

CSY3024 DATABASE 3

AS1: Portfolio



janurary 17, 2024

UNIVERSITY OF NORTHAMPTON

Elizabeth Aning-Amponsah 18412818

Table of Contents

[Portfolio 2](#_Toc161535357)

[Research 2](#_Toc161535358)

[Weekly Learning Activities 3](#_Toc161535359)

[Week3: Cypher Query language1 3](#_Toc161535360)

[Week4: Cypher Query language2 7](#_Toc161535361)

[21](#_Toc161535362)

[Week5: cypher query language 3 – CSV 26](#_Toc161535363)

[Week 6: Cypher Query language 4 34](#_Toc161535364)

[41](#_Toc161535365)

# Portfolio

## Research

Neo4j is one of many database systems available to us but is one of the leading graph databases and has often been the case for years now. Neo4j is a No-SQL database that is Java based, optional for schemas and hugely scalable.

There are many other graph databases that are open sourced like Neo4j such as JanusGraph, ArangoDB, OrientDB, Virtuoso etc. There are many others, but these graph databases seem to be just as popular as Neo4j with Neo4j being one of the Most popular amongst them all.

However, these graph databases differ amongst each other in several aspects including aspects such as their data models, query languages, scalability and the cases of usage.

Neo4j is a native graph database, meaning it stores and processes data in the form of nodes, relationships, and properties. It’s well known in its handling of highly connected data. Its query language is Cypher, which has been specifically designed for graph traversal and data manipulation. Neo4j is mostly known for its scalability to which neo4j offers a horizontal scaling – practically splitting the load between different servers and enabling the addition of more instances of machines/nodes without changing existing specifications allowing for the sharing of processing power and load balancing. This means that Neo4j allows for the handling of large-scale graph data. This is most common in its usages for applications that require complex relationship management such as social networks, recommendation systems and even fraud detection. An example of the neo4j usage is eBay who can rely on Neo4j’s drive recommendations and promotions.

JanusGraph is a distributed graph database that supports a variety of storage backends including Apache Cassandra and google cloud Bigtable. JanusGraph is based on the property graph model which is a data model which is also used in neo4j – it represents data as a collection of nodes, relationships and properties just the same and allows for flexible and efficient representation of highly connected nodes. JanusGraph uses the Apache TinkerPop Gremlin query language which is also a graph traversal language. Just like Neo4j, JanusGraph allows for the handling of large-scale graphs as it has been designed for horizontal scalability. As well as this, just like neo4j, JanusGraph its uses are suitable for applications requiring real-time graph analytics such as recommendations systems and fraud detection.

However, OrientDB is a multi-model database that supports not only graph models but also document and object data models. It does so by combining the features of a graph database with the flexibility of a document database. OrientDB supports the usage of SQL-like queries for document and graph data, but it also uses a specialised query language called Gremlin for graph traversal. Additionally, OrientDB supports both single server and distributed data, offering horizontal scalability through clustering and replication. Its common use cases are management systems and recommendation engines as it is suitable for applications that require a flexible data model.

Out of the lot, Neo4j is undeniably the most popular and widely adopted graph database with its large user base, active community, and long-standing industry support, as well as it often being the first choice for developers and organisations who tend to look for a more reliable graph database solution. However, JanusGraph does excel in scalability and distributed processing, making it well suited for large-scale graph analytics and processing tasks. Whilst OrientDB offers a versatile and flexible characteristic in supporting multiple data models within a single database, making it much suitable for applications that require hybrid data models. At the end of the day the choice between these databases is ultimately depending on specific project requirements such as scalability, data modelling needs, and performance as well as compatibility.

