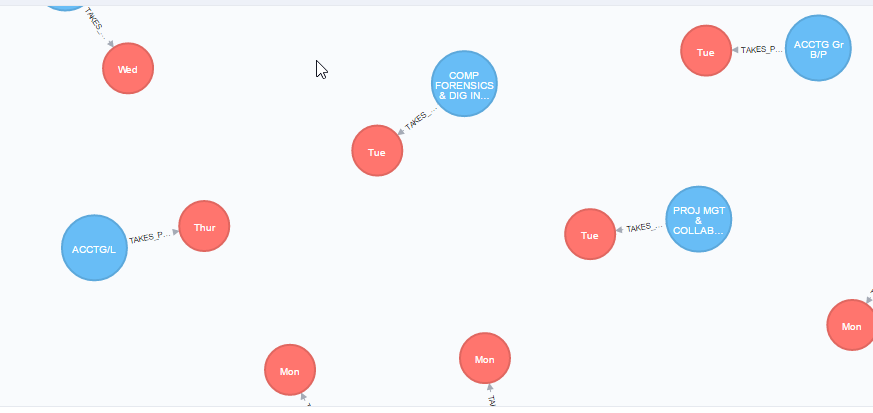
**Relationship between Module and Day:**

MATCH (p:Module),(c:Day)

WHERE p.ClassId = c.ClassId

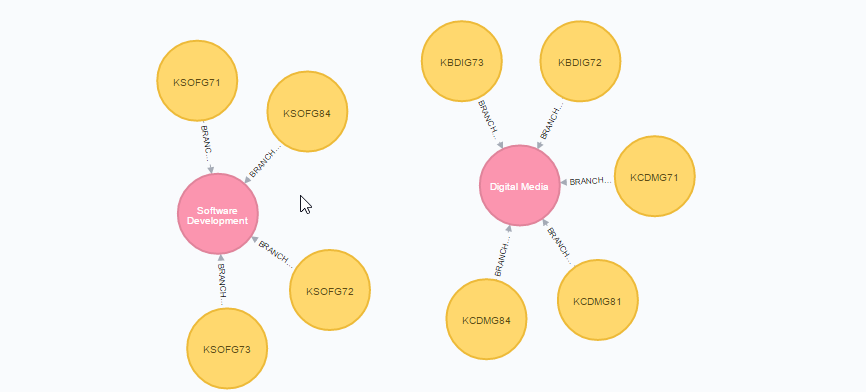
CREATE (p)-[:TAKES\_PLACE\_ON]->(c)

MATCH p=()-[r:TAKES\_PLACE\_ON]->() RETURN p

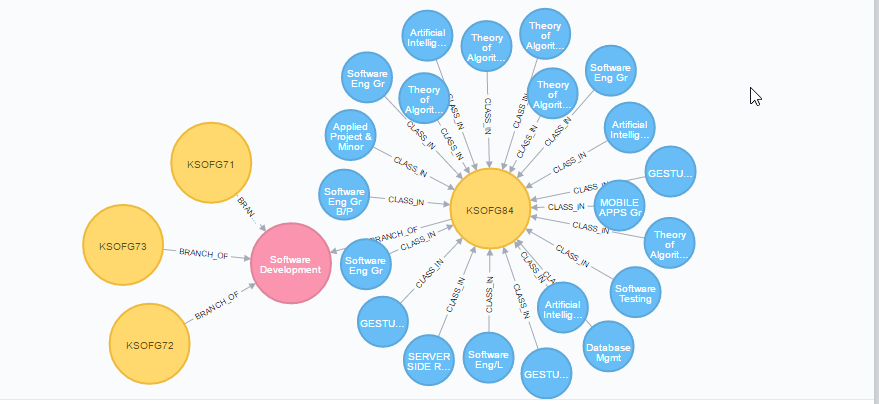


Then to create a separation between the Digital media courses and the software Development courses, I created two category nodes to represent software and digital media. These are connected to nodes that represent the course code, year and level. From those nodes, you can see the modules that fall under that course code in the modules to the course node.

