|  |  |  |  |
| --- | --- | --- | --- |
| **CSY3024:Database3** | | | |
| Due for Issue (week commencing): |  | **Last Date for Submission**: | **Sunday 14th January 2017 23:59:59** |
| Agreed Date for late submission: |  | Module Tutor**:** | Dr James Xue |
| Student Name:  Student ID: | Sushil Pun 16401146 |  |  |
| **Assessment Feedback** | | | | |

**Video demo:**

Part I: https://www.youtube.com/watch?v=J7Je2weibV4&feature=youtu.be

Part II: https://www.youtube.com/watch?v=Jce0JsVuU0w&feature=youtu.be

Table of Contents

[PART I 3](#_Toc503733415)

[Introduction: 3](#_Toc503733416)

[Aims: 3](#_Toc503733417)

[Objective: 3](#_Toc503733418)

[Reflection: 3](#_Toc503733419)

[Over all Summaries: 3](#_Toc503733420)

[PART II 5](#_Toc503733421)

[Introduction 5](#_Toc503733422)

[Aim: 5](#_Toc503733423)

[Objective: 5](#_Toc503733424)

[Design: 5](#_Toc503733425)

[Process: 6](#_Toc503733426)

[Query 6](#_Toc503733427)

[APPENDIX: 14](#_Toc503733428)

[Part I 14](#_Toc503733429)

[Exercises 14](#_Toc503733430)

[PART II 57](#_Toc503733431)

# TABLE OF FIGURES

[Figure 1: Total number of matches played. 8](#_Toc503733432)

[Figure 2: Arsenal Fc details. 9](#_Toc503733433)

[Figure 3: Arsenal Fc details 2. 10](#_Toc503733434)

[Figure 4: Number of matches Liverpool lost. 11](#_Toc503733435)

[Figure 5: Number of matches Liverpool lost 2. 11](#_Toc503733436)

[Figure 6: Top five team with best scoring power. 12](#_Toc503733437)

[Figure 7: Team with poorest defending. 13](#_Toc503733438)

[Figure 8: Club name with best winning record. 13](#_Toc503733439)

[Figure 9: Team with maximum number of draws. 15](#_Toc503733440)

[Figure 10: Arsenal Fc consecutive win. 16](#_Toc503733441)

[Figure 1: Clear the database using the DETACH DELETE command. 17](#_Toc503733442)

[Figure 2: Creating two simple nodes with label person and attribute name. 17](#_Toc503733443)

[Figure 3: Creating two simple nodes with label car and attribute name. 17](#_Toc503733444)

[Figure 4: Creating Relationship between person name Mayweather and car name Ferrari. 18](#_Toc503733445)

[Figure 5: Creating a relationship where person name is Diaz and car name is Audia6. 18](#_Toc503733446)

[Figure 6: Deleting Relationship between person Diaz and car Audia6 with nodes also. 19](#_Toc503733447)

[Figure 7: Adding new property to node 20](#_Toc503733448)

[Figure 8: Removed property profession from node. 20](#_Toc503733449)

[Figure 9: Alternative technique to remove property. 21](#_Toc503733450)

[Figure 10: Adding multiple properties at once. 21](#_Toc503733451)

[Figure 11: Removing multiple properties at once. 22](#_Toc503733452)

[Figure 12: Adding properties to relationship. 22](#_Toc503733453)

[Figure 13: Removing properties from relationship. 23](#_Toc503733454)

[Figure 14: Adding new label to existing one. 23](#_Toc503733455)

[Figure 15: Removing label from the node. 24](#_Toc503733456)

[Figure 16: Merge Case I. 25](#_Toc503733457)

[Figure 17: Merge Case II - Creating two nodes and merging relationship. 25](#_Toc503733458)

[Figure 18: Deleting nodes and it's relationship. 26](#_Toc503733459)

[Figure 19: Creating a node without relationship. 26](#_Toc503733460)

[Figure 20: Merge Case III. 27](#_Toc503733461)

[Figure 21: Deleting First the multiple person name Connor. 27](#_Toc503733462)

[Figure 22: Creating Unique Constraint on person name attribute. 28](#_Toc503733463)

[Figure 23: Merge Case IV - UNIQUE CONSTRAINT 28](#_Toc503733464)

[Figure 24: Removing Unique Constraint From name property of person label. 29](#_Toc503733465)

[Figure 25: Creating node and unidirectional relationship between them. 29](#_Toc503733466)

[Figure 26: Merge Case V - Unidirectional Relationship. 30](#_Toc503733467)

[Figure 27: Deleting Nodes and relationship. 30](#_Toc503733468)

[Figure 28: Deleting Relationship. 31](#_Toc503733469)

[Figure 29: Merge Case VI Un-directed relationship. 31](#_Toc503733470)

[Figure 30: Merge Case VII. 31](#_Toc503733471)

[Figure 31: Merge Case VIII. No duplication of nodes and relationship. 32](#_Toc503733472)

[Figure 32: Creating a node. 33](#_Toc503733473)

[Figure 33: Match node and create unique relationship between them. 33](#_Toc503733474)

[Figure 34: Two nodes Created. 34](#_Toc503733475)

[Figure 35: Unique Relationship 1. 34](#_Toc503733476)

[Figure 36: Unique Relationship 2. 35](#_Toc503733477)

[Figure 37: Creating property value for each nodes of same label. 35](#_Toc503733478)

[Figure 38: All nodes with property marked value true. 36](#_Toc503733479)

[Figure 39: Loading a csv file with-out headers. 37](#_Toc503733480)

[Figure 40: Loading a csv file with headers. 38](#_Toc503733481)

[Figure 41: loading a csv file with field delimiter. 39](#_Toc503733482)

[Figure 42: merging pre-requiste relationship between modules. 40](#_Toc503733483)

[Figure 43: merging co-requiste relationship between modules. 41](#_Toc503733484)

[Figure 44: loading pathway csv file. 42](#_Toc503733485)

[Figure 45: merging relationship between two nodes of different labels. 43](#_Toc503733486)

[Figure 46: shortest path between tom and laurence. 45](#_Toc503733487)

[Figure 47: displaying all the shortest path between tom and laurence. 46](#_Toc503733488)

[Figure 48: shortest path between 5 relationships. 47](#_Toc503733489)

[Figure 49: displaying the shortest path without none clause. 47](#_Toc503733490)

[Figure 50: match return no change. 48](#_Toc503733491)

[Figure 51: Optional match. 48](#_Toc503733492)

[Figure 52: nodes with relationship. 49](#_Toc503733493)

[Figure 53: Boolean operation. 50](#_Toc503733494)

[Figure 54: filtering on node label. 50](#_Toc503733495)

[Figure 55: where clause on node property. 51](#_Toc503733496)

[Figure 56: where clause on relationship property. 51](#_Toc503733497)

[Figure 57: where clause on property exists. 52](#_Toc503733498)

[Figure 58: where clause on starting of the string. 52](#_Toc503733499)

[Figure 59: where clause to match the end of string. 53](#_Toc503733500)

[Figure 60: where clause to match anywhere in the string. 53](#_Toc503733501)

[Figure 61: filtering where string does not ends with p. 54](#_Toc503733502)

[Figure 62: filtering with the regular expression. 54](#_Toc503733503)

[Figure 63: CASE INSENSITIVE REGULAR EXPRESSION. 55](#_Toc503733504)

[Figure 64: using path patter in where clause. 56](#_Toc503733505)

[Figure 65: filtering patters using not clause on where 56](#_Toc503733506)

[Figure 66: filtering on patter with properties. 57](#_Toc503733507)

[Figure 67: filter on relationship type. 58](#_Toc503733508)

[Figure 68: filtering with the list in where clause. 58](#_Toc503733509)

[Figure 69: filtering with the matching property. 59](#_Toc503733510)

# PART I

## Introduction:

The assignment is divided into two parts.

Part I is about the learning of the exercise done and documenting the learning path. The exercises are provided on a nile weekly basis.

## Aims:

The aim of this part is to complete exercises provided on nile and writing the learning activities. There are six exercise on cypher query language which needs to be done with documenting what is been learned.

## Objective:

* To write reflection of each exercises completed.
* To write an overall reflection of part 1 assignment.
* To describe the method used on query.
* Reason behind the query used implemented.

## Reflection:

|  |  |  |
| --- | --- | --- |
| Exercise | Attempted | Completed |
| Cypher 1 | Attempted | Completed |
| Cypher 2 | Attempted | Completed |
| Cypher 3 | Attempted | Completed |
| Cypher 4 | Attempted | Completed |
| Cypher 5 | Attempted | Completed |
| Cypher 6 | Attempted | Completed |

## Over all Summaries:

Graph database store data in linked-list manner where these data are connected. Information is stored in nodes and relationship. Relationship is the relation between the nodes. Both nodes and relationship have properties or attributes. Exercise cypher 1 was all about creating node and making relationship between them and being able to delete those nodes and relationship. Completing this exercise author get familiar with nodes, relationship, label and properties. The label can be added to both nodes and relationship and one thing author get from this exercise was command “DETACH DELETE” what it does is it deletes all the node of the database.

Through exercise cypher 2 author finds that there are many possible way to add properties to a nodes or relationship. Adding properties can be done single at a time or multiple properties at once. One important thing author learned from this exercise was setting properties value to null will remove the property from nodes or relationship. Learned way to remove label from nodes was a pretty simple.

On cypher 3, not tricky but the interesting topic till now. On this topic there are 8 technique of implementing merge clause. Merge is a very powerful command; it can create if not created before. If the nodes and relationship does not exist it will create node or relationship, or if nodes and relationship does exist then it will simple merge them without creating duplication. Programmer needs to be very precise while using merge case because in some case it can create duplication and also one thing to care about is using the direction of the relationship while using merge clause.

On cypher 4, there is one clause create unique which is basically match what it can and create what is missing. Now a day this clause is deprecated. Merge can do what create unique clause can, so merge and create unique are somehow identical both is used to prevent duplication but create unique is used to prevent relationship duplication whereas merge is generally used to prevent nodes duplication. In this exercise author gets to know about the foreach clause what it does is, that it creates the same property value pair for each node linked through same relationship.

On cypher 5, author gets to know the technique to load a file to a neo4j database. Csv file are comma separated value. These comma separated file can have headers on it or without header and also can be with field delimiter. All three type of this csv file can be loaded to a database with a certain command for each one. Not only csv file can be loaded but a number of different file can be loaded such as json file and also from Google sheet. Author find easiest way to load a dataset is from a csv file. If the file is not on csv format there are different tools available to convert from to a csv file.

On this same exercise there is available a technique to find a shortest way possible of finding the relationship between the two nodes. There are many two ways to find shortest way possible one method is shortestPath and another one is allShortestPath. Nodes are the parameter taken by these methods. Another imported lesson learned from this exercise is about optional match. Optional match are like match clause but it will return null for the missing parts.

On cypher 6, author gets to know that there are tons of ways that can be used to get the required result. Where clause can be used in nodes as well as labels and also relationship. To match a string only there are various method available for example to match the starting string there is STARTS WITH used in where clause to return the string match with the provided starting string likewise there is End With to match the ending of the string with the provided string in the query.

And there is one simple function CONTAINS which match the string that can be anywhere. Regular expression is also used to match the string in where clause.

# PART II

## Introduction

This part of the assignment is to develop a graph database for a provided dataset. The information contains on the dataset is about the English premier league matches.

## Aim:

The aim of the project is to design and create a graph database.

## Objective:

The list of the objective to develop a graph database is listed below:

* The number of matches been played.
* Details of all the matches that “Arsenal Fc” has involved.
* The number of matches Liverpool has lost.
* Displaying the top five teams that have best scoring record.
* Displaying the top five teams that have poorest defending record.
* Displaying the top five teams that have best winning record.
* Displaying the top five teams that have best home winning record.
* Displaying the top five teams that have best home losing record.
* The team with most draws.
* Team that has most consecutive wins

## Design:

Author decides to store the list of club that are used in dataset to store in a club.csv file for the sake of easy. It is very tedious and time consuming to create one club node at a time so all the club is placed in file which has a header club with 20 clubs on it.

After 20 clubs are stored in database they are made relationship on the basis of home and away club which is already given in the available dataset. Author decided to name played\_with relationship to build between the nodes. After nodes and relationship are created author decided to put attributes or properties on it.

## Process:

-- Loading club.csv file

LOAD CSV WITH HEADERS FROM ("file:///club.csv") AS row

create(n:Club {name:'row.club'};

--Loading result.csv file and making relationship between home and away club

LOAD CSV WITH HEADERS FROM ("file:///result.csv") AS row

match(a:Club{name:row.home}),(b:Club{name:row.away}) Merge (a)-[r:played\_with]->(b);

--Loading attributes to relation

LOAD CSV WITH HEADERS FROM ("file:///result.csv") AS row

match(a:Club{name:row.home})-[r:played\_with]-(b:Club{name:row.away})

set r+={status:row.status,home:row.home, away:row.away,score:[row.score\_\_001, row.score\_\_002], time:row.time, id:row.id, link:row.link };

## Query

1. **Display total number of matches played.**

**MATCH (a:Club)-[r:played\_with{status:'played'}]->(b:Club) return count(r) AS Total\_MATCH\_Played;**

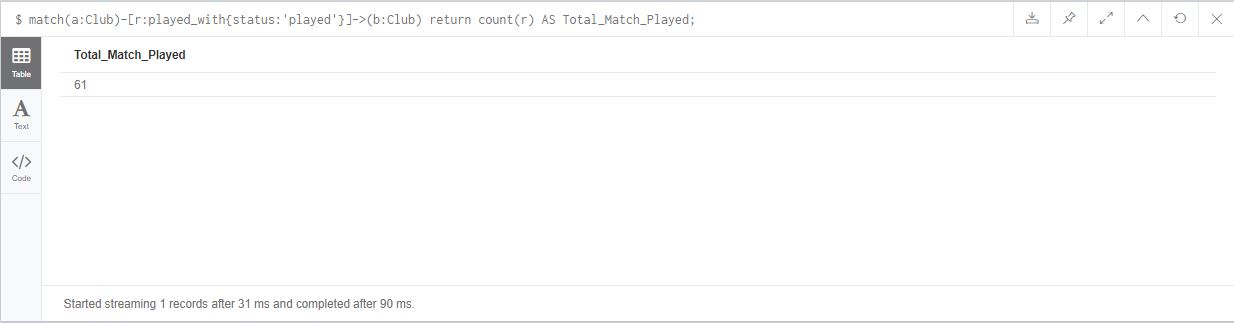
****

Figure : Total number of matches played.

**2) Display details of all matches involved “Arsenal FC”.**

**MATCH (a:Club{name:'Arsenal FC'})-[r:played\_with{status:'played'}]-(b:Club) return r AS Arsenal\_FC\_MATCHES;**

****

Figure : Arsenal Fc details.



Figure : Arsenal Fc details 2.

**3) Display the number of matches “Liverpool FC” has lost.**

**-- Technique One:**

**MATCH (a:Club{name:'Liverpool FC'})**

**OPTIONAL MATCH (a)-[r:played\_with]->(:Club)**

**WHERE r.score[0]<r.score[1]**

**WITH a, count(r) as result1**

**OPTIONAL MATCH (a)<-[r:played\_with]-()**

**WHERE r.score[0]>r.score[1]**

**WITH a, result1, count(r) as result2**

**RETURN result1 + result2 as Total\_MATCH\_Liverpool\_Lost;**

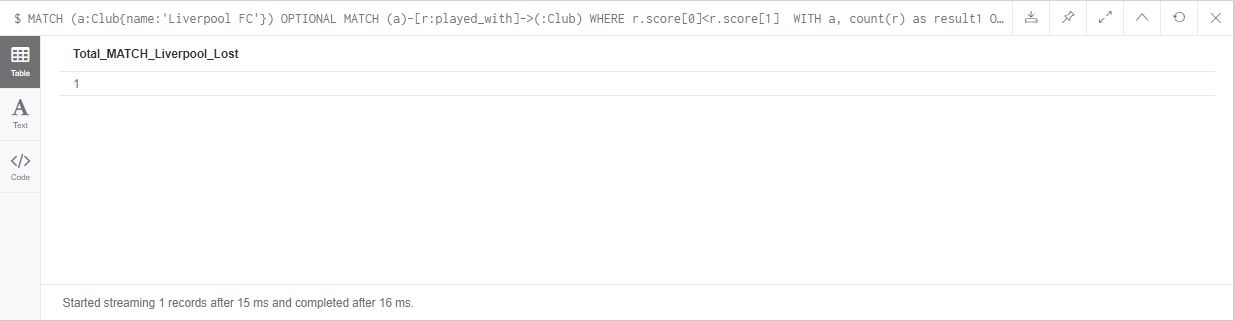
****

Figure : Number of matches Liverpool lost.