## Weekly Learning Activities

### Week3: Cypher Query language1

Clear the database

MATCH (n) detach delete n

Create simple graph nodes and relationships

First I started creating student, module and lecturer nodes

CREATE (:Student{name:"Elizabeth"}), (:Student{name:"Pelumi"}), (:Student{name:"Jack"}), (:Student{name:"Yomie"})

CREATE (:Lecturer{name:"James"}), (:Lecturer{name:"Carol"}), (:Lecturer{name:"Mark"}), (:Lecturer{name:"Rashid"})

CREATE (:Module{code:"CSY2030", name:"System design and development"}), (:Module{code:"CSY3024", name:"Database 3"}), (:Module{code:"CSY3025", name:"Artificial Intelligent"}), (:Module{code:"CSY3023", name:"Cybersecurity and Cryptography"})

A screenshot of a computer

Description automatically generated

Then I began to create some relationships between my nodes

MATCH (m:Module{code:"CSY3024"})

MATCH (s:Student{name:"Elizabeth"})

CREATE (s)-[:STUDIES]->(m)

MATCH (m:Module{code:"CSY3025"})

MATCH (s:Student{name:"Elizabeth"})

CREATE (s)-[:STUDIES]->(m)

MATCH (m:Module{code:"CSY3023"})

MATCH (s:Student{name:"Elizabeth"})

CREATE (s)-[:STUDIES]->(m)

MATCH (m:Module{code:"CSY2030"})

MATCH (s:Student{name:"Elizabeth"})

CREATE (s)-[:STUDIED]->(m)

MATCH (m:Module{code:"CSY2030"})

MATCH (s:Student{name:"Yomie"})

CREATE (s)-[:STUDIED]->(m)

MATCH (m:Module{code:"CSY3024"})

MATCH (s:Student{name:"Yomie"})

CREATE (s)-[:STUDIES]->(m)

MATCH (m:Module{code:"CSY3025"})

MATCH (s:Student{name:"Yomie"})

CREATE (s)-[:STUDIES]->(m)

MATCH (m:Module{code:"CSY3023"})

MATCH (s:Student{name:"Yomie"})

CREATE (s)-[:STUDIES]->(m)

MATCH (m:Module{code:"CSY3023"})

MATCH (s:Student{name:"Pelumi"})

CREATE (s)-[:STUDIES]->(m)

MATCH (m:Module{code:"CSY3024"})

MATCH (s:Student{name:"Pelumi"})

CREATE (s)-[:STUDIES]->(m)

MATCH (m:Lecturer{name:"Mark"})

MATCH (s:Student{name:"Pelumi"})

CREATE (m)-[:TEACHES]->(s)

MATCH (m:Lecturer{name:"Mark"})

MATCH (s:Student{name:"Elizabeth"})

CREATE (m)-[:TEACHES]->(s)

MATCH (m:Lecturer{name:"Mark"})

MATCH (s:Student{name:"Jack"})

CREATE (m)-[:TEACHES]->(s)

MATCH (m:Lecturer{name:"James"})

MATCH (s:Student{name:"Elizabeth"})

CREATE (m)-[:TEACHES]->(s)

MATCH (m:Lecturer{name:"Rashid"})

MATCH (s:Student{name:"Elizabeth"})

CREATE (m)-[:TEACHES]->(s)

MATCH (m:Lecturer{name:"Rashid"})

MATCH (s:Student{name:"Pelumi"})

CREATE (m)-[:TEACHES]->(s)

MATCH (m:Lecturer{name:"Carol"})

MATCH (s:Student{name:"Jack"})

CREATE (m)-[:TEACHES]->(s)

A screenshot of a computer

Description automatically generated

Delete some nodes and relationships:

Since all my nodes have at least one relationship, before I can delete a node, I would have to delete the relationships. there’s a few ways to do this, either using the WHERE clause and deleting by using its attribute such as id or directly by placing a variable in the relationship when matching it. Deleting it with the variable name is shorter and better than with id unless there were many similar nodes/duplicates it would be better to use the latter method.

Here I delete the teaches relationship between carol and jack by specifying the id of the relationship then deleting it

MATCH (m:Lecturer{name:"Carol"})-[t:TEACHES]->(s:Student{name:"Jack"})

WHERE id(t)=4331

delete t

whereas here I just stored a variable name ‘t’ to the teaches relationship between Mark and Pelumi and deleted the relationship using the variable name

MATCH (m:Lecturer{name:"Mark"})-[t:TEACHES]->(s:Student{name:"Pelumi"})

delete t

here I matched the lecturer node named Carol then deleted it.