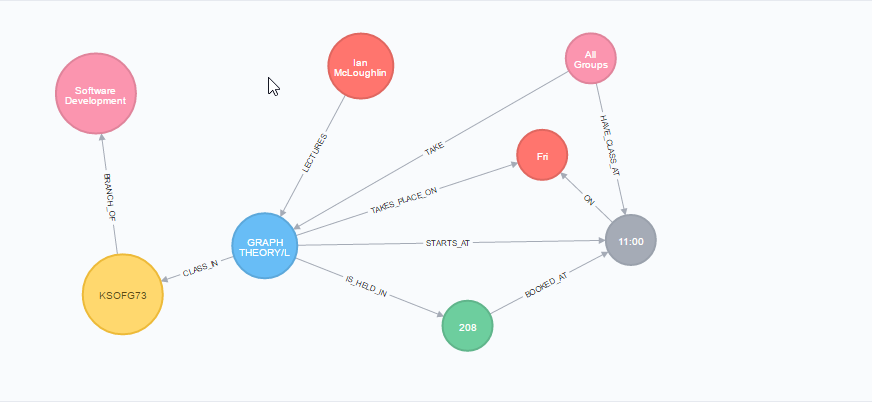
**The relationship between the course department and course code:**



**The relationship between course code and modules:**



**Complete depth of GMIT Timetable relationships:**



Here is an extract of the schema to show all the vertices and edges for my groups lecture at 2am for this module. From looking at the graph we know that KSOFG73 is a branch of the software development course, or to be exact we could read the labels:



From there we can tell that the module Graph theory is a Class in Year 3 of the Level 7 Software development course and is taught by Ian McLoughlin, this class starts at 2pm on Monday in the prefab PF06. 16 students are meant to attend this(I asked about that number at the bottom of the timetable cell and was told that’s the class size but feel there is more that attend that class, this is the data from GMIT’s official website) who are all part of Group A.

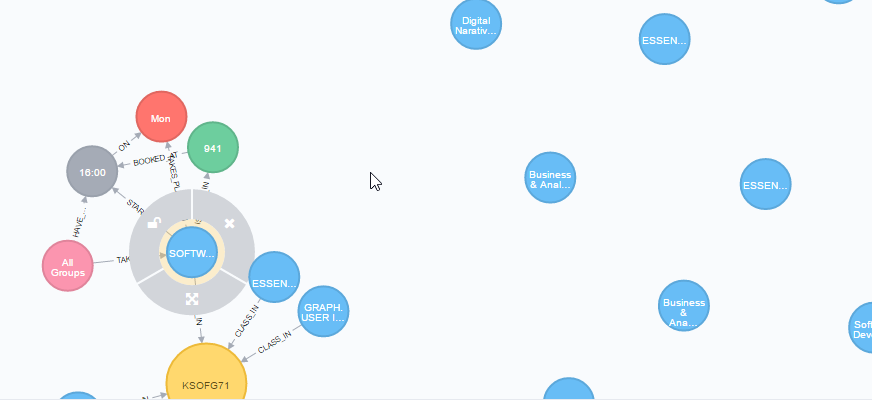
Testing the database

In order to see if this is an effective design for a timetable database I will include some queries that show what this schema is capable off.

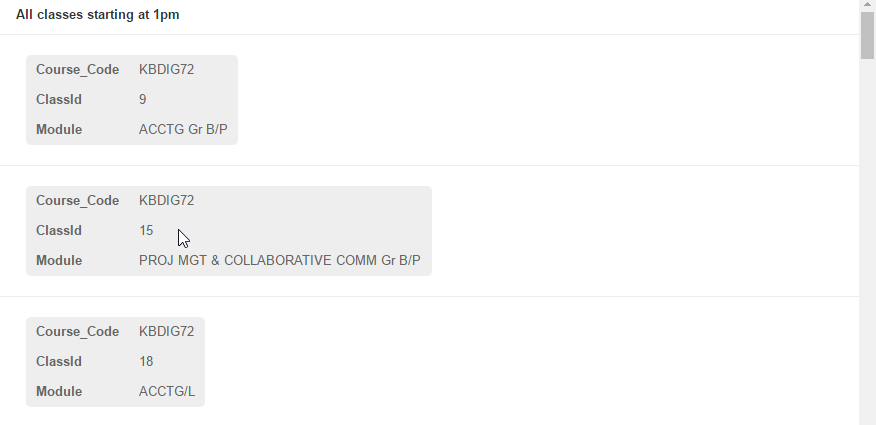
**Show all modules that take place on a Monday:**

MATCH (p:Module),(c:Day)

WHERE c.ClassId = p.ClassId AND c.Day = "Mon"