**-- Technique Two:**

**MATCH(a:Club{name:'Liverpool FC'})-[r:played\_with]->(b:Club)**

**WHERE r.score[0]<r.score[1] return r as result**

**UNION**

**MATCH(a:Club)-[r:played\_with]->(b:Club{name:'Liverpool FC'})**

**WHERE r.score[0]>r.score[1] return r as result;**

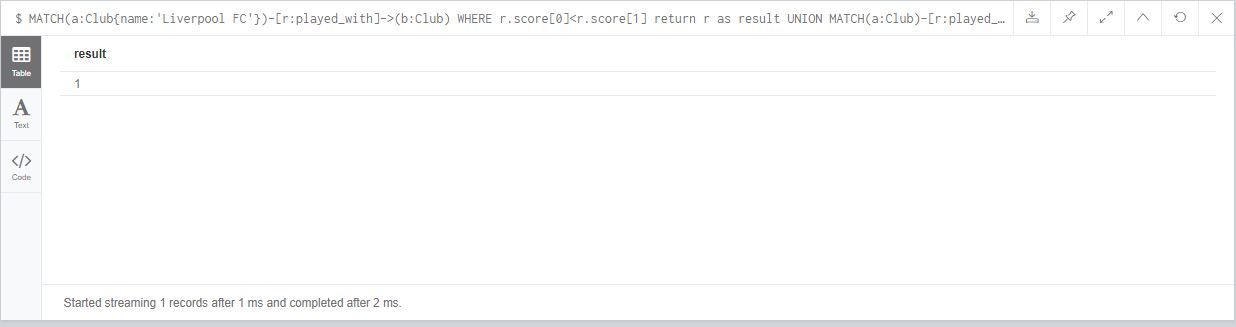


Figure : Number of matches Liverpool lost 2.

**4) Display top five teams that have best scoring power.**

**---- Club Home score**

**MATCH(a:Club)-[r:played\_with]->(b:Club)**

**return a.name AS Club, sum(toInteger(head(r.score))) AS Home\_Score;**

**-- Club Away score**

**MATCH(a:Club)<-[r:played\_with]-(b:Club)**

**return a.name AS Club, sum(toInteger(last(r.score))) AS Away\_Score;**

**--Finally adding the home score and away score.**

**LOAD CSV WITH HEADERS FROM ("file:///club.csv") AS row**

**MATCH(a:Club{name:row.club})-[r:played\_with]->(b:Club)**

**WITH a,sum(toInteger(head(r.score))) AS Home\_score**

**OPTIONAL MATCH (a)<-[r:played\_with]-(b:Club)**

**WITH a,Home\_score, sum(toInteger(last(r.score))) AS Away\_score**

**RETURN a.name AS CLUB, Home\_score+Away\_score AS Total\_Score**

**ORDER BY Total\_Score DESC LIMIT 5;**

****

Figure : Top five team with best scoring power.

**5) Display top five teams that have poorest defending.**

**LOAD CSV WITH HEADERS FROM ("file:///club.csv") AS row**

**MATCH(a:Club{name:row.club})-[r:played\_with]->(b:Club)**

**WITH a,sum(toInteger(last(r.score))) AS Home\_concede**

**OPTIONAL MATCH (a)<-[r:played\_with]-(b:Club)**

**WITH a,Home\_concede, sum(toInteger(head(r.score))) AS Away\_concede**

**RETURN a.name, Home\_concede+Away\_concede AS Total\_concede**

**ORDER BY Total\_concede DESC LIMIT 5;**

****

Figure : Team with poorest defending.

**6) Display top five teams that have best winning records.**

**LOAD CSV WITH HEADERS FROM ("file:///club.csv") AS row**

**MATCH(a:Club{name:row.club})-[r:played\_with]->(b:Club)**

**WHERE r.score[0]>r.score[1]**

**WITH a, count(r) as result1**

**MATCH(a:Club)<-[r:played\_with]-(b:Club)**

**WHERE r.score[0]<r.score[1]**

**WITH a,result1, count(r) as result2**

**return a.name AS CLUB, result1+result2 AS WINNING\_RECORDS**

**ORDER BY WINNING\_RECORDS DESC LIMIT 5;**

****

Figure : Club name with best winning record.

**7) Display top five teams with best home winning records.**

**LOAD CSV WITH HEADERS FROM ("file:///club.csv") AS row**

**MATCH(a:Club{name:row.club})-[r:played\_with]->(b:Club)**

**WHERE r.score[0]>r.score[1]**

**RETURN a.name, count(r) AS HOME\_WINNIG\_RECORDS**

**ORDER BY HOME\_WINNIG\_RECORDS DESC LIMIT 5;**

****

**8) Display top five teams with worst home losing recording.**

**LOAD CSV WITH HEADERS FROM ("file:///club.csv") AS row**

**MATCH(a:Club{name:row.club})-[r:played\_with]->(b:Club)**

**WHERE r.score[0]<r.score[1]**

**RETURN a.name, count(r) AS WORST\_HOME\_RECORD**

**ORDER BY WORST\_HOME\_RECORD DESC LIMIT 5;**



**9) Which teams had most “draw”?**

**LOAD CSV WITH HEADERS FROM ("file:///club.csv") AS row**

**MATCH(a:Club{name:row.club})-[r:played\_with]-(b:Club)**

**WHERE r.score[0]=r.score[1]**

**return a.name AS CLUB, count(r) AS DRAWS**

**ORDER BY DRAWS DESC LIMIT 7;**

****

Figure : Team with maximum number of draws.

**10) Display the team with most consecutive “wins”.**

**-- To find the consecutive win of Arsenal FC**

**MATCH (a:Club {name:'Arsenal FC'})-[r:played\_with]-(:Club)**

**WITH ((CASE a.name WHEN r.home THEN 1 ELSE -1 END) \* (TOINT(r.score[0]) - TOINT(r.score[1]))) > 0 AS win, r**

**ORDER BY TOINT(r.time)**

**RETURN REDUCE(s = {max: 0, curr: 0}, w IN COLLECT(win) |**

**CASE WHEN w**

**THEN {**

**max: CASE WHEN s.max < s.curr + 1 THEN s.curr + 1 ELSE s.max END,**

**curr: s.curr + 1}**

**ELSE {max: s.max, curr: 0}**

**END**

**).max AS result;**

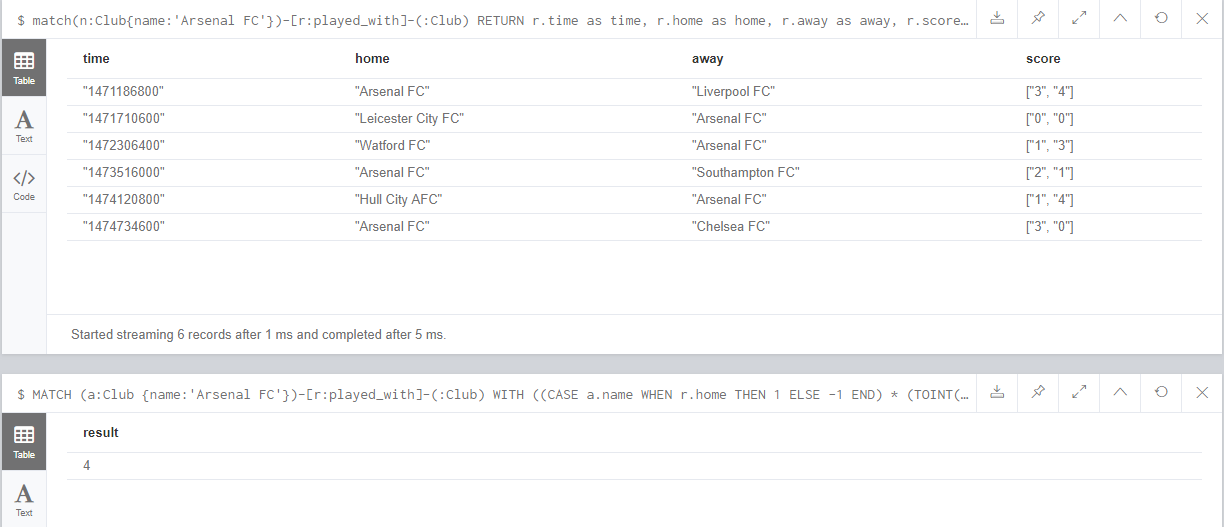


Figure 10: Arsenal Fc consecutive win.

# APPENDIX:

## Part I

### Exercises

#### Cypher 1:

• Start the Neo4j database.

• Clear the database using the DETACH DELETE command.

• Create simple graph nodes and relationships.

• Delete some nodes and relationships.

• Log your learning activities and reflection in your learning diary.

*-- Clear the database using the DETACH DELETE command*

**MATCH (n) DETACH DELETE n;**

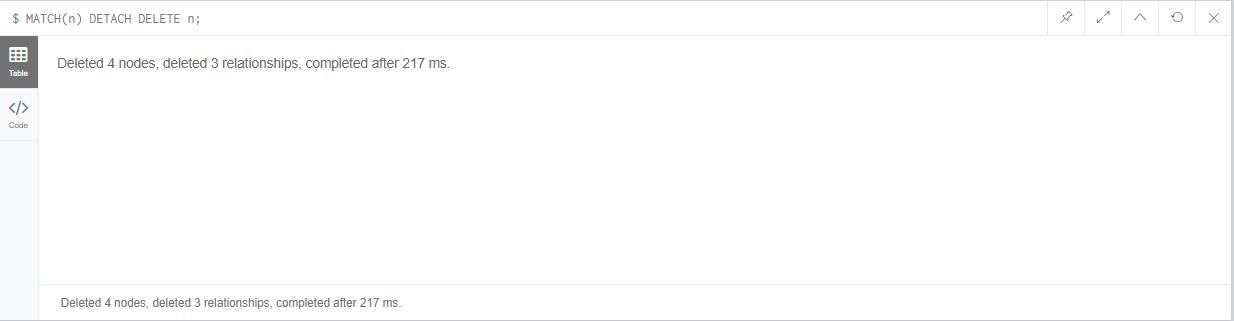
****

Figure 1: Clear the database using the DETACH DELETE command.

*-- Creating two simple nodes with label person and attribute name*

**CREATE (a:person{name:'Mayweather'}), (b:person{name:'Diaz'});**



Figure 2: Creating two simple nodes with label person and attribute name.

*-- Creating two simple nodes with label car and attribute name*

**CREATE (a:car{name:'Ferrari'}), (b:car{name:'Audia6'})**

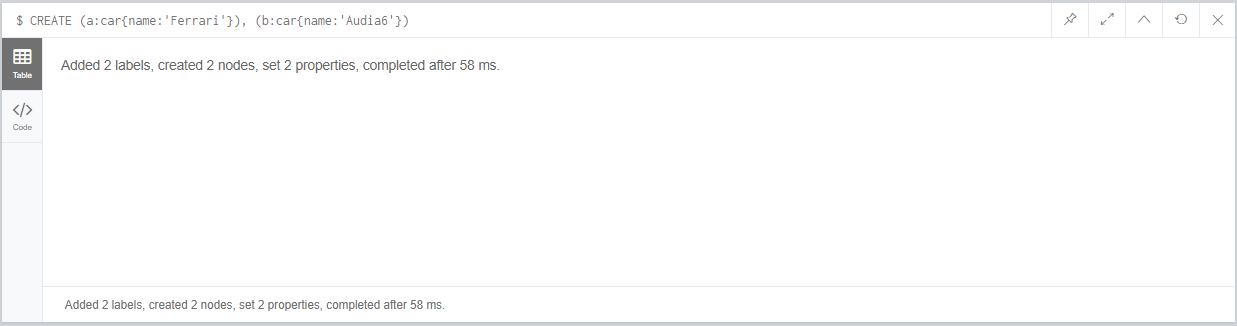


Figure 3: Creating two simple nodes with label car and attribute name.

*--creating relationship between them*

**MATCH (a:person{name:'Mayweather'}),(b:car{name:'Ferrari'}) MERGE (a)-[o:owns]->(b);**

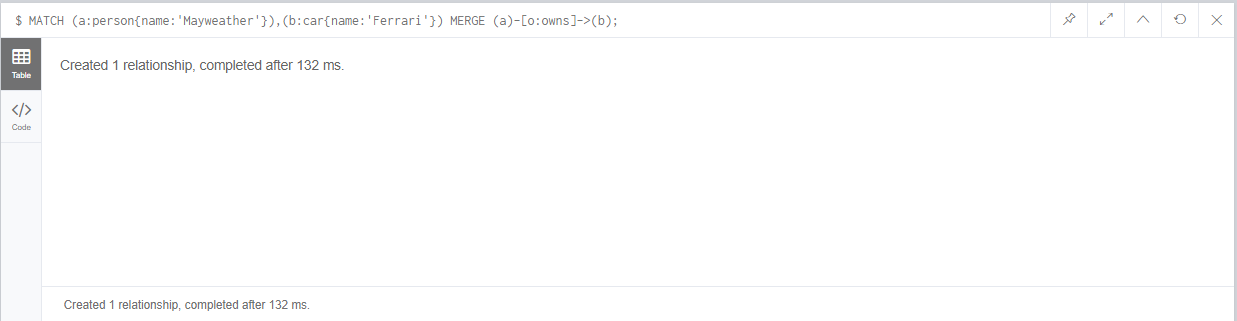


Figure 4: Creating Relationship between person name Mayweather and car name Ferrari.

*-- creating a relationship where person name is Diaz and car name is Audia6*

**MATCH (a:person),(b:car)**

**WHERE a.name='Diaz' AND b.name = 'Audia6'**

**CREATE (a)-[o:owns]->(b);**



Figure 5: Creating a relationship where person name is Diaz and car name is Audia6.

*-- Delete some nodes and relationships*

**MATCH(a:person{name:'Diaz'})-[r:owns]->(b:car{name:'Audia6'}) DELETE r,a,b;**



Figure 6: Deleting Relationship between person Diaz and car Audia6 with nodes also.

/\* LOGS \*/

*-- Creating nodes and relationship was fun*

*-- 1. Simple nodes, nodes with label and properties can be created as wish.*

*-- 2. Relationship is the relation between the nodes. It can have label as well as attribute. Relationship can be created with different technique above shown are two of the technique.*

*-- 3. Deleting nodes and relationship:*

*-- 3.a. Before deleting nodes relationship must be deleted first*

*-- 4. DETACH DELETE Command deletes all the nodes and relationship of database*

#### Cypher 2:

• Start the Neo4j database

• Using the reference codes in the early slides and practice how to add, remove properties in nodes/relationships and add and remove labels.

• Log your learning activities and reflection in the diary.