MATCH (m:Lecturer{name:"Carol"})

DELETE m

MATCH (m:Lecturer{name:"Mark"})-[t:TEACHES]->(s:Student{name:"Jack"})

delete t

MATCH (m:Student{name:"Yomie"})-[t:STUDIES]->(s:Module{code:"CSY3024"})

delete t

MATCH (m:Student{name:"Yomie"})-[t:STUDIES]->(s:Module{code:"CSY3023"})

delete t

MATCH (m:Student{name:"Elizabeth"})-[t:STUDIES]->(s:Module{code:"CSY3025"})

delete t

MATCH (m:Lecturer{name:"James"})-[t:TEACHES]->(s:Student{name:"Elizabeth"})

delete t

MATCH (m:Lecturer{name:"Mark"})-[t:TEACHES]->(s:Student{name:"Elizabeth"})

delete t

MATCH (m:Lecturer{name:"Mark"})

A screenshot of a computer

Description automatically generateddelete m

### Week4: Cypher Query language2

Exercise 1:

* Create a module node with code: CSY3024

CREATE (Module{code:"CSY3024"})

Output: Created 1 node, set 1 property, completed after 1 ms.

* Run the same creation code again to see what happens

Created 1 node, set 1 property, completed after 1 ms.

When I match (n) return n, it shows me duplicates of the same node created shown in my screenshot below

A screenshot of a computer

Description automatically generated

Exercise 2:

Find out how to copy properties between relationships:

First, I created two person nodes with some properties, persons being Elizabeth and Samuel

CREATE (:Person{name:"Samuel", age:60, hobbies:"DIY"})

CREATE (:Person{name:"Elizabeth", age:24, hobbies:"dancing"})

Then I matched both nodes to find them in the database, provided they exist, I created a relationship between the person nodes, Elizabeth and Samuel, with a FATHER\_TO relationship being that Samuel is the father TO Elizabeth. I also added properties to the relationship being the relationship type and family of.

MATCH (p:Person{name: "Samuel"})

MATCH (q:Person{name: "Elizabeth"})

CREATE (p)-[:FATHER\_TO {relationshipType: "FATHER", FamilyOf:"Amponsah Family"}]->(q)

RETURN p, q

I did the same for the same two nodes but in the opposing direction, being Elizabeth is Daughter OF Samuel. I also added properties but for the property in this relationship, the name in Family of is different to the name in family of for the previous relationship

MATCH (p:Person{name:"Samuel"})

MATCH (q:Person{name:"Elizabeth"})

CREATE (q)-[:DAUGHTER\_OF {relationshipType: "Daughter", FamilyOf:"Aning Family"}]->(p)

RETURN p, q

Now I copy the properties in Samuels relationship to Elizabeth and this will effectively remove the relationship that was once there and replace it with the properties in Samuels relationship to Elizabeth relationship properties as shown below in the screenshot.

MATCH (p:Person{name: "Samuel"})-[oldRel:FATHER\_TO]->(q:Person{name: "Elizabeth"})

MATCH (q)-[newRel:DAUGHTER\_OF]->(p)

SET newRel = properties(oldRel)

RETURN p, q

A screenshot of a computer

Description automatically generated

To:

A screenshot of a computer

Description automatically generated

Exercise 3:

Create some nodes and relationships and practise the MERGE clause.

First I created 3 nodes for Lecturer and 3 nodes for Student with only name properties in each one

CREATE (:Student {name: "Elizabeth" }),

(:Student {name: "Kenneth" }),

(:Student {name: "Lisa" })

CREATE (:Lecturer {name: "James"}),

(:Lecturer {name: "Mandy"}),

(:Lecturer {name: "Carol"})

Added 6 labels, created 6 nodes, set 6 properties, completed after 1 ms.