Return p

**Show all modules starting at 1pm:**

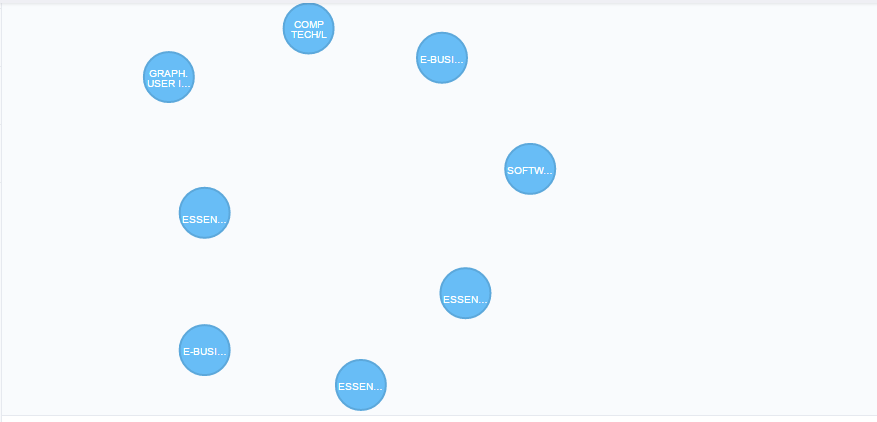


**Show all modules that take place in room 941**

MATCH (p:Module),(g:Room)

WHERE g.ClassId = p.ClassId AND g.Room = "941"

Return p



**All Modules in Software Development:**

MATCH (g:Course), (p:Module)

WHERE g.Course\_Code = p.Course\_Code AND g.Course\_Name = "Software Development"

Return p



Known Bugs and design flaws

After coming the final steps in my project, I realised that there are some duplicate nodes in my rooms and lecturers. Although this doesn’t affect the practicality of my database or the ease of finding a class, this is still disappointing as its bad practice to have these within a database. After researching I discovered there’s no official way to fix this in neo4j once the relationships are set up so I’d have to find a way to redesign these 2 nodes.

Attempts at a solution: I tried starting a new database and using MERGE when importing the csv files.

LOAD CSV WITH HEADERS FROM "file:///Module.csv" AS row1

LOAD CSV WITH HEADERS FROM "file:///Lecturers.csv" AS row2

Merge (p:Module{ModuleId: row1.Module\_Name})

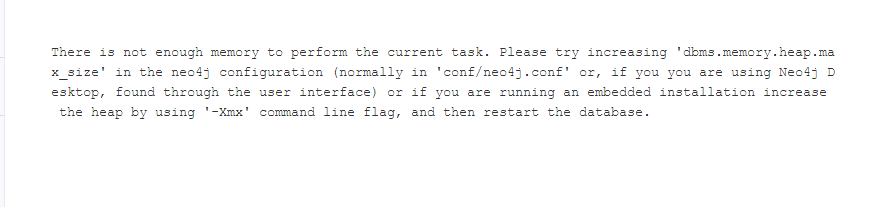
ON CREATE set p.ClassId = row1.ClassId

Merge (m:Lecturer{Module:row2.Lecturer})

ON CREATE set m.ClassId = row2.ClassId

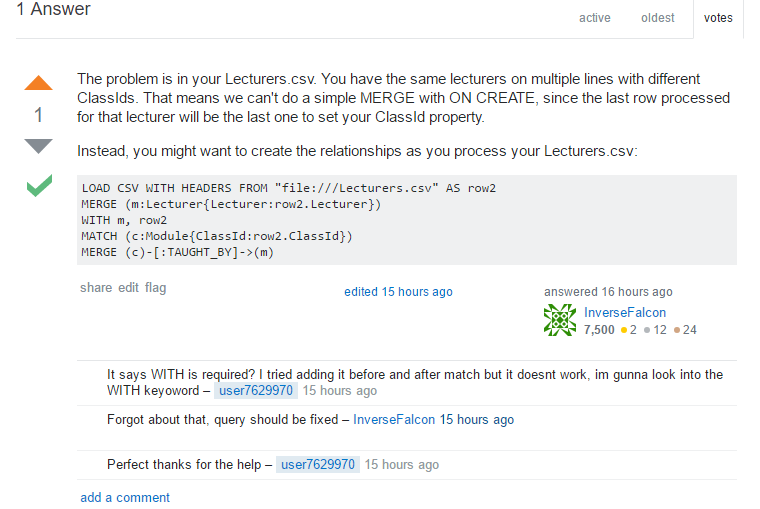
Merge (p)-[:TAUGHT\_BY]->(m)

From what I read in the Neo4j documentation this should work, but the command I made caused the following:

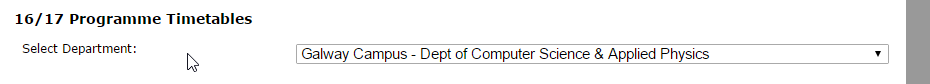


After numerous attempts at fixing usign MERGE statements the issue I decided to post on StackOverflow asking for help, after afew hours and discussing back and forth with some site members in the comments someone pointed out why my merge wasn’t working. I then remodeled my graph around there answer now that I understood what was the cause of the duplicates.