*-- NODE WITH PERSON LABEL AND NAME ATTRIBUT ALREADY EXITS, NOW ADDING ANOTHER PROPERTIES, PROFESSION TO NODE*

MATCH (a:person{name:'Mayweather'})

SET a.profession = 'Boxer'

RETURN a;

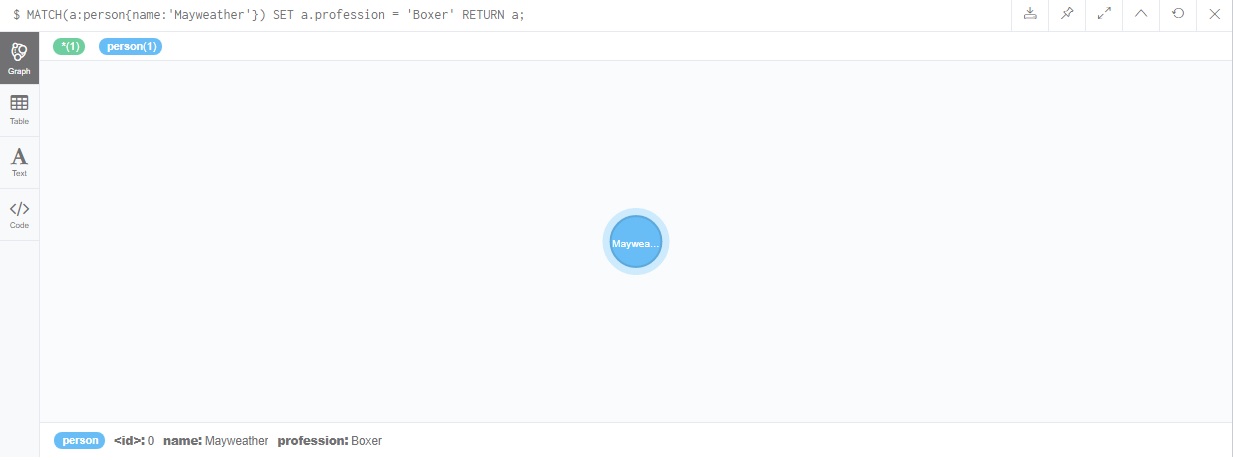


Figure 7: Adding new property to node

*-- REMOVING PROPERTIES FROM NODE*

MATCH (a:person{name:'Mayweather'})

SET a.profession = NULL

RETURN a;

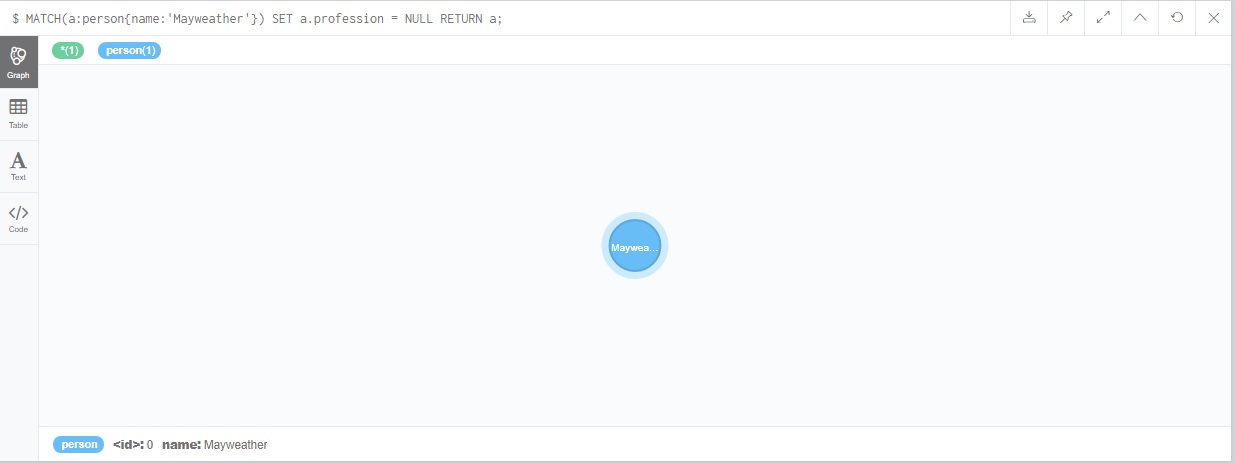


Figure 8: Removed property profession from node.

*-- ANOTHER TECHNIQUE TO REMOVE PROPERTIES FROM NODE*

MATCH (a:person{name:'Mayweather'})

REMOVE a.profession

RETURN a;

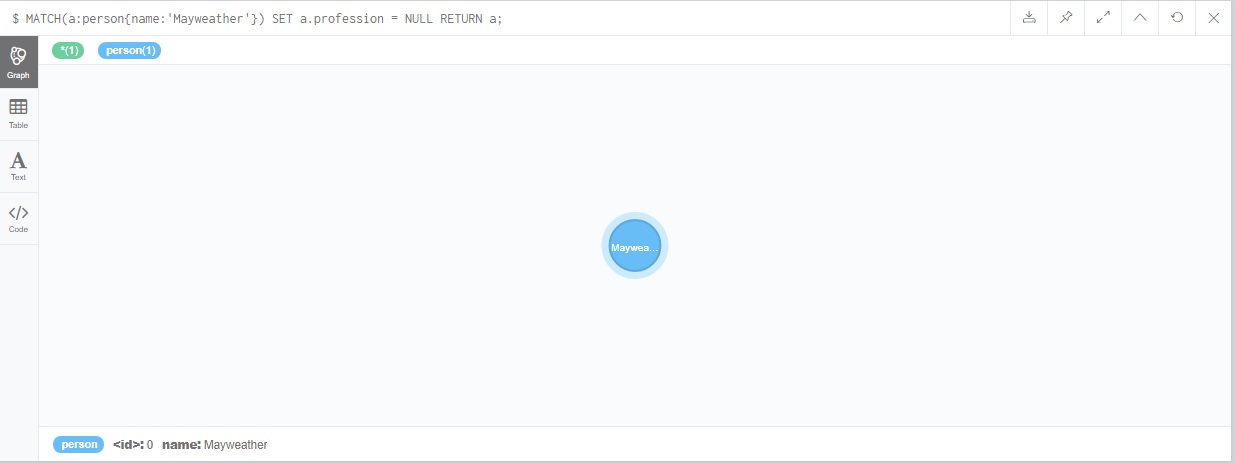


Figure 9: Alternative technique to remove property.

*-- Adding multiple properties at once to node*

MATCH (a:person{name:'Mayweather'})

SET a += { profession: 'Boxer', country:'United States'}

RETURN a;

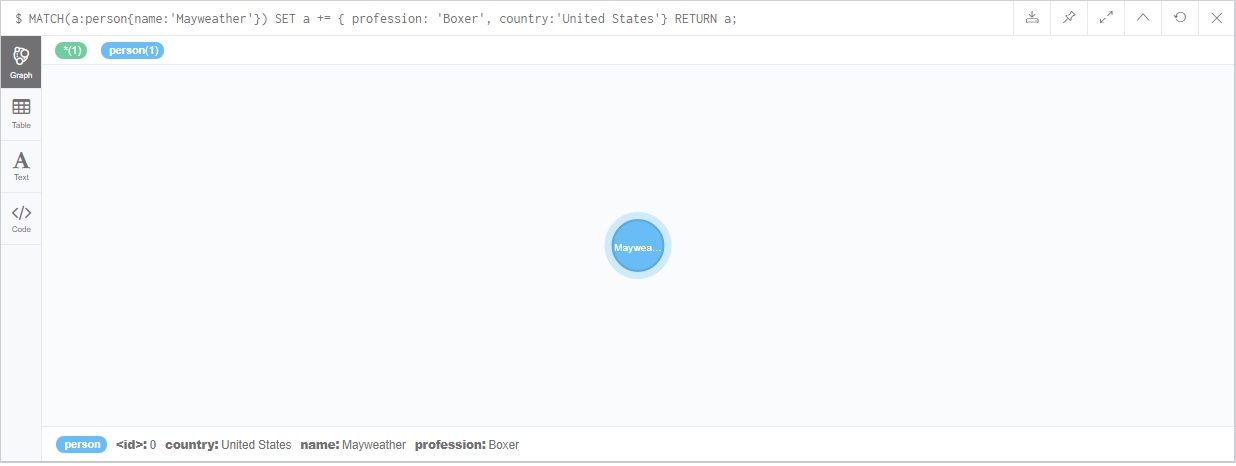


Figure 10: Adding multiple properties at once.

*--Removing multiple properties at once from node*

MATCH (a:person{name:'Mayweather'})

REMOVE a.profession, a.country

RETURN a;

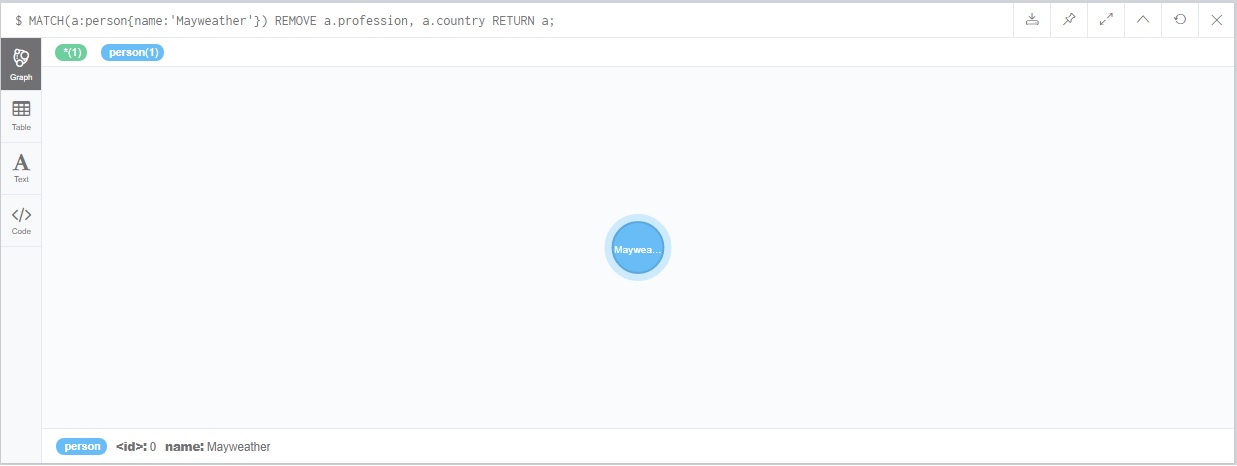


Figure 11: Removing multiple properties at once.

*-- ADDING PROPERTIES TO RELATIONSHIP*

MATCH (a:person{name:'Mayweather'})-[r:owns]->(b:car{name:'Ferrari'})

SET r.since = 2002

RETURN a,b,r;

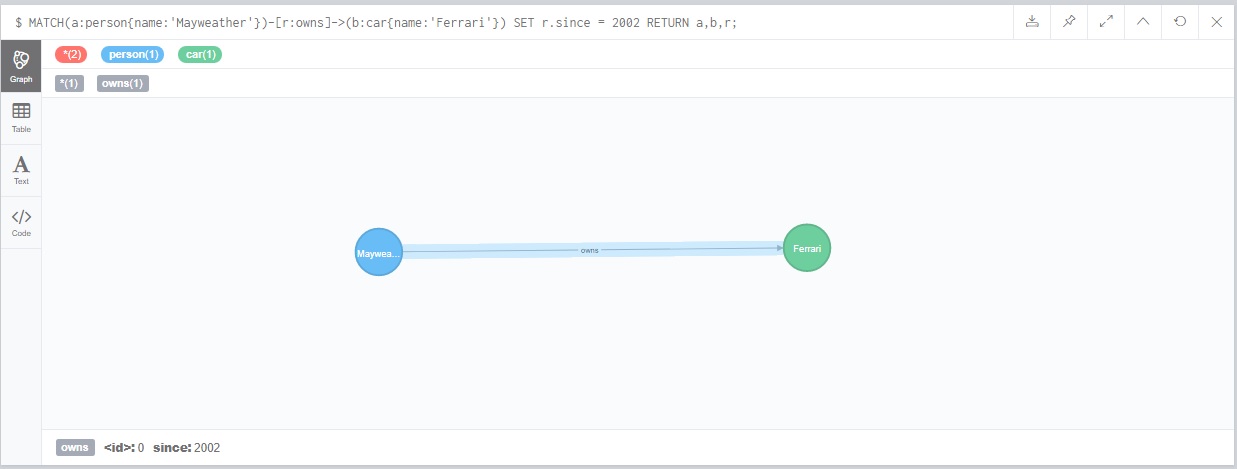


Figure 12: Adding properties to relationship.

*-- REMOVE PROPERTIES FROM RELATIONSHIP*

MATCH (a:person{name:'Mayweather'})-[r:owns]->(b:car{name:'Ferrari'})

SET r.since = NULL

RETURN a,b,r;

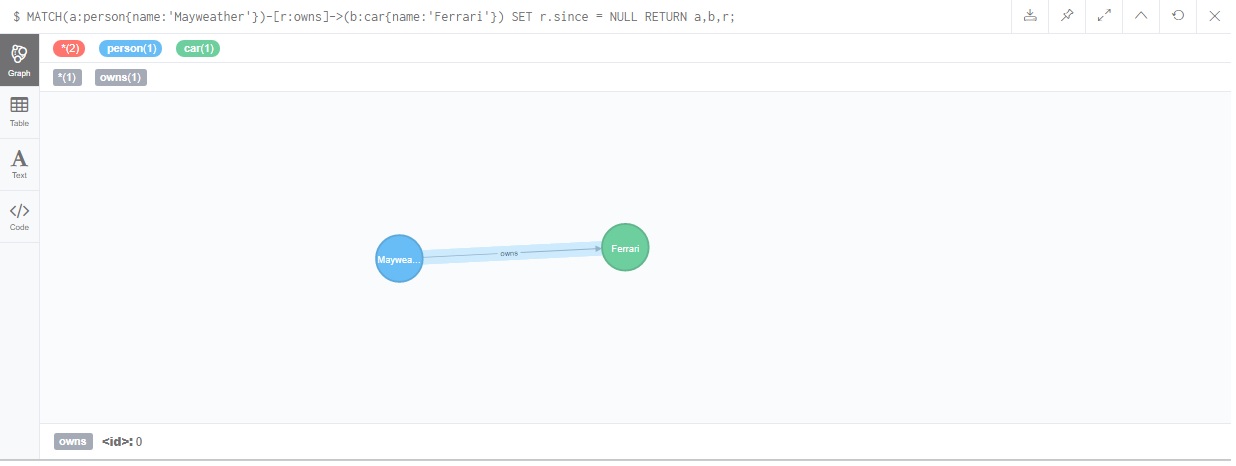


Figure 13: Removing properties from relationship.

*-- ADDING NEW LABEL TO, NODES*

MATCH (a:person{name:'Mayweather'})

SET a:Coach

RETURN a ;

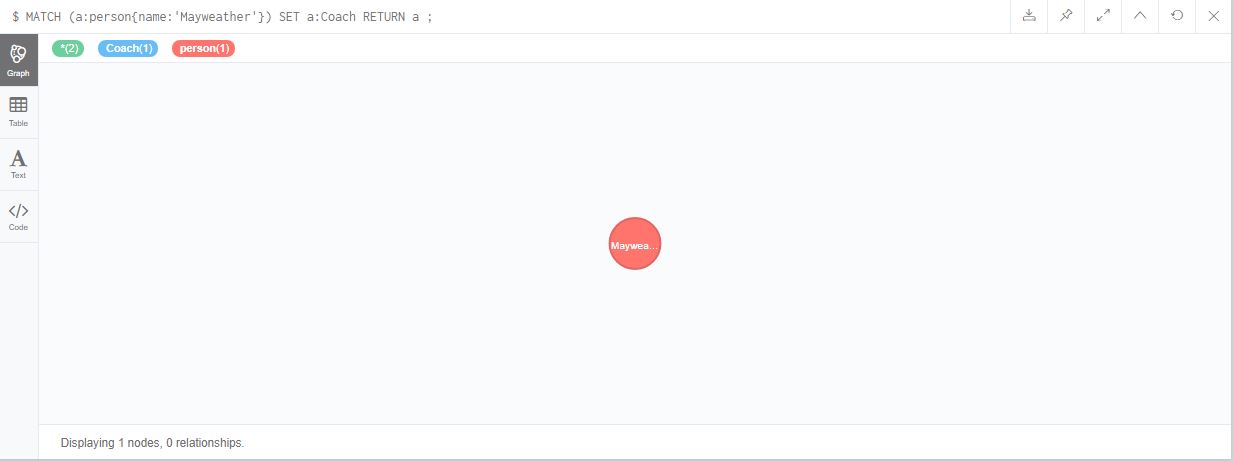


Figure 14: Adding new label to existing one.

*-- REMOVING LABEL FROM EXISTING NODES*

MATCH (a:person{name:'Mayweather'})

REMOVE a:Coach

RETURN a;

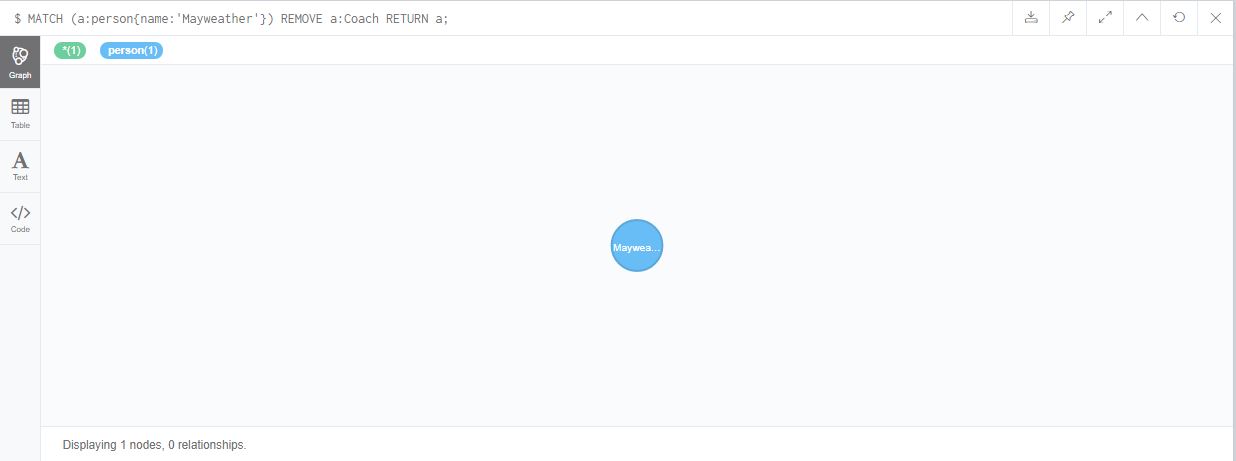


Figure 15: Removing label from the node.