To test out merge I created another node for mandy

MERGE (:Lecturer {name: "Mandy"})

(no changes, no records)

No records were made as it already exists

Case1: when 2 nodes and the relationship exist, if a MERGE command is used:

First I created a relationship between two existing nodes

MATCH (l:Lecturer {name: "Mandy"})

MATCH (m:Lecturer {name: "James"})

CREATE (m)-[:SUPERVISES]->(l)

RETURN l, m

Created 1 relationship, started streaming 1 records in less than 1 ms and completed after 1 ms.

Then I performed the creation of the already existing relationship using merge

m (l:Lecturer{name:"James"})-[:SUPERVISES]->(m:Lecturer{name:"Mandy"})

RETURN l, m

No changes were made and a duplicate relationship had not been created as you can see below

A screenshot of a computer

Description automatically generated

Case 2: When two nodes exist, but not the relationship, if a MERGE command is used:

The two nodes Lecturer and Student with the names Elizabeth and James already exists as I created it earlier on, however the relationship TEACHES does not exist, when I merged it, it created a relationship between the two nodes as you can see below.

MERGE (l:Lecturer{name:"James"})-[:TEACHES]->(m:Student{name:"Elizabeth"})

RETURN l, m

A screenshot of a computer

Description automatically generated

Case 3: when only one node exists, no relationship exists, if a MERGE command is used:

In this instance only the student node Elizabeth exists out of this query as I created it earlier, however the lecturer node with the name Rashid does not exist and neither does its relationship TEACHES, to Elizabeth either. This should create that non-existent node as well as its relationship as shown below.

MERGE (l:Lecturer{name:"Rashid"})-[:TEACHES]->(m:Student{name:"Elizabeth"})

RETURN l, m

A screenshot of a computer

Description automatically generated

Case 4: When some node exists, and there is a unique constraint on some property of the node, if a MERGE command is used:

Firstly, I created a unique constraint being the name of the lecturers as shown below, then I tried to merge by creating another node and relationship using an existing name “James” to see if it would create but it didn’t, there was an error message as you can see below.

CREATE CONSTRAINT FOR (l:Lecturer) REQUIRE l.name IS UNIQUE

However, I ran into an issue when I matched n returned n I realised when I merged the relationship between the two nodes named James and Mandy, I hadn’t spelt the relationship SUPERVISES correctly so it created another node for James somehow so before I could do this I removed the relationship first by using the query below

MATCH (l:Lecturer{name:"James"})-[t:SUPERVISES]->(p:Lecturer{name:"Mandy"})

DELETE t

Then I deleted the second James node, but due to the fact I had two James nodes I only wanted to delete 1 so I used its id to delete just the one I wanted using the query below.

MATCH (l:Lecturer{name:"James"})

WHERE id(l) = 499

DELETE l

I then proceeded to create the constraint using the unique constraint query above which successfully created then proceeded to merge using James’s name again.

MERGE (l:Lecturer{name:"James"})-[:TEACHES]->(s:Student{name:"Maya"})

RETURN l, s

As predicted an error occurred as James already exists and has a unique constraint of the name property therefore all names for the lecturer node must be unique and not the same, due to this it wasn’t able to create a node for student named Maya as the first part of the query failed to run as shown below:  
 A screenshot of a computer

Description automatically generated

Case 5: two nodes and a relationship in between exist, the relationship has different direction with the one in the merge command:

So previously I have a relationship of James supervising Mandy, but I want to change the direction using the same existing relationship. Now it shows a two-way relationship with the same existing relationship as you can see below

MATCH (l:Lecturer{name:"James"}), (p:Lecturer{name:"Mandy"})

MERGE (p)-[:SUPERVISES]->(l)

RETURN l, p

A screenshot of a computer

Description automatically generated

Case 6: two nodes exist without a relationship, if merge a un-directed relationship with MATCH and MERGE.

Since carol and Lisa have no pre-existing relationship but are existing nodes I used these two nodes in this instance. In the relationship for TEACHES I have not specified a direction; however the direction is shown in the graph below that Carol indeed teaches Lisa. This is because the direction of the relationships when created is left to right by default in neo4j.