The solution I was given in the post:



I also noticed a seperation in in digital media and software development was illogical and it would make more sense to join these in some was as this database was really meant to be a representation of the school of Computing in GMIT.



Solution: Create a node reppresenting our department head who is in charge of the department of computing and the courses they offer, this would actually be practical for students as this is who you would contact if you had a issue with the timetable or a reason you couldn’t attend a class.

Final Build

Now knowing what was wrong with the project a deleted all the lecture and room nodes and began adding them back in with the new statement and making new relationships related to them.

LOAD CSV WITH HEADERS FROM "file:///room.csv" AS row2

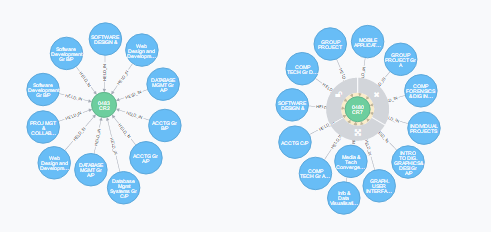
MERGE (m:Room{Room:row2.Room})

WITH m, row2

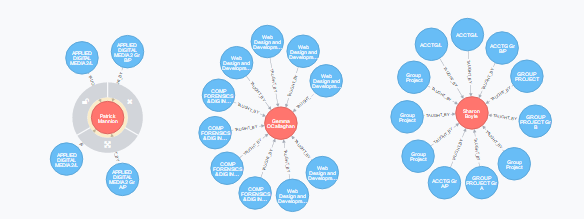
MATCH (c:Module{ClassId:row2.ClassId})

MERGE (c)-[:HELD\_IN]->(m)

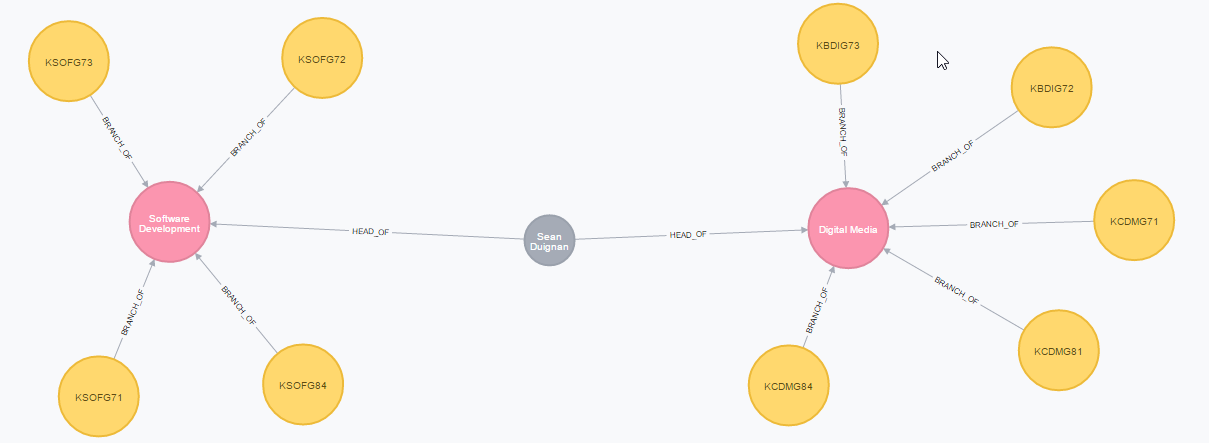
Some nodes from the new HELD\_IN relationship:



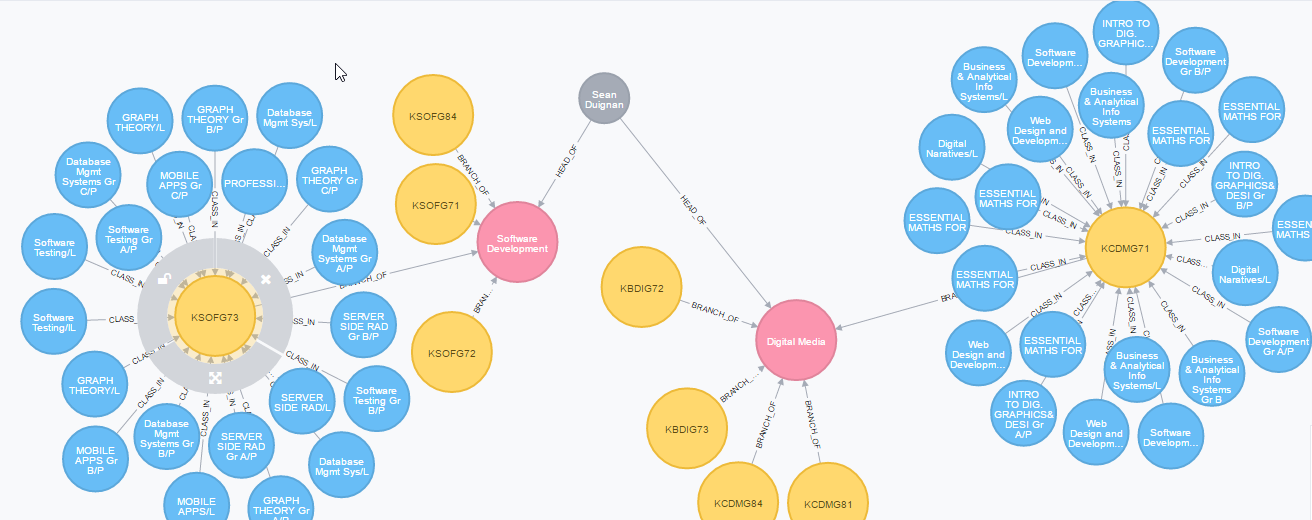
Some nodes from the new TAUGHT\_BY relationship:

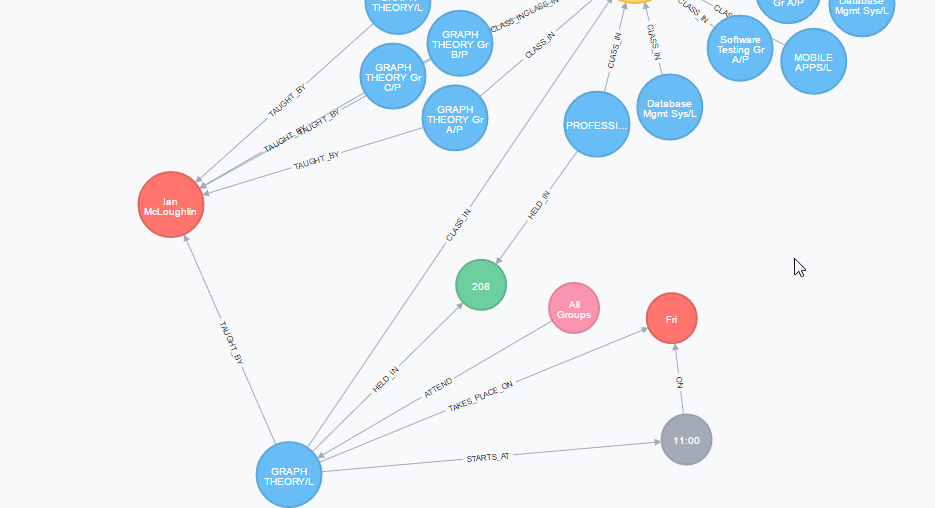


Now with the the room and lecturers remodeled I added a final node called head of department to join the two departments together, this node had our head of departments contacts as labels.



Final database screenshots:





Conclusion

I feel the final product of this project achieves all the goals originally set by my myself along with the ones outlined In the brief. I feel this sturture is more efficient then the hardcoded cell pages we have for our current timetabe online. The database is ment to act as a Timetable that provides information revlivent to students and staff off GMIT in a fast and efficient way and I think my attempt does exactly that. My database makes it easy for anyone or any system to retirve things like Day,Time,Module,Room and Course Code etc. which are all common things people would access a timetable for and contain accurate data.This database could be implmented in all schools across GMIT and its campuses. The full database can be found on the following GitHub repository:

<https://github.com/ConorTighe1995/GMIT-Timetable>

References

StackOverflow:

[http://stackoverflow.com/questions/43553880/how-can-i-get-rid-of-the-duplicates-in-my-database-and-still-be-able-to-link-the/43554360?noredirect=1#comment74161073\_43554360](http://stackoverflow.com/questions/43553880/how-can-i-get-rid-of-the-duplicates-in-my-database-and-still-be-able-to-link-the/43554360?noredirect=1%23comment74161073_43554360)

Neo4j Documentation:

<https://neo4j.com/docs/>

Beautiful Soup Documentation:

<https://www.crummy.com/software/BeautifulSoup/bs4/doc/>

Beautiful Soup tutorial:

<https://www.youtube.com/watch?v=XQgXKtPSzUI>