/\* --- LOG --- \*/

Author finds that adding properties to nodes or relationship is possible in various ways

Adding an attributes or properties can be done single at a time or many properties at once.

Author gets to know that setting properties to null will remove a property from node/relation

Adding and removing a label is pretty simple.

#### Cypher 3:

• Start Neo4j database.

• Create some nodes and relationships based on the diagram on the previous slides (or a similar diagram of your choice).

• Practice the MERGE clause for all the eight different cases listed on the previous slides.

• Log your learning activities and reflection in your diary.

*-- Case I:*

MERGE (p:person{name:'Mayweather'})-[o:owns]->(c:car{name:'Ferrari'});

*-- outputs no changes because person name 'Mayweather' already owns a car name 'Ferrari'*



Figure 16: Merge Case I.

*-- Case II:*

*-- Creating two nodes (person, car) and relationship between them.*

MERGE (p:person{name:'Connor'})-[o:owns]->(c:car{name:'Lamborgini'});

*-- outputs: Added 2 labels, created 2 nodes, set 2 properties, created 1 relationship, completed after 15 ms.*

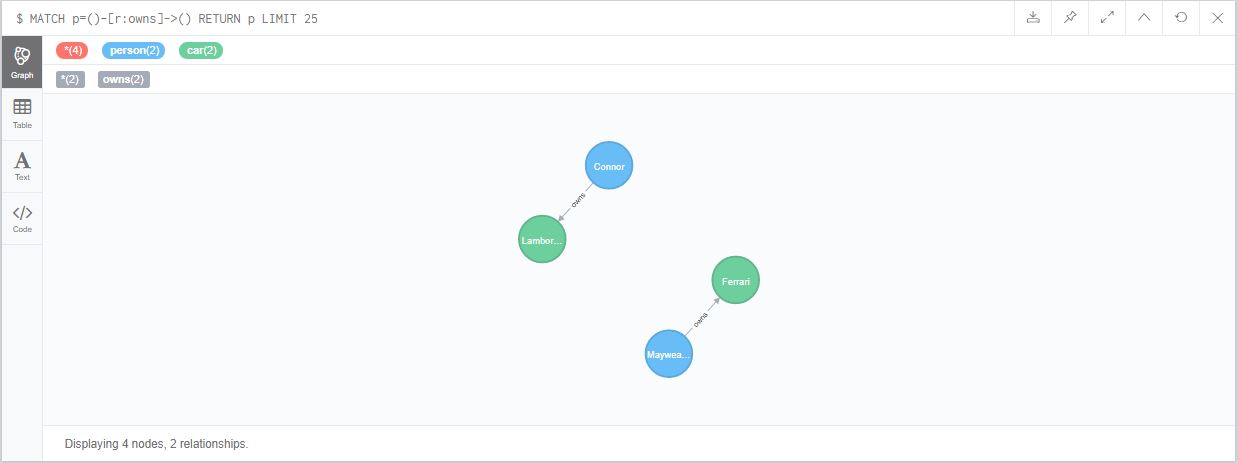


Figure 17: Merge Case II - Creating two nodes and merging relationship.

*-- Case III:*

*-- for case 3 there must be one already exit node with no relationship so deleting previously created nodes and relationship.*

MATCH (p:person{name:'Connor'})-[o:owns]->(c:car{name:'Lamborgini'}) delete o,p,c;



Figure 18: Deleting nodes and it's relationship.

Create(p:person{name:'Connor'});

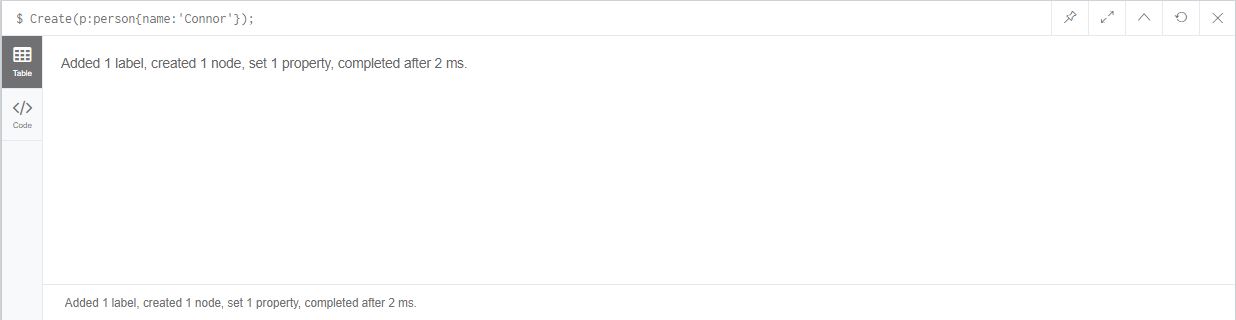


Figure 19: Creating a node without relationship.

MERGE (p:person{name:'Connor'})-[f:fightsIN]->(s:sports{name:'MMA', division:'Lightweight'});

*-- This command matches node p with label person and creates node s with label sports with attributes on it and makes a relationship such that p fights in s.*

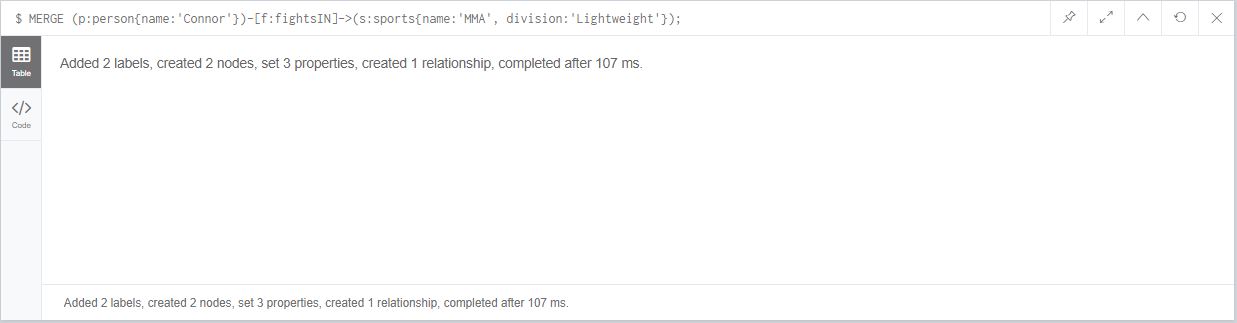


Figure 20: Merge Case III.

*--CASE IV:*

*when some node exists , and there is a unique constraint on some property of the node; use MERGE only*

*MATCH (p:person{name:'Connor'})-[f:fightsIN]->(s:sports{name:'MMA', division:'Lightweight'}) DELETE f,p,s;*

**

Figure 21: Deleting First the multiple person name Connor.

*-- Creating constraint on name property of person node*

*CREATE CONSTRAINT ON (p:person) ASSERT p.name IS UNIQUE;*

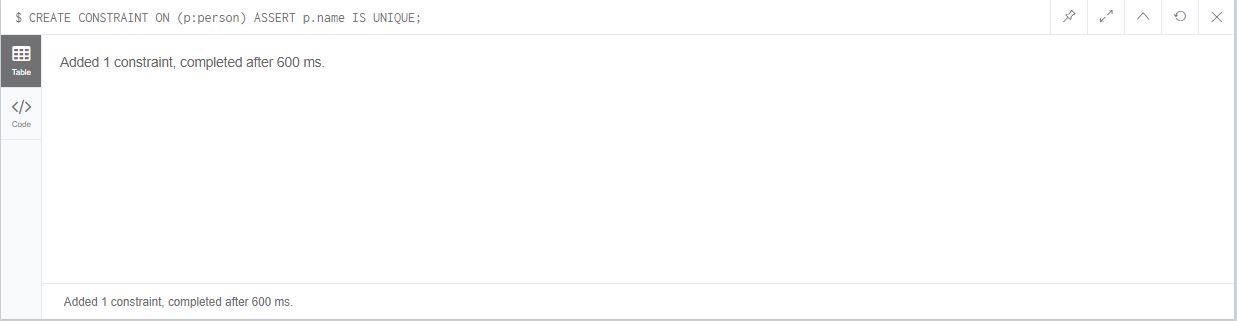
**

Figure 22: Creating Unique Constraint on person name attribute.

*Node with label person and property Mayweather already exist now trying to merge it.*

*MERGE (p:person{name:'Mayweather'})-[:owns]->(c:car{name:'Audi'});*

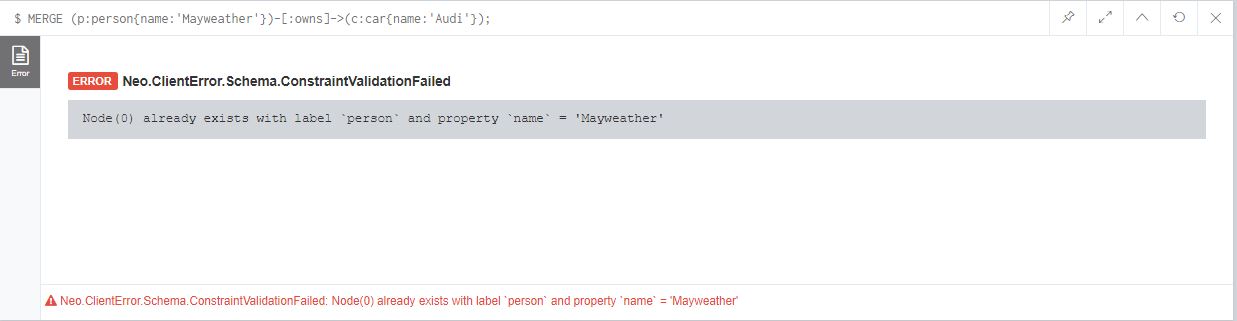
**

Figure 23: Merge Case IV - UNIQUE CONSTRAINT

*-- Outputs: Already exists with label `person` and property `name` = 'Mayweather'*

-- Case V:

*--Two nodes and a relationship between them exists, with a direction Ronaldo envy messi*

*-- Person name has constraint it must be unique so first drop constraint*

DROP CONSTRAINT ON (p:person) ASSERT p.name IS UNIQUE;

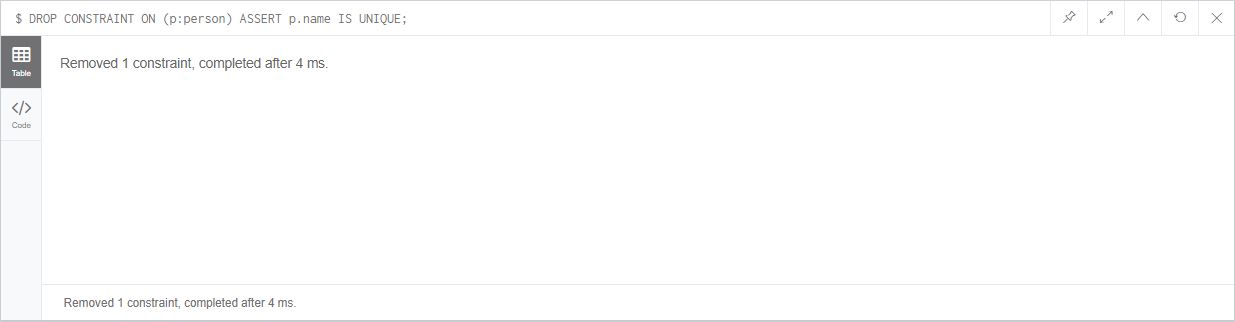


Figure 24: Removing Unique Constraint From name property of person label.

MERGE(:person{name:'RONALDO'})-[e:envy]->(:person{name:'MESSI'});

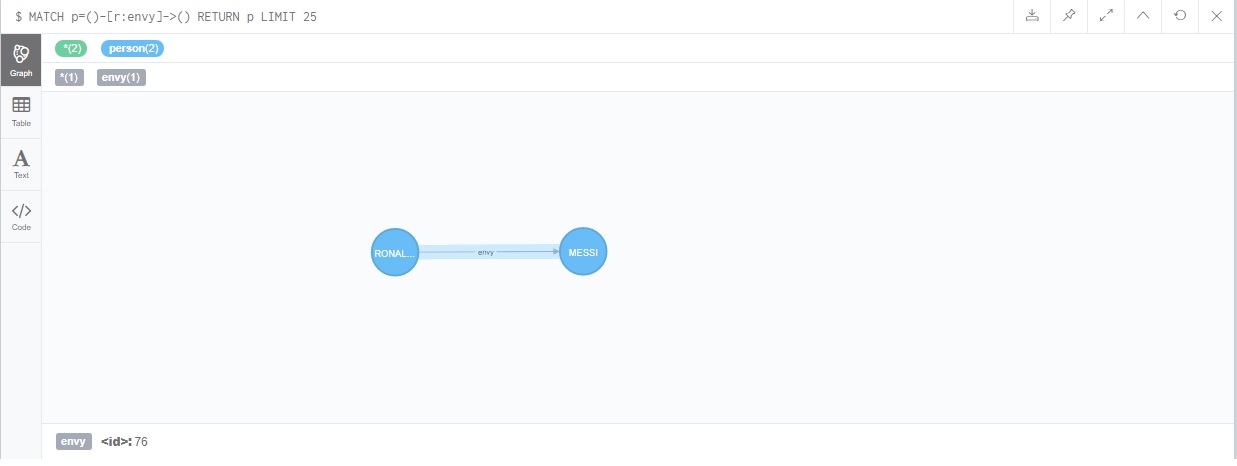


Figure 25: Creating node and unidirectional relationship between them.

*-- The code below makes messi also envy Ronaldo.*

*-- next command will make envy both directional*

MERGE(:person{name:'MESSI'})-[e:envy]->(:person{name:'RONALDO'});

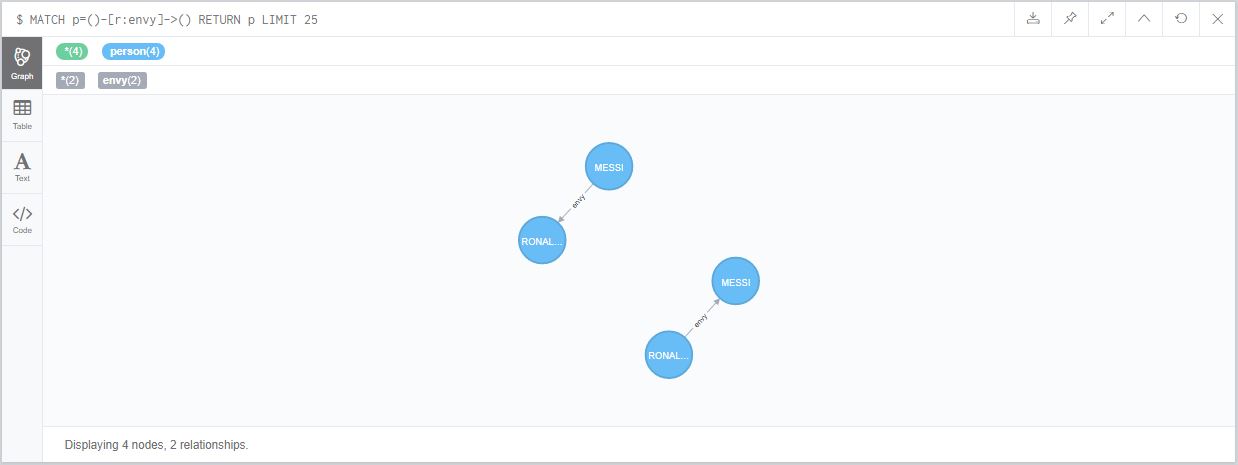


Figure 26: Merge Case V - Unidirectional Relationship.

-- Case VI:

*-- Two node exist without relationship; merge an un-directional relationship with MATCH and MERGE*

*-- So first delete the relationship*

MATCH(a:person{name:'MESSI'})-[e:envy]->(b:person{name:'RONALDO'}) DELETE e,a,b;



Figure 27: Deleting Nodes and relationship.

*-- OUTPUT: TWO NODES WITH RELATIONSHIP DELETED*

*-- NOW DELETE THE RELATIONSHIP OF REMAINING NODES*

MATCH(a:person{name:'RONALDO'})-[e:envy]->(b:person{name:'MESSI'}) DELETE e;



Figure 28: Deleting Relationship.