MATCH (l:Lecturer{name:"Carol"})

MATCH (m:Student{name:"Lisa"})

MERGE (l)-[:TEACHES]-(m)

RETURN l, m

A screenshot of a computer

Description automatically generated

Case 7: Two nodes exist with a relationship, if merge a un-directed relationship with MATCH and MERGE

So prior I created an undirected relationship with Lisa and Carol, so it’d be best to use this in this instance.

I matched to find the lecturer and student Carol and Lisa and Merged an undirected relationship between the two that already exists. Without returning the variable names this produces a no changes output and given that this relationship exists already it doesn’t matter if the direction is specified, it exists therefore it will not create as shown below.

MATCH (l:Lecturer{name:"Carol"})

MATCH (m:Student{name:"Lisa"})

MERGE (l)-[:TEACHES]-(m)

A screenshot of a computer

Description automatically generated

Case 8: two nodes exist with relationship which has a different direction with the relationship to be merged; if use MATCH and MERGE:

The pre-existing relationship that exists is that Mandy supervises James, however in this instance I changed the direction of the relationship into James supervises Mandy as you can see in the query below, despite the nodes already existing and the relationship already existing too but the other way round, the output was just as the same in case 5 as it is now as it was able to produce a relationship in the direction as shown in the command below

MATCH (l:Lecturer{name:"James"}), (m:Lecturer{name:"Mandy"})

MERGE (l)-[:SUPERVISES]->(m)

RETURN l, m

**A screenshot of a computer

Description automatically generated**

Case 9: when one match fails to match to some nodes before the merge command – when there is only one existing node – merging relationships.

The lecturer node for James already exists whereas the lecturer node for Mohammed does not exist, therefore the results will be no changes and no records as shown below as if one part of the command fails the rest fails. Consequently, no relationship was merged/created

A screenshot of a computer

Description automatically generated

Case 10: if optional match is used, the following code will create a relationship between the two nodes.

First, I matched the lecturer node for James and an optional match for Rashid, meaning if the lecturer node for Rashid does not exist then the query should still return results if not then a null value, but in this instance, it would be an error as the last line of the query is merging a relationship on the two nodes not returning values in the graph database.

MATCH (l:Lecturer{name:"James"})

OPTIONAL MATCH (m:Lecturer{name:"Rashid"})

MERGE (l)-[:SUPERVISES]->(m)

Created 1 relationship, completed after 1 ms.

Case 11: only one node exists; match on the existing node and OPTIONAL match on the non-existing nodes before the merge command:

I matched the existing node for James and optional matched the non-existing node to the lecturer Mark then merged a relationship between the two nodes that resulted in an error as shown below.

MATCH (l:Lecturer{name:"James"})

OPTIONAL MATCH (m:Lecturer{name:"Mark"})

MERGE (l)-[:SUPERVISES]->(m)

A screenshot of a computer

Description automatically generated

Exercise 4:

Create graph nodes without an attribute called ‘marked’ and relationships. Run the FOREACH command to set the value of marked attribute to true.

In this query I matched all relationships from start to finish of a path in a right directional relationship where the starting node begins equals to James and the ending node is equalled to Lisa. For each node, update the node with an additional attribute named marked and set it to true.

In this instance it set 3 properties as there was only 3 nodes in total on that path as shown below

MATCH p=(begin)-[\*]->(end)

WHERE begin.name="James" AND end.name="Lisa"

FOREACH (n IN NODES (p) | SET n.marked = TRUE)

# A screenshot of a computer Description automatically generatedA screenshot of a computer Description automatically generated

Exercise 5:

1. Delete all nodes and relationships:

MATCH (n) DETACH DELETE n

A screenshot of a computer

Description automatically generated

1. Create a unique constraint on module code.

First created my module nodes and assigned to attributes, code and name.