*-- TWO NODES EXIST WITH-OUT RELATIONSHIP, NOW GIVE UN-DIRECTIONAL RELATIONSHIP*

MATCH(a:person{name:'RONALDO'}),(b:person{name:'MESSI'}) MERGE (a)-[:envy]-(b);

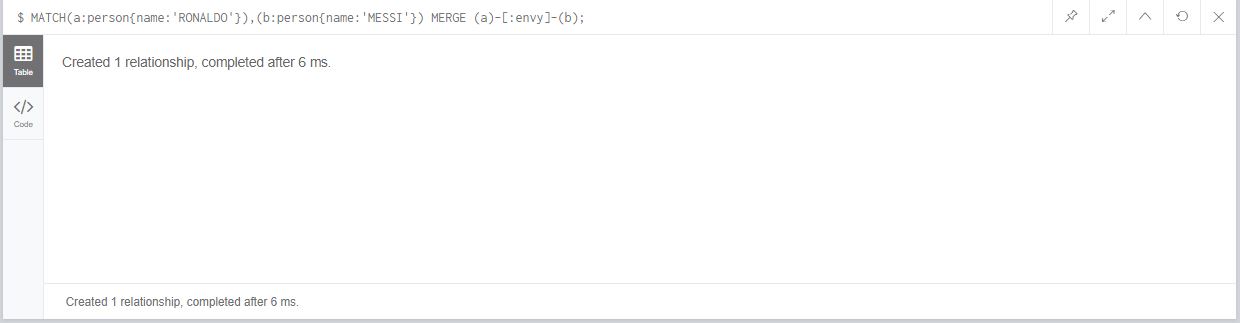


Figure 29: Merge Case VI Un-directed relationship.

-- Case VII:

*-- Two nodes with relatinship already exists, merge a un-directional relationship with match and merge.*

MATCH(a:person{name:'MESSI'}),(b:person{name:'RONALDO'}) MERGE (a)-[:envy]-(b);

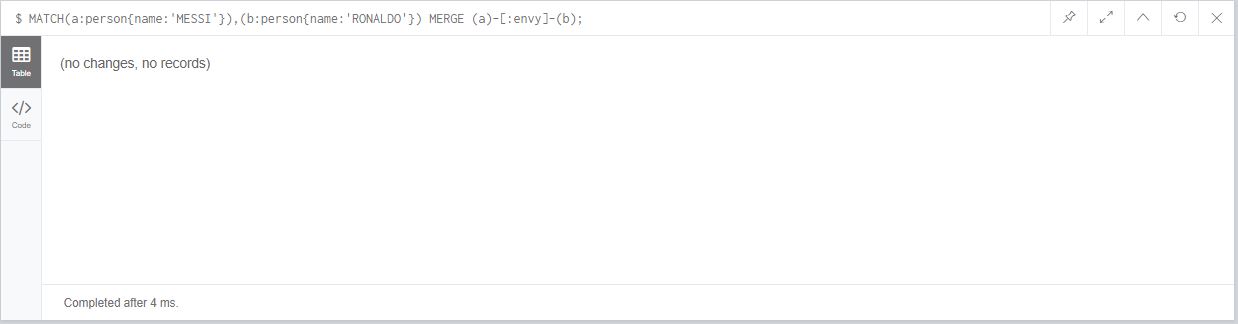


Figure 30: Merge Case VII.

-- Case VIII:

*--Two nodes exist with relationship, which has different direction with relationship to be merged*

*-- There exists a relationship where 'RONALDO' envy 'MESSI'*

MATCH (a:person{name:'MESSI'}),(b:person{name:'RONALDO'}) MERGE (a)-[:envy]->(b);

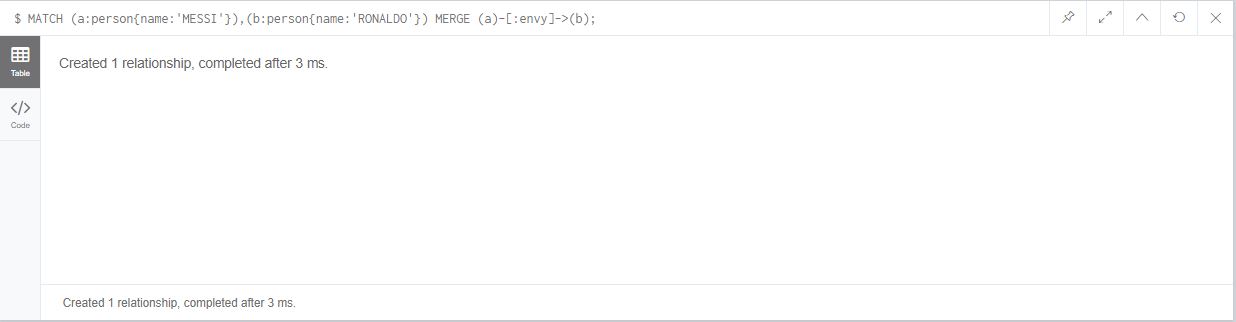


Figure 31: Merge Case VIII. No duplication of nodes and relationship.

*-- Now with the above command 'MESSI' also envy 'RONALDO'*

/\* --- LOGS --- \*/

Merge is a very useful command. It has a power to create itself also.

MERGE Command creates if the node, relationship does not exit and if nodes and relationship does not exist it will match and merge relationship.

Programmer needs to precise which merge case to use, in some case it can create duplication of nodes and relationship. And while using merge direction of the relationship must be taken carefully.

#### Cypher 4:

• Refer to the diagram on the previous slide, and write some commands to create a KNOWS relationship between the ‘Scott’ and ‘Gary’ nodes with an attribute: value ‘since: 1996’.

• You need to use the CREATE UNIQUE command.

• Refer to previous codes if you are not sure how to do it.

-Creating a node with label Person and property Scott

CREATE (scott:Person{name:'Scott'});

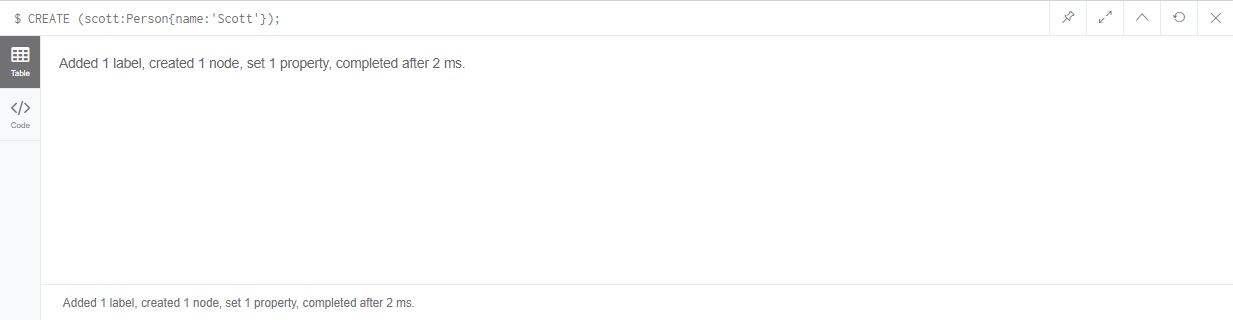


Figure 32: Creating a node.

--Create unique will prevent relationship duplication.

MATCH (scott:Person{name:'Scott'})

CREATE UNIQUE (scott)-[:KNOWS{since:1996}]->(gary:Person{name:'Gary'});



Figure 33: Match node and create unique relationship between them.

• Create some graph nodes (without an attribute called ‘marked’) and relationships based on the model on the previous slide.

• Run the FOREACH command from the previous slice to set the value of ‘marked’ attribute to ‘TRUE’.

• Verify the output.

-Creating a two nodes

CREATE (luis:Person{name:'Luis'}), (hector:Person{name:'Hector'});



Figure 34: Two nodes Created.

-- creating a unique relationship between the nodes.

MATCH(n:Person{name:'Gary'}),(m:Person{name:'Luis'}) MERGE (n)-[:KNOWS]->(m);

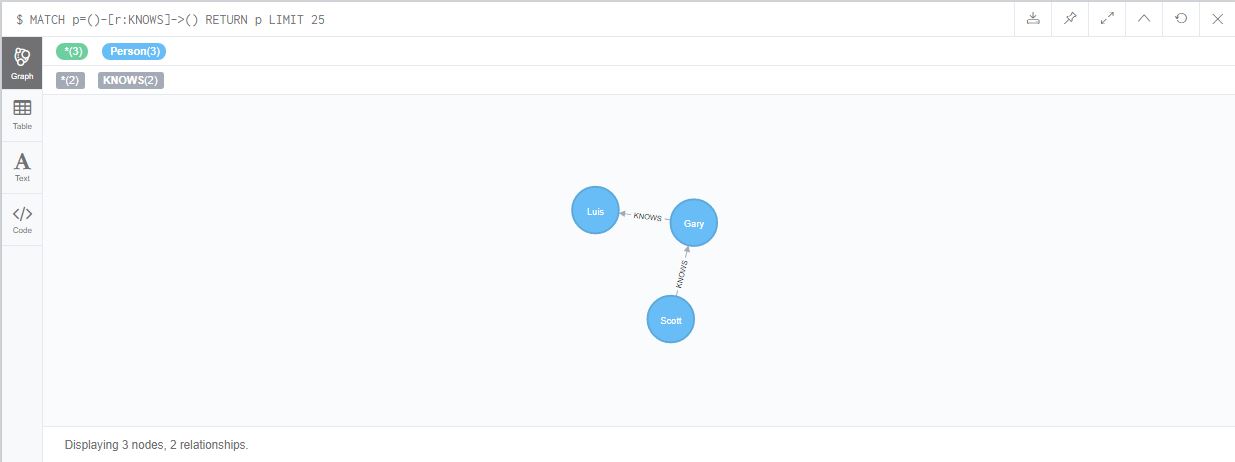


Figure 35: Unique Relationship 1.

MATCH(n:Person{name:'Luis'}),(m:Person{name:'Hector'}) MERGE (n)-[:KNOWS]->(m);

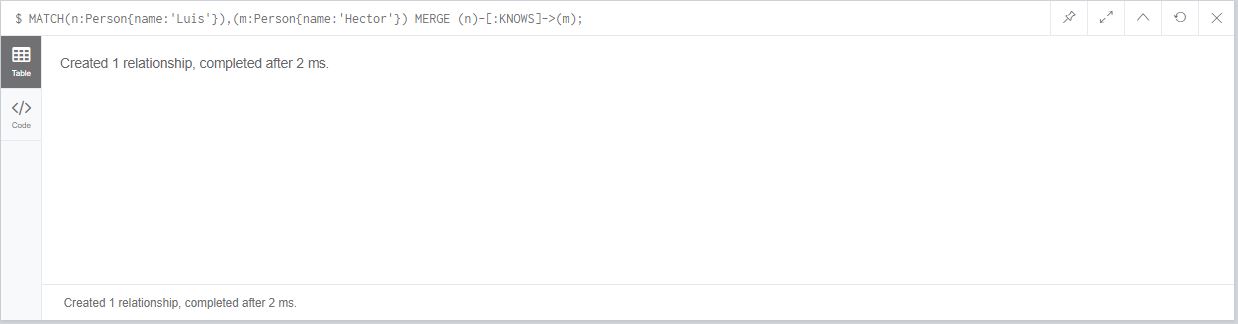


Figure 36: Unique Relationship 2.

-Creating a property marked true to each nodes of the same label.

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

WHERE begin.name = Scott AND end.name = 'Hector'

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



Figure 37: Creating property value for each nodes of same label.

-- verify the output

MATCH (n:Person{marked:true}) return n;

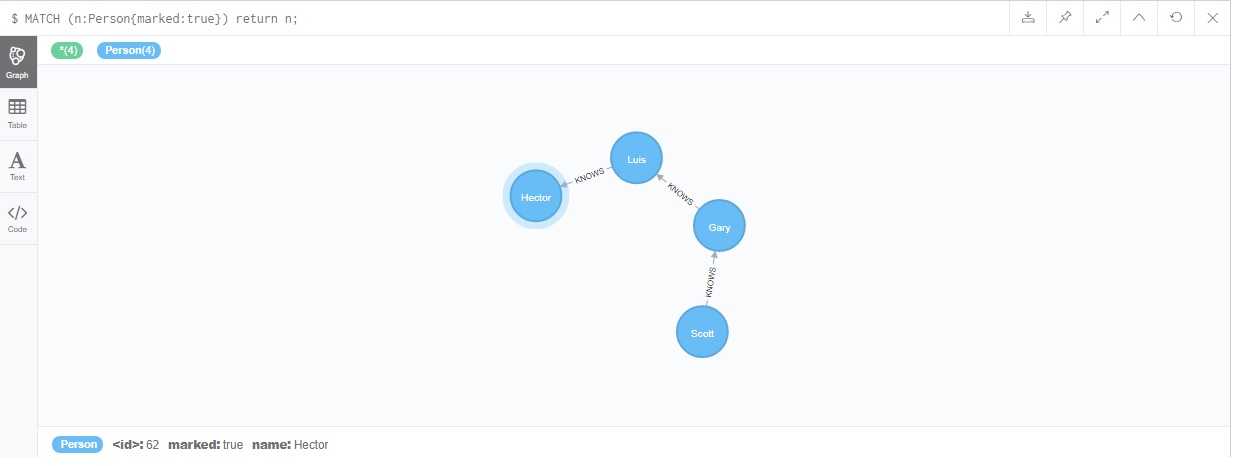


Figure 38: All nodes with property marked value true.

/\* --- LOGS --- \*/

Basically Merge clause and Create Unique Clause are same, both prevent duplication but Create unique is used to prevent relationship duplication.

In this exercise author get to know about the foreach clause which does, for each create the same property value pair for each node.

#### Cypher 5:

• Using the LOAD CSV command and load the data and create graph nodes with corresponding attributes and relationships.

• Research on how to load data in other formats (e.g., XML).

• Research on how to load data from online storage, e.g., Google sheet, Dropbox, GibHub, etc.

• Log your learning activity and reflection in the diary

-- loading a CSV file Without Header

LOAD CSV FROM

"file:///computing\_modules\_without\_headers.csv" AS row

CREATE(m:Module{code:row[0],title:row[1],level:toInteger(row[2]),credits:toInteger(row[3])})



Figure 39: Loading a csv file with-out headers.

-- loading a CSV file with headers

LOAD CSV WITH HEADERS FROM

"file:///computing\_modules.csv" AS row

CREATE(m:Module{code:row.code,title:row.title,level:toInteger(row.level),credits:toInteger(row.credits)})

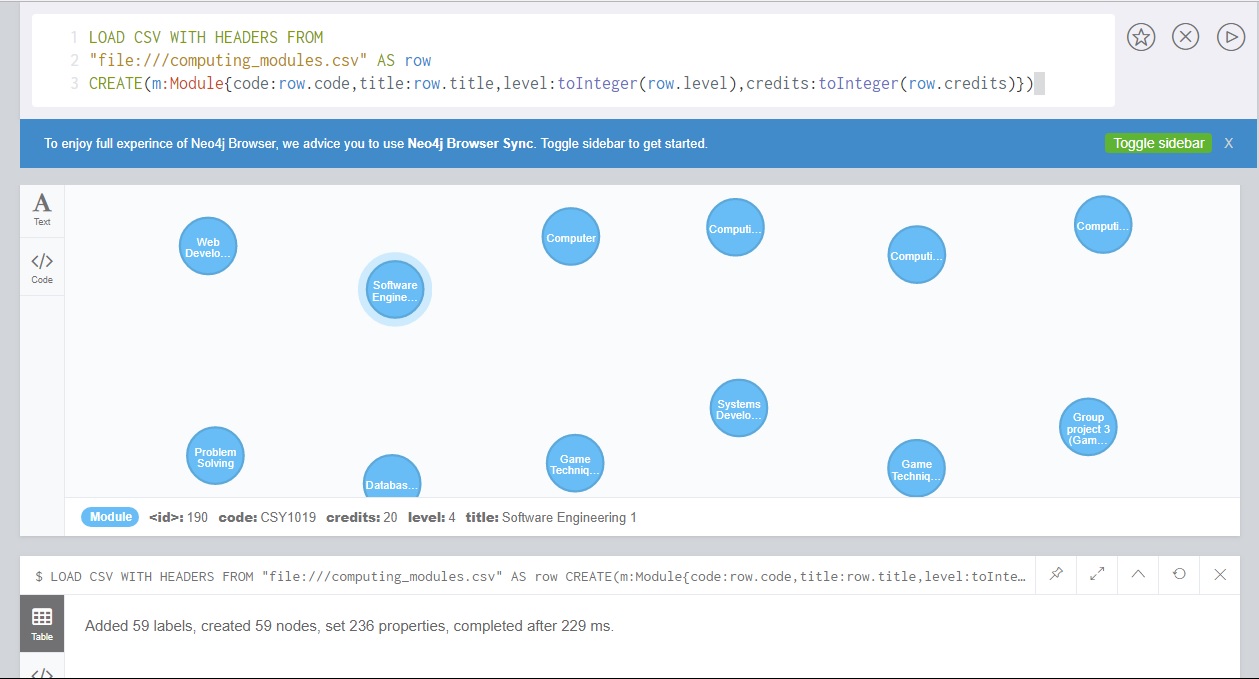


Figure 40: Loading a csv file with headers.

-- loading a CSV file WITH DELIMITER

LOAD CSV WITH HEADERS FROM

"file:///computing\_modules\_semicolon.csv" AS row

FIELDTERMINATOR ';'

CREATE (m:Module{code:row.code,title:row.title,level:toInteger(row.level),credits:toInteger(row.credits)})

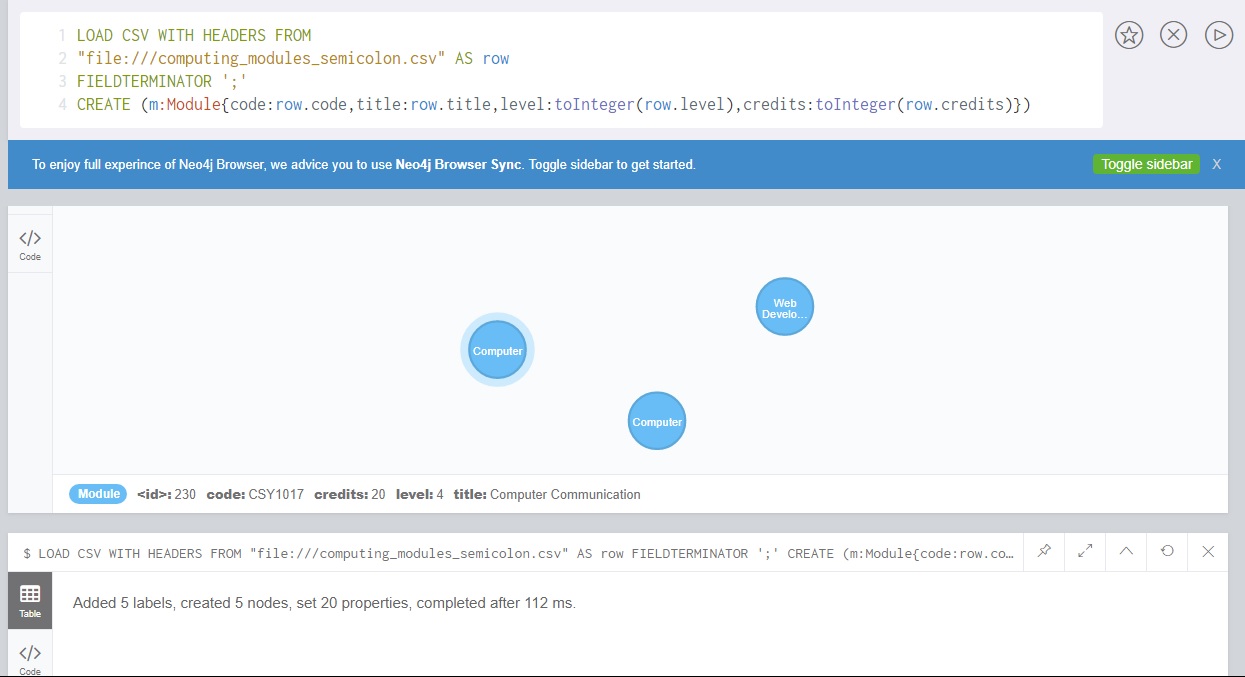


Figure 41: loading a csv file with field delimiter.

-- loading a CSV file pre-requisites.csv

LOAD CSV WITH HEADERS FROM

"file:///computing\_pre\_requisites.csv" AS row

MATCH(n:Module{code:row.module1}),(m:Module{code:row.module2})

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

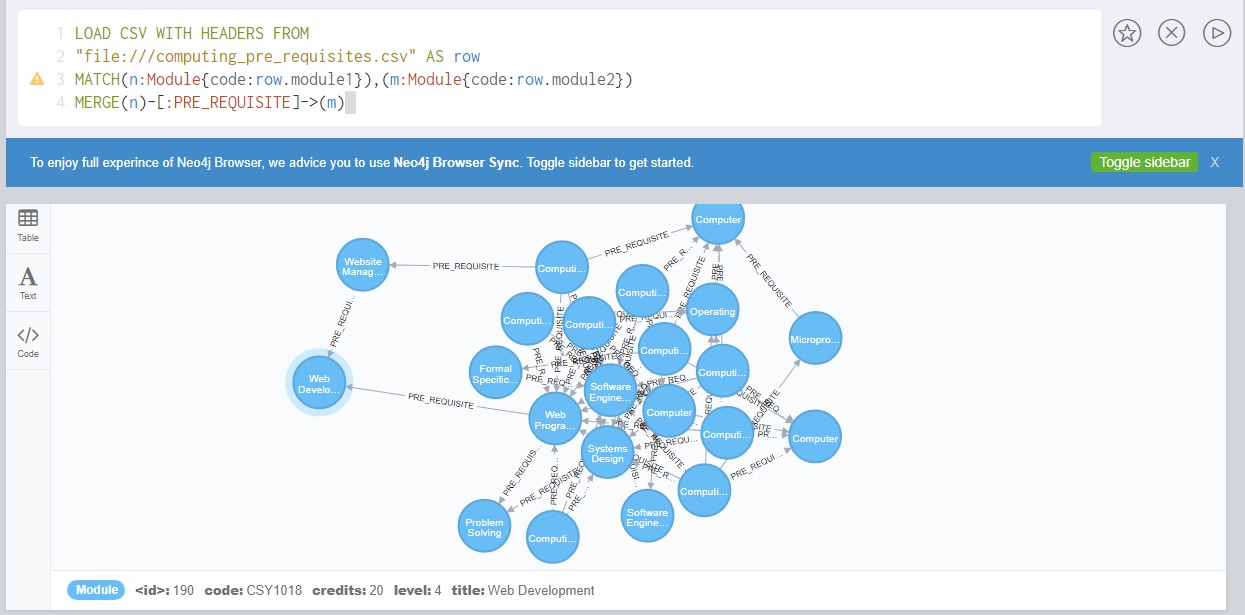


Figure 42: merging pre-requiste relationship between modules.

-- loading a CSV file Co-requisites.csv

LOAD CSV WITH HEADERS FROM

"file:///computing\_co\_requisites.csv" AS row

MATCH(n:Module{code:row.module1}),(m:Module{code:row.module2})

MERGE(n)-[:CO\_REQUISITE]->(m)

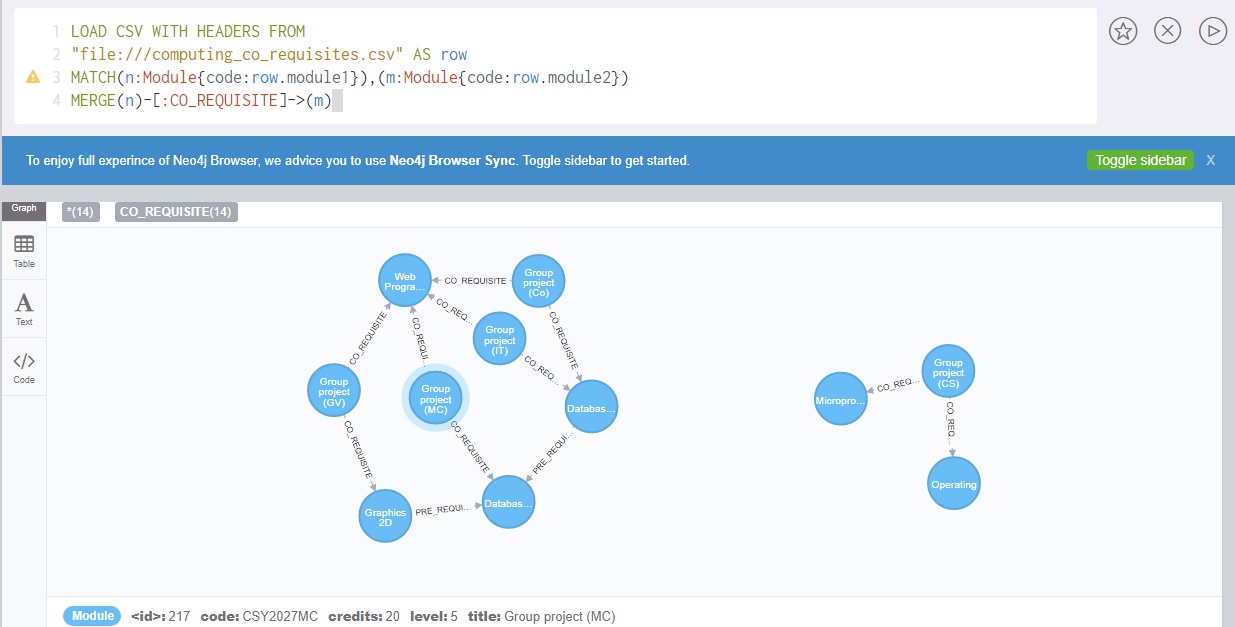


Figure 43: merging co-requiste relationship between modules.

-- loading a CSV file Computing\_pathways.csv

LOAD CSV WITH HEADERS FROM

"file:///computing\_pathways.csv" AS row

CREATE (p:Pathways{code:row.code,title:row.title})

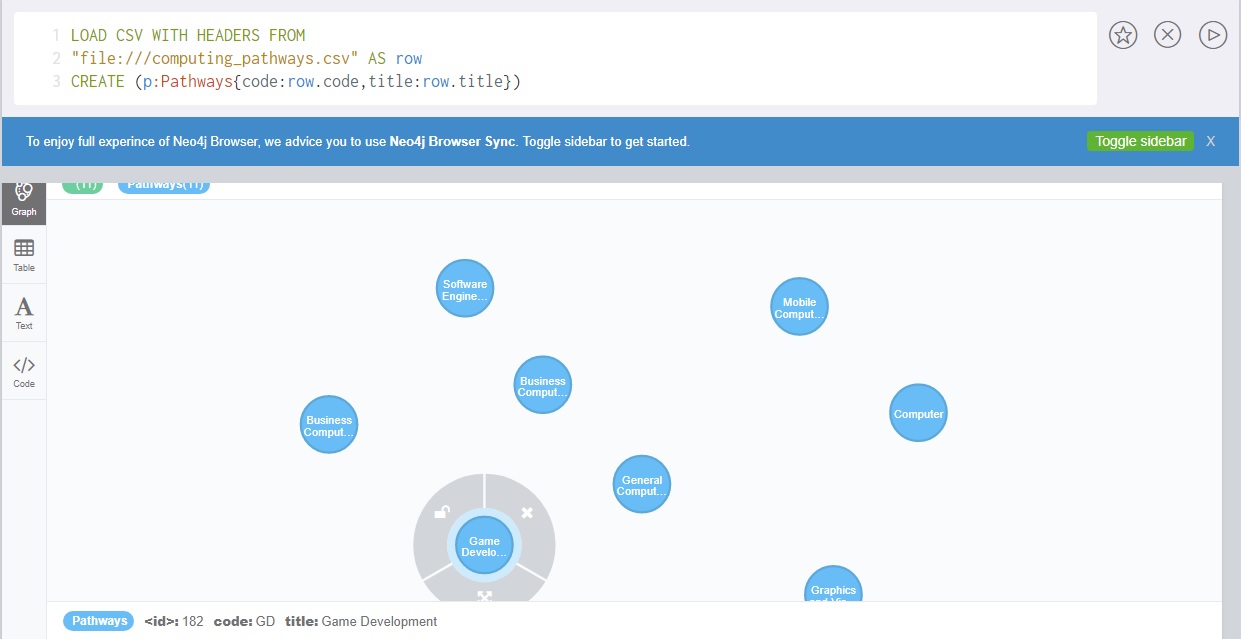


Figure 44: loading pathway csv file.

-- loading a CSV file computing\_on\_pathway.csv and merging relationship between module and pathway

LOAD CSV WITH HEADERS FROM

"file:///computing\_on\_pathway.csv" AS row

MATCH(n:Module{code:row.m\_code}),(m:Pathways{code:row.pw\_code})

MERGE(n)-[:ON\_PATHWAY]->(m)

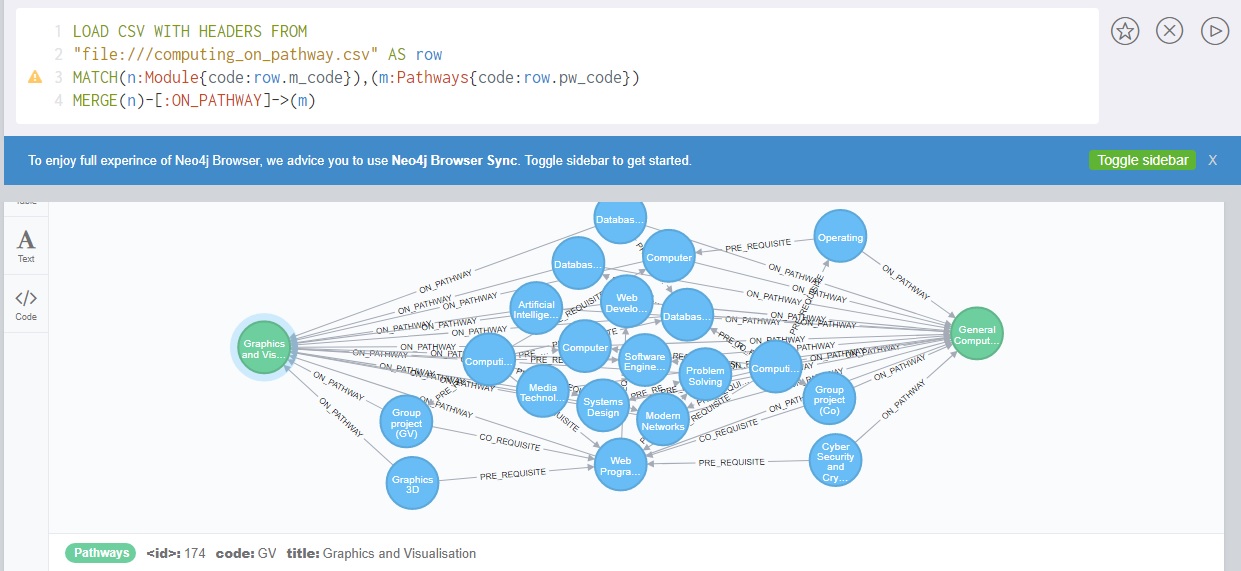


Figure 45: merging relationship between two nodes of different labels.

/\* --- LOGS --- \*/

Three type of csv file can be load file without headers, with headers and file with field delimiter.

On how to load xml file to neo4j, there is an apoc procedure library from which xml file can be loaded. The code is:

CALL apoc.load.json(“file:/users/filename.json”) YIELD value

RETURN value;

Loading google sheet

The spreadsheet needs to be publicly accessible over the internet. Otherwise Google will ask for authenticate first, and the Neo4j Load CSV process will not know how to do that. Generate the download URI of the CSV export. Then look at your browser download history to figure out what the URI of the download was at last

LOAD CSV WITH HEADERS FROM “URI”

Author finds loading a csv file easy option than other. If the dataset is not on csv there is a lot of converter website which can convert json file to csv.

NEXT EXERCISE

• Clear the database.

• Load the Movie graph database.

• Run the commands to practice ”shortestPath()” and “allShortestPaths()” functions

• Make the following changes and run the codes again:

• Change the path lengths from “\*..15” to “\*..5” in the first “shortestPath” example

• Remove the NONE command in the second “shortestPath()” example

• Run the commands on the previous slides to practice the MATCH and OPTIONAL MATCH clause.

• Log your learning activities and reflection in the diary.