CREATE (m:Module{code:"CSY3024", name:"Database 3"}), (:Module{code:"CSY3025", name:"Artificial Intelligence"}), (:Module{code:"CSY3023", name:"Cybersecurity and Cryptography"})

Then I went ahead and created a unique constraint for code in Module

CREATE CONSTRAINT FOR (m:Module) REQUIRE m.code IS UNIQUE

To double check that the constraint was created I used the schema query

:schema

Which indeed showed that it created as shown below, however when I did this, I also realised the constraint I created earlier on for name was there even though the lecturer node no longer exists in my database so I took it upon myself to remove that constraint from my database so my schema turned from this:

A screenshot of a computer

Description automatically generated

To this using the drop constraint command:

drop constraint constraint\_58bfbac6

A screenshot of a computer

Description automatically generated

1. Create two module nodes with code and title as below: code: “CSY3024”,title:“Databases 3”), (code: “CSY2038”, title: “Databases 2”)

Since I had already created the database 3 node prior, I just created the database 2

CREATE (:Module{code:"CSY2038", name:"Database 2"})

1. Try create another module node with the (code:”CSY3024”, title:”Databases 4) and see what happens

I got an error stating the node already exists which it would be the case as I had added a unique constraint to code in the module node, meaning that all the codes must be unique which avoids duplicates as seen below.

CREATE (:Module{code:"CSY3024", name:"Database 3"})

A screenshot of a computer

Description automatically generated

1. Create a PRE\_REQUISITE relationship between the two module nodes.

First, I matched the two nodes I want to create a relationship with before finally creating the relationship on the create query

MATCH (m:Module{code:"CSY3024"})

MATCH (n:Module{code:"CSY2038"})

CREATE (n)-[:PRE\_REQUISITE]->(m)

1. Suppose module CSY3024 has another pre\_requisite module called (code:”CSY2030”, name:”SYSTEM DESIGN AND DEVELOPMENT), create a pre\_requisite relationship:

First, I matched the CSY3024 node before creating the new node CSY2030, then created the pre\_requisite relationship between them.

MATCH (m:Module{code:"CSY3024"})

CREATE (n:Module{code:"CSY2030", name:"System design and development"})

CREATE (n)-[:PRE\_REQUISITE]->(m)

### Week5: cypher query language 3 – CSV

Exercise 5:

Practise codes from previous file.

Here I created a module node and filled it with the attributes based on my modules.csv file with attributes such as code which is what I named it in neo4j followed by row as I want my data to be viewed in a row format followed by mod\_code which is the actual name of the column in the csv file.

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

CREATE (m:Module{code:row.mod\_code, title:row.title, credit:toInteger(row.credits)})

Added 51 labels, created 51 nodes, set 153 properties, completed after 5 ms.

A screenshot of a computer program

Description automatically generated

Create necessary constraints for the 3 csv files.

First, I created course and award nodes as an addition to my module nodes

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

CREATE (:Course{code:toInteger(row.course\_code), award\_title:row.award\_title, level:row.level})

LOAD CSV WITH HEADERS FROM "file:///award\_maps.csv" AS row

CREATE (:Award{courseCode:toInteger(row.course\_code), modCode:row.mod\_code, requirement:row.requirement})

I attempted to create a unique constraint for the awards node, but an error showed that for mod\_code two cells were the same there it means two properties with the label reward has the same mod\_code attribute. When I looked at the csv file properly, I realised there were two CSY1061 in the mod\_code column.

CREATE CONSTRAINT FOR (a:Award) REQUIRE (a.modCode) IS UNIQUE

Unable to create Constraint( name='constraint\_a8e98564', type='UNIQUENESS', schema=(:Award {modCode}) ):

Both Node(46) and Node(65) have the label `Award` and property `modCode` = 'CSY1020'

However, I was able to make the code in the module node unique

CREATE CONSTRAINT FOR (a:Module) REQUIRE (a.code) IS UNIQUE

Added 1 constraint, completed after 46 ms.

And the same for the courses node:

CREATE CONSTRAINT FOR (a:Course) REQUIRE (a.code) IS UNIQUE

Added 1 constraint, completed after 50 ms.