--- clearing the database

MATCH(n) DETACH DELETE n;

--- -- displaying the shortest path between tom and laurence

MATCH (tom:Person{name:"Tom Cruise"}),

(laurence:Person{name:"Laurence Fishburne"}),

p=shortestPath((tom)-[\*]-(laurence))

WHERE NONE (r IN rels(p) WHERE type(r) = "DIRECTED")

RETURN p

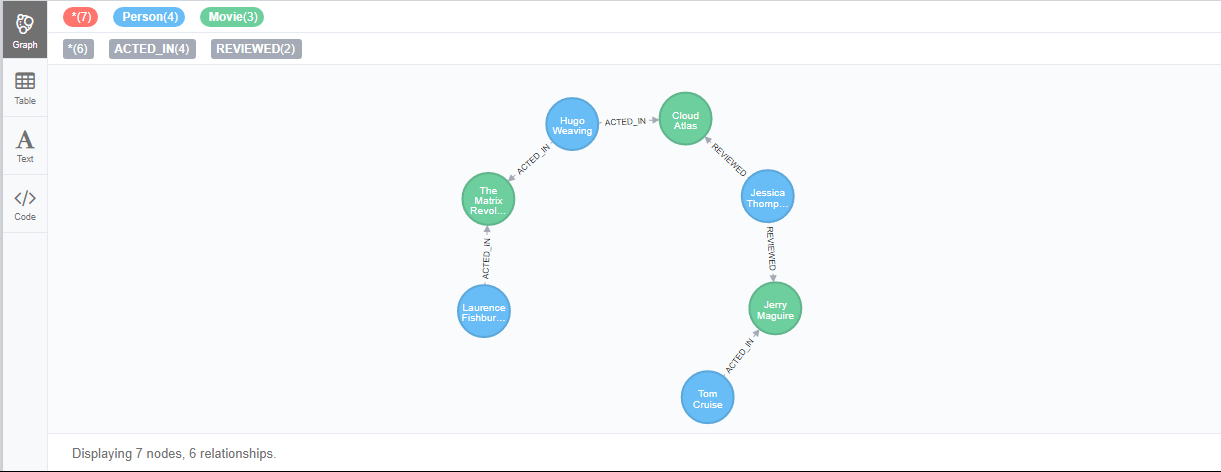


Figure 46: shortest path between tom and laurence.

--- displaying the all shortest path between tom and laurence

MATCH (tom:Person{name:"Tom Cruise"}),

(laurence:Person{name:"Laurence Fishburne"}),

p=allShortestPaths((tom)-[\*]-(laurence))

RETURN p

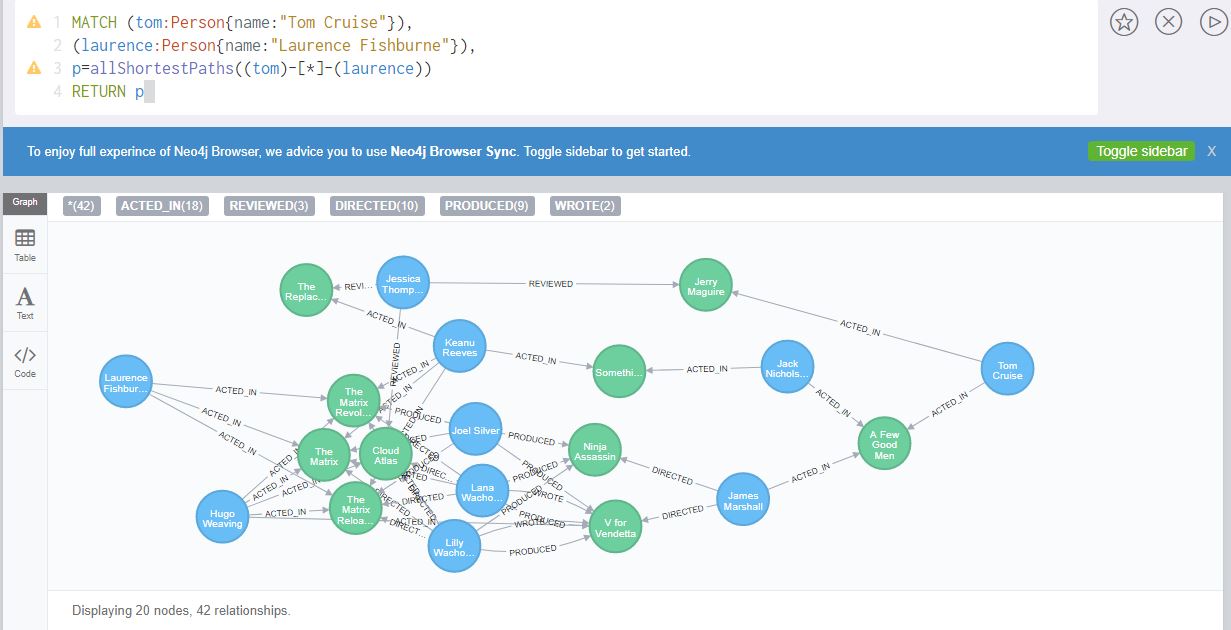


Figure 47: displaying all the shortest path between tom and laurence.

--- -- displaying the shortest path between tom and laurence

MATCH (tom:Person{name:"Tom Cruise"}),

(laurence:Person{name:"Laurence Fishburne"}),

p=shortestPath((tom)-[\*..5]-(laurence))

RETURN p



Figure 48: shortest path between 5 relationships.

Outputs: no changes because node tom and node Laurence shortest path has 7 relationship so shortest path does not come under 5.

--- Displaying the shortest path without NONE clause

MATCH (tom:Person{name:"Tom Cruise"}),

(laurence:Person{name:"Laurence Fishburne"}),

p=shortestPath((tom)-[\*]-(laurence))

RETURN p

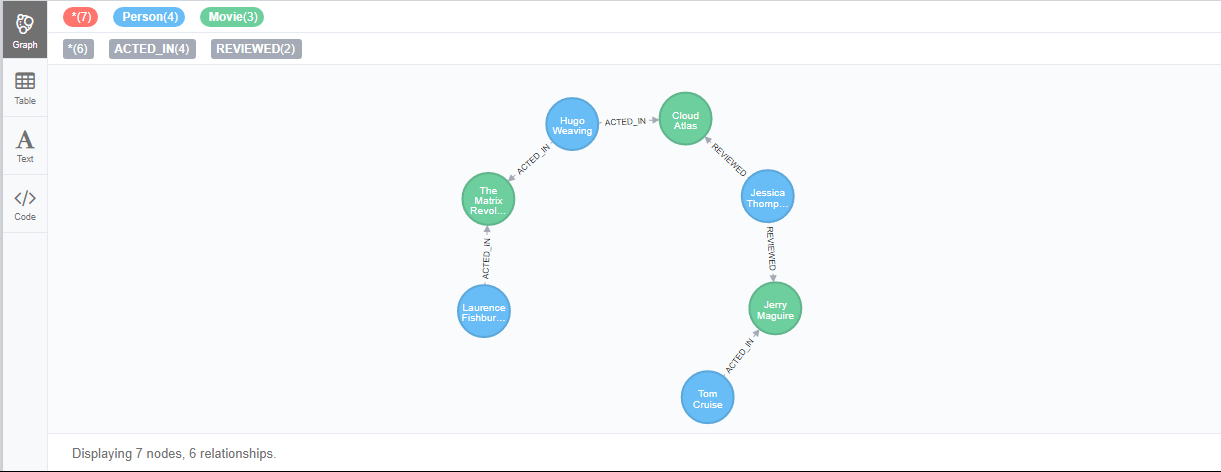


Figure 49: displaying the shortest path without none clause.

--- Match returns no change

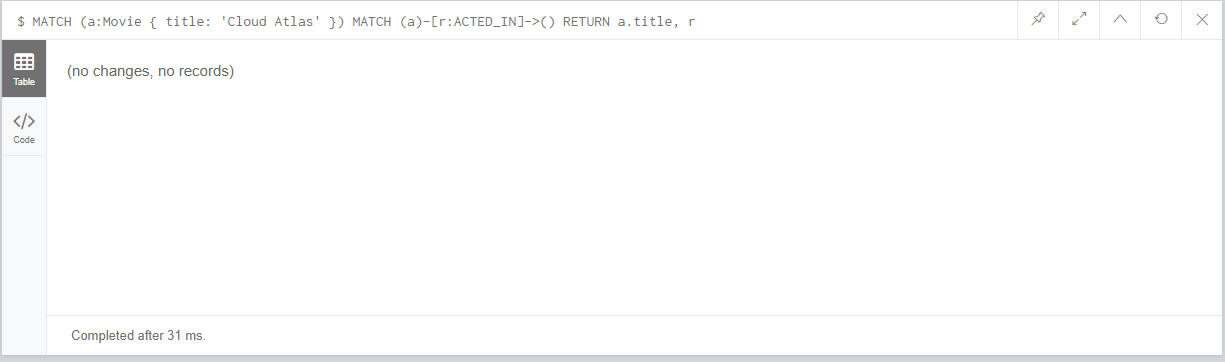


Figure 50: match return no change.

--- Optional match return null

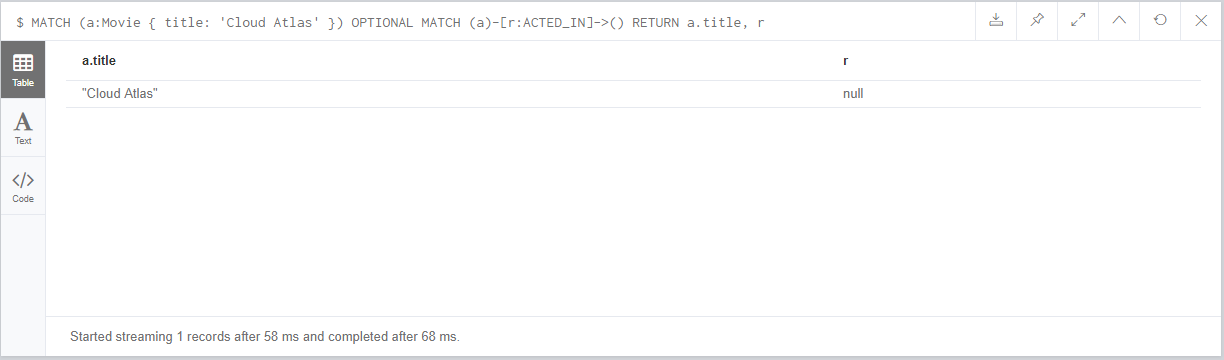


Figure 51: Optional match.

/\*-- Logs --\*/

Through this activity author gets to know the way to find out the shortest way possible of finding the relationship between the two nodes. There are two function to find the shortest way possible one is shortestPath and another is allShortestPaths and inside these function goes two individuals nodes.

And the next thing is about optional match, optional match are like match but where it differ is if no matches are found it will use null for the missing parts.

#### Cypher 6:

**Exercises**

• Create some nodes and relationships based on the model from previous slides.

• Run the reference commands on the previous slides to practice WHERE clause.

• Log your learning activities and reflection in the diary.

-- creating a node

CREATE (n:Person{name: 'Sandeep', age:'21', sex:'Male'});

CREATE (n:Person{name: 'Rukshan', age:'20', address:'Chabahil'});

CREATE (n:Person{name: 'Prakash', age:'21', email:'lightstone@gmail.com'});

-- creating a relationship between the nodes

match(n:Person{name:'Sandeep'}),(m:Person{name:'Rukshan'}) MERGE(n)-[r:KNOWS{since:'2015'}]-(m)

match(n:Person{name:'Sandeep'}),(m:Person{name:'Prakash'}) MERGE(n)-[r:KNOWS{since:'2014'}]-(m)

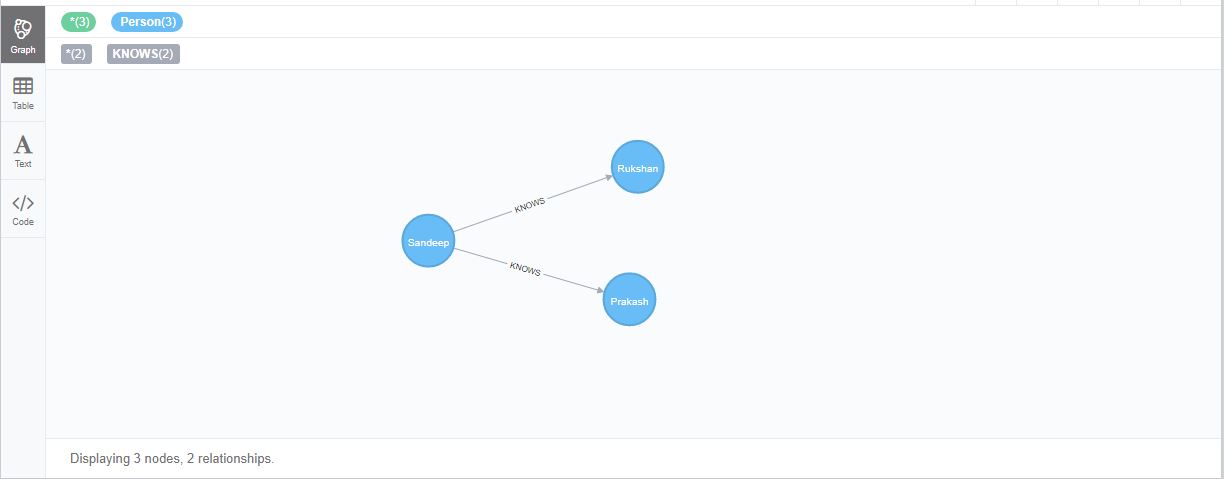


Figure 52: nodes with relationship.

-- boolean operation

match(n) WHERE n.name = 'Rukshan' XOR (n.age < 25 AND n.name='Prakash')

OR NOT (n.name='Prakash' OR n.name='Rukshan')

RETURN n;

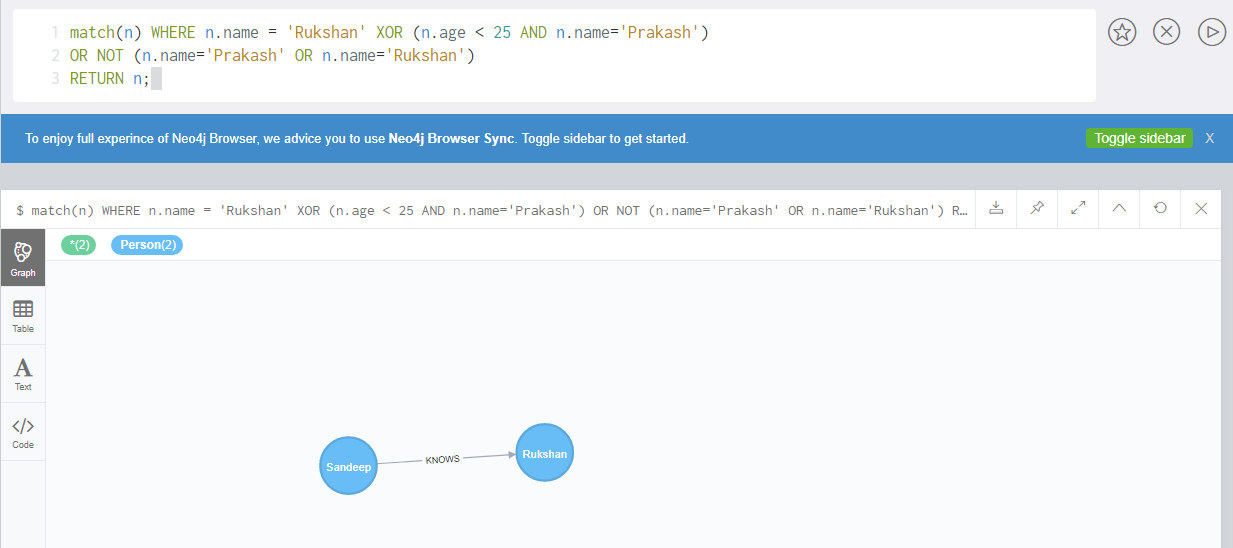


Figure 53: Boolean operation.

-- filter on node label

match(n) WHERE n:Persos RETURN n.name;

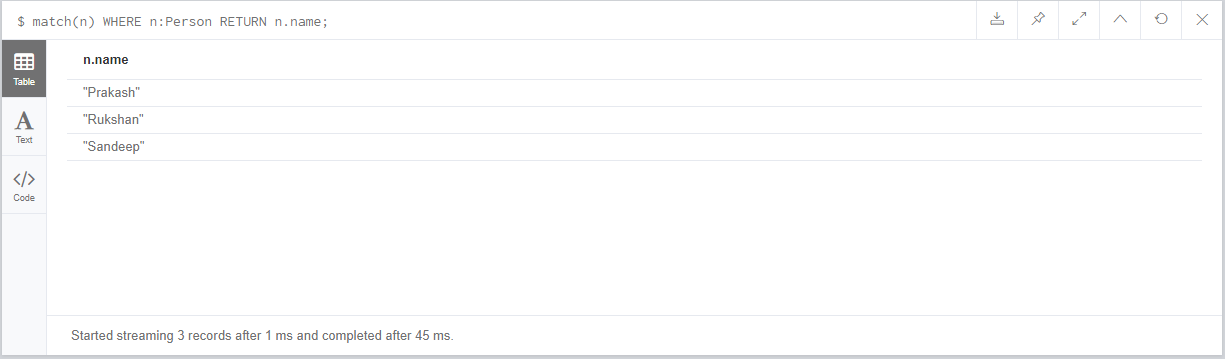


Figure 54: filtering on node label.