Create is\_on relationships based on awards\_maps

Firstly I deleted the awards node

match (a:Award)

DELETE a

I then loaded the csv files via awards maps and matched the module nodes and course codes how I created them e.g. in the code for Course, because I created it as an integer I also need to match it as an integer otherwise this will not work, I later realised this and finally was able to create the is\_on relationship on the final line

LOAD CSV WITH HEADERS FROM "file:///award\_maps.csv" AS row

MATCH (m:Module{code:row.mod\_code})

MATCH (n:Course{code:toInteger(row.course\_code)})

CREATE (m)-[:IS\_ON]->(n)

A screenshot of a computer

Description automatically generated

Try to create relationships between the corresponding modules based on the “requirement” values in the award\_maps.csv file.

LOAD CSV WITH HEADERS FROM "file:///award\_maps.csv" AS row

MATCH (c:Course{code:toInteger (row.course\_code)})

MATCH (m:Module{code:row.mod\_code})

WHERE row.requirement = "compulsory"

MERGE (m)-[:COMPULSORY]->(c)

RETURN m, c

Exercise 6:

Run the commands to practise shortest path and allshortestpath functions.

Firstly, I loaded my movie graph as shown below.

A screenshot of a computer

Description automatically generated

Here I am practising the shortest path with some actors:

MATCH (Tom:Person{name:"Tom Tykwer"}), (Penny:Person{name:"Penny Marshall"}), p= shortestPath((Tom)-[\*]-(Penny))

RETURN p

A screenshot of a computer

Description automatically generated

I also practised shortest path with predicates despite not having a relationship between actors such as one actor being related to another to hide those relationships and show the rest, but I still queried even though no changes were made

MATCH (Tom:Person{name:"Tom Tykwer"}), (Penny:Person{name:"Penny Marshall"}), p = shortestPath((Tom)-[\*]-(Penny))

WHERE ALL (r IN relationships(p) WHERE type(r) <> "ACTED\_IN")

RETURN p

Find the shortest path via lengths and change them around to see what appears.

For this query the same 5 nodes and 4 relationships was the output

MATCH (Tom:Person{name:"Tom Tykwer"}), (Penny:Person{name:"Penny Marshall"}), p = shortestPath((Tom)-[\*..5]-(Penny))

RETURN p

A screenshot of a computer

Description automatically generated

I changed the names and changed the number to 15.

MATCH (Ron:Person{name:"Ron Howard"}), (Tom:Person{name:"Tom Tykwer"}), p = shortestPath((Ron)-[\*..15]-(Tom))

RETURN p

A screenshot of a computer

Description automatically generated

Then to 5, strangley the output was the same…

MATCH (Ron:Person{name:"Ron Howard"}), (Tom:Person{name:"Tom Tykwer"}), p = shortestPath((Ron)-[\*..5]-(Tom))

RETURN p

A screenshot of a computer

Description automatically generated

Finding allshortest path between two nodes:

MATCH (Ron:Person{name:"Ron Howard"}), (Tom:Person{name:"Tom Tykwer"}), p = allshortestPaths((Ron)-[\*]-(Tom))

RETURN p

A screenshot of a computer

Description automatically generated

Practise match and optional match:

This query is with 2 existing nodes, and it outputted the nodes as predicted and shown below

MATCH (Ron:Person{name:"Ron Howard"})

OPTIONAL MATCH (Tom:Person{name:"Tom Tykwer"})

RETURN Tom, Ron

A screenshot of a computer

Description automatically generated

I then ran the query again but this time with one existing node and one non-existing node and it displayed only the existing node with no errors as the optional match command allows to return null values rather than values if the node/relationship does not exist.

MATCH (Ron:Person{name:"Ron Howard"})

OPTIONAL MATCH (Elizabeth:Person{name:"Elizabeth"})

RETURN Elizabeth, Ron

A screenshot of a computer

Description automatically generated

### Week 6: Cypher Query language 4

Some return exercises.

Created two nodes with the names Elizabeth and Emmanuel. Created a relationship labelled knows therefore the relationship between Elizabeth is Elizabeth knows Emmanuel. Return p is returning the relationship between the two nodes.

CREATE (n:person{name:"Elizabeth"})

CREATE (m:person{name:"Emmanuel"})

CREATE p=(n)-[:KNOWS]->(m)

RETURN p

Created a new person node named Kenneth and with that I matched person to Elizabeth of which I was able to create a relationship between the two nodes labelled knows and returned the relationship between the 2.

CREATE (n:person{name:"Kenneth"})

WITH n

MATCH (m:person{name:"Elizabeth"})

CREATE p=(n)-[:KNOWS]->(m)

RETURN p

Filter on aggregate function results:A screenshot of a computer

Description automatically generated

At first my hypothesis was that it’d be Kenneth, but I realised that it was looking for more than 2 outgoing relationships of the person that was connected to Elizabeth. So the solution to this is either change the code that it would be any type of relationship (to or from using --) or I create a new node of a person like I did in the above screenshot as an outgoing relationship from Kenneth and then use the 🡪 symbol to show the direction of the relationship is an outgoing relationship from other person that has more than 1 friend of a friend.

Additionally, it didn’t work because Elizabeth is already declared in the variable n, of which Elizabeth cannot be a friend of herself. The result was Kenneth.

MATCH (n:Person{name:"Elizabeth"})--(otherPerson)-->()

WITH otherPerson, COUNT(\*) AS foaf

WHERE foaf > 1

RETURN otherPerson

When I try to run this query without the WITH command, I get a syntax error as you can see below

A screenshot of a computer program

Description automatically generated

Unwind commands:

UNWIND ["A", "B", "C","D"] AS x RETURN x

A screenshot of a computer

Description automatically generated

WITH ["A", "B", "C","D"] AS coll UNWIND coll AS x

WITH DISTINCT x

RETURN collect(x) AS SET

A screenshot of a computer

Description automatically generated

Display all computer science modules running in semester 2:

Since my modules node does not have a semester attribute, I will update my node using the set function, and as you can see the attribute has been successfully added

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

MATCH (m:Module {code: row.mod\_code})

SET m.semester = row.semester

RETURN m

I then proceeded to find every row where semester is equal to S2 using the WITH and WHERE function.

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

WITH row WHERE row.semester = 'S2'

RETURN row

A screenshot of a computer program

Description automatically generated

Display all modules with at least a pre\_requisite

Firstly, I updated my modules node to have the prerequisite column (which I didn’t have to if I use the load csv file query but I still use it as a form of practising)

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

MATCH (m:Module {code: row.mod\_code})

SET m.pre\_requisite = row.pre\_requisite

RETURN m

I then use the WITH clause to pass the rows to the next part of my query whilst filtering my rows to only show rows that are not equal to ‘none’ using my WHERE function then return the result:

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

WITH row WHERE row.pre\_requisite <> 'none'

RETURN row

A screenshot of a computer

Description automatically generated

Which course has the most designated modules:

Firstly, I loaded the award\_maps csv file as row and with that row using the WITH to pass through each row in my csv file, I then proceeded to use the WHERE clause to filter row in the requirement column to equal to designated. With that I converted the course code to integer as that’s how it is stored in the csv file as well as my node. After filtering I proceed to count every row that is included within this filter and store it in my variable designated count. I order the rows based on the designatedCount variable in descending order whilst limiting it to one, to only show one record. I then match course code in courses whilst matching the created variable courseCode from earlier then return the count.

LOAD CSV WITH HEADERS FROM "file:///award\_maps.csv" AS row

WITH row

WHERE row.requirement = 'designated'

WITH toInteger(row.course\_code) AS courseCode, count(\*) AS designatedCount

ORDER BY designatedCount DESC

LIMIT 1

MATCH (c:Course {code: courseCode})

RETURN c, designatedCount

# A screenshot of a computer Description automatically generated

Which module appears the most in courses?

LOAD CSV WITH HEADERS FROM "file:///award\_maps.csv" AS row

WITH row

MATCH (m:Module {code: row.mod\_code})

RETURN m, COUNT(\*) AS appearanceCount

ORDER BY appearanceCount DESC

LIMIT 1

A screenshot of a computer

Description automatically generated