-- filter on node property

MATCH(n)

WHERE toInt(n.age ) > 20

RETURN n.name, n.age

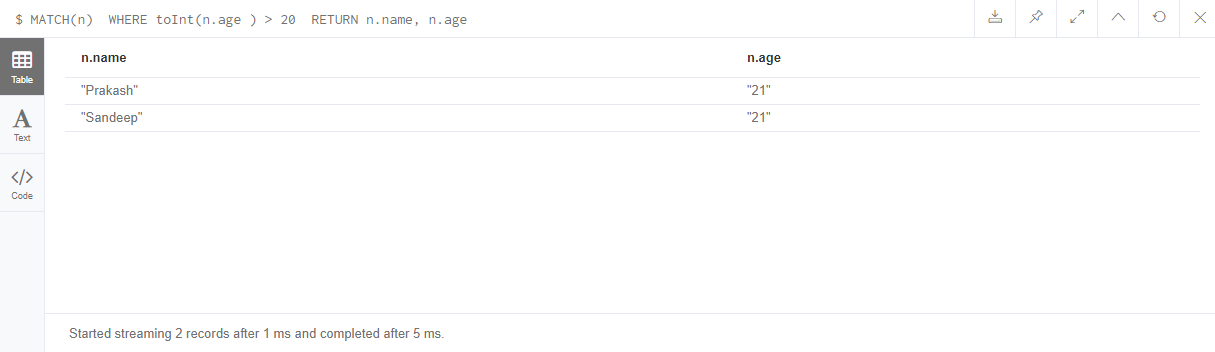


Figure 55: where clause on node property.

--- Filter On Relationship Property

match(n)-[k:KNOWS]->(f) WHERE toInt(k.since) < 2015 RETURN f

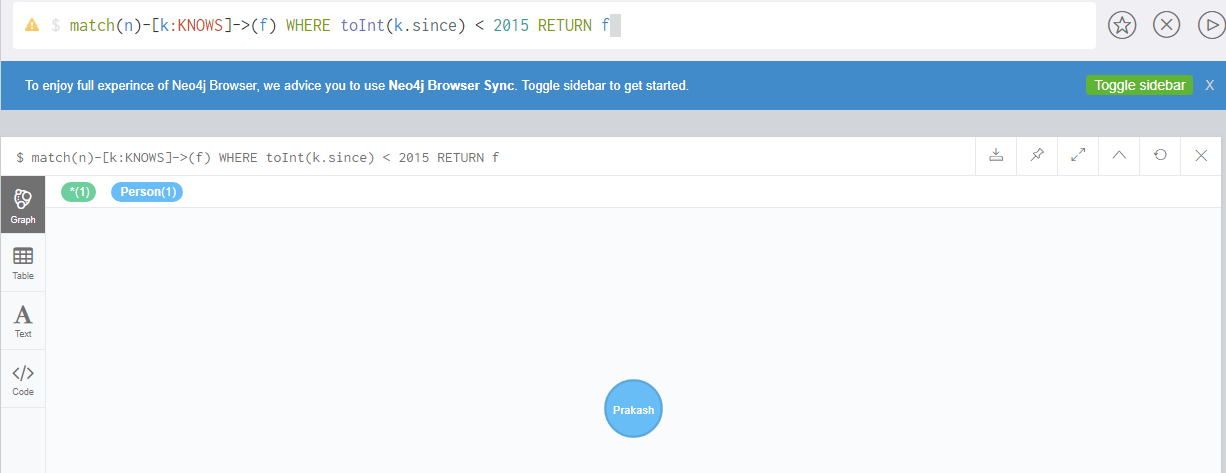


Figure 56: where clause on relationship property.

--- property exists

match(n) WHERE exists(n.sex) RETURN n;

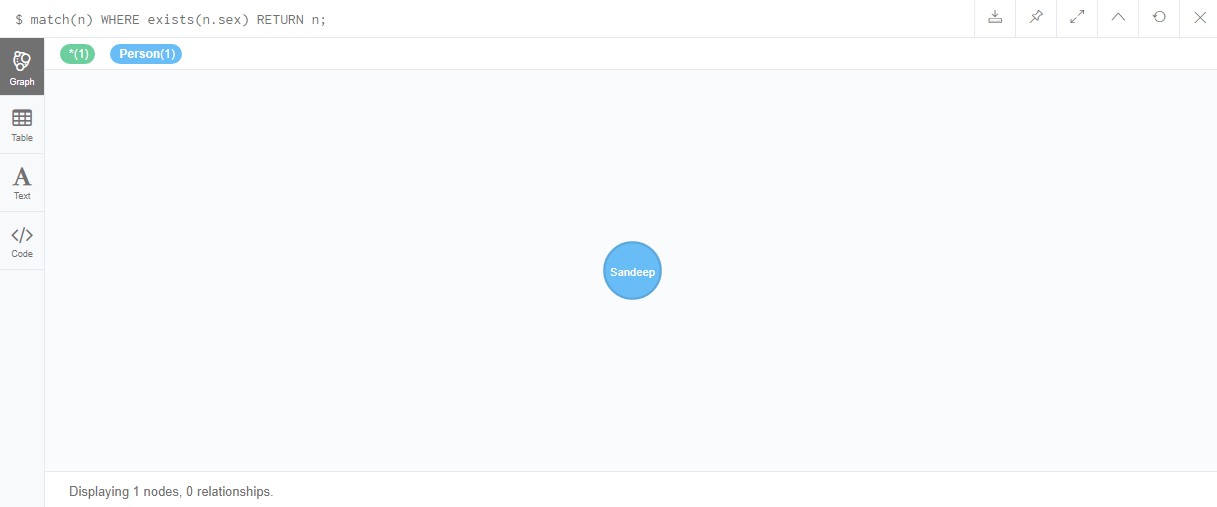


Figure 57: where clause on property exists.

-- match the start of the string

match(n) WHERE n.name STARTS WITH 'Ruk'

RETURN n

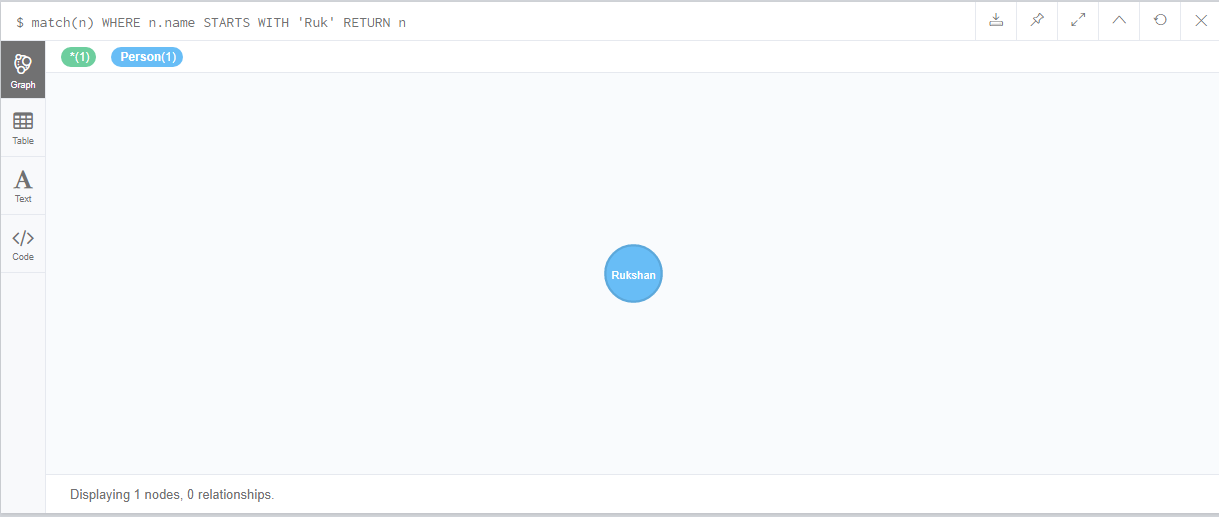


Figure 58: where clause on starting of the string.

-- match the end of the string

match(n) WHERE n.name ENDS WITH 'han'

RETURN n

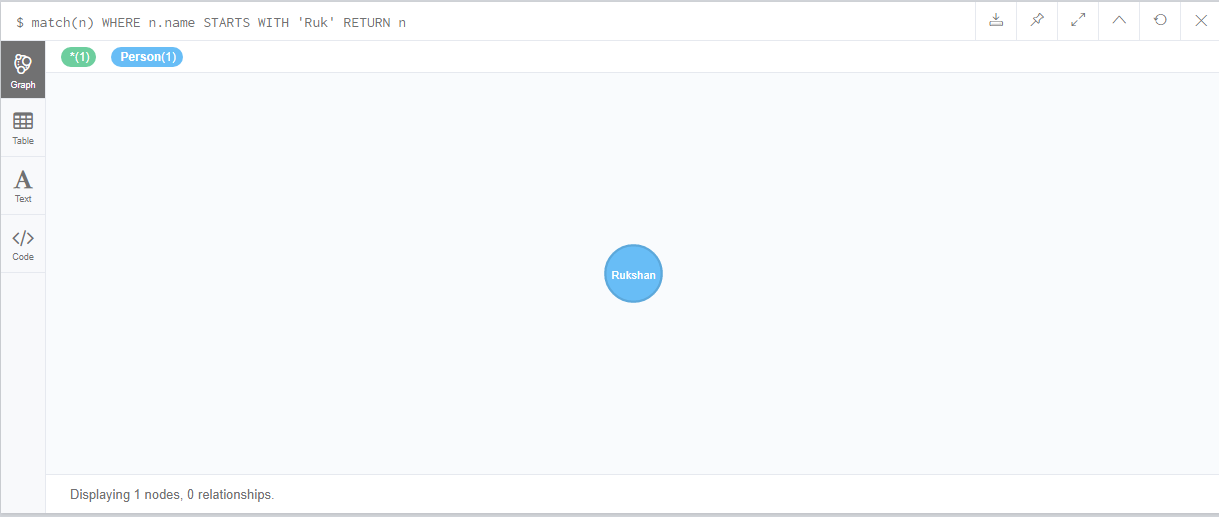


Figure 59: where clause to match the end of string.

-- Match Anywhere In A String

match(n) WHERE n.name CONTAINS 'ndee'

RETURN n

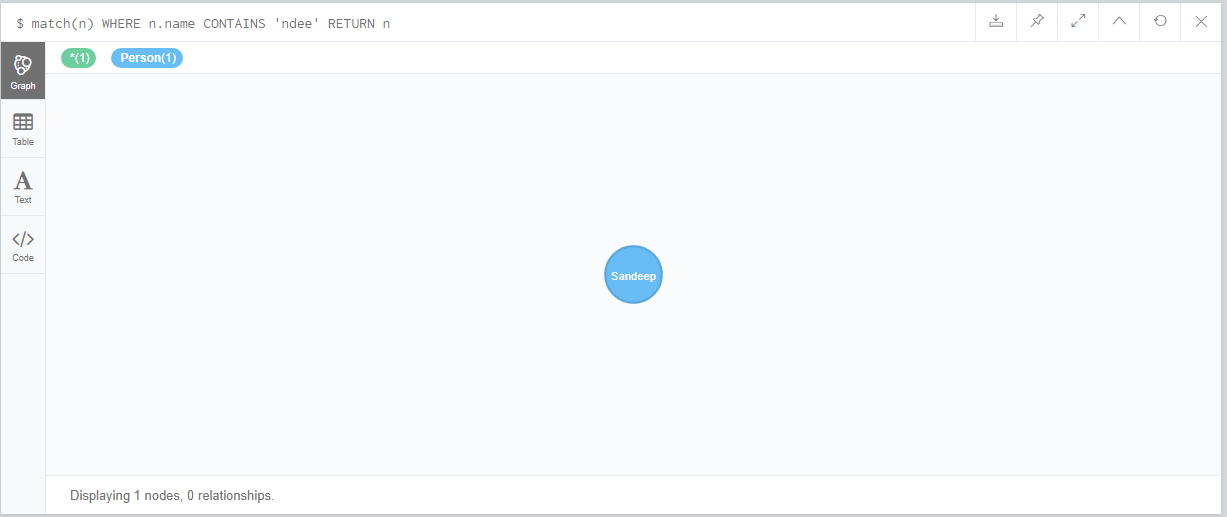


Figure 60: where clause to match anywhere in the string.

-- String matching Negation

match(n) WHERE NOT n.name ENDS WITH 'p'

RETURN n

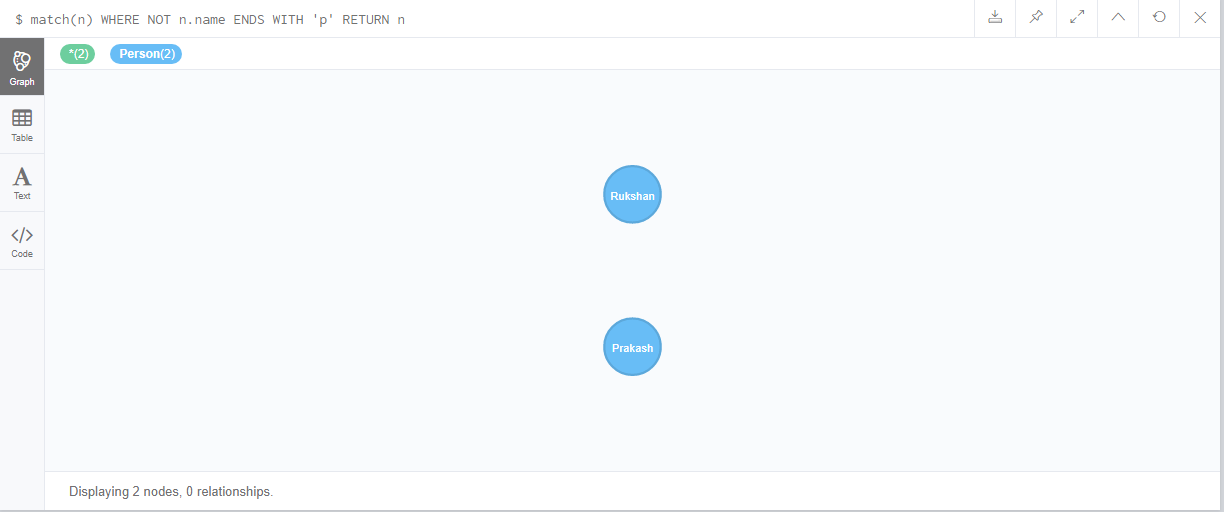


Figure 61: filtering where string does not ends with p.

-- Regular Expressions

match(n) WHERE n.name =~ 'Ruk.\*'

RETURN n

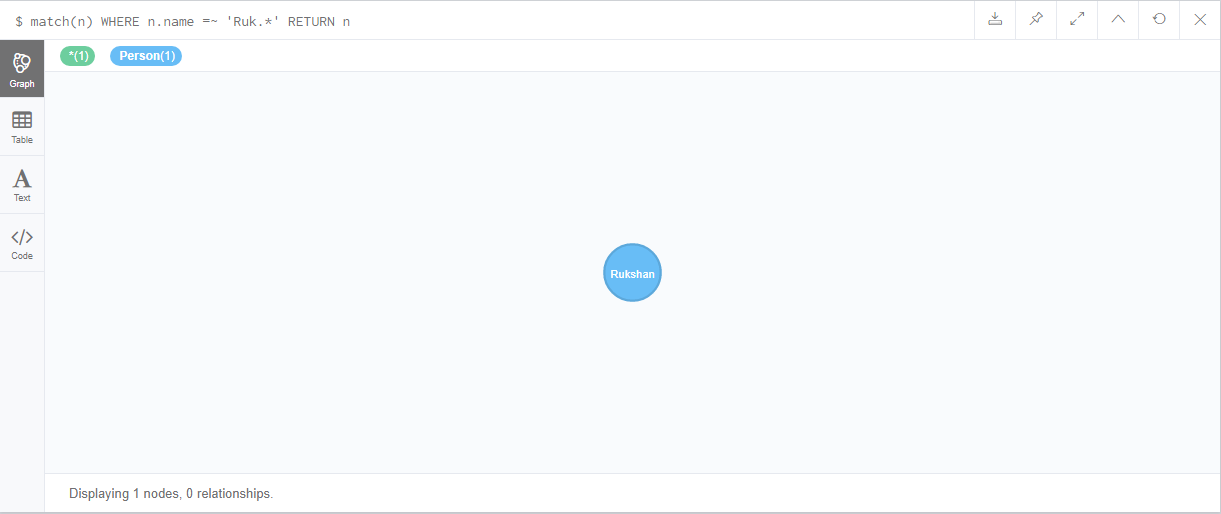


Figure 62: filtering with the regular expression.

-- Case Insensitive Regular Expressions

match(n) WHERE n.name =~ '(?i)RUKS.\*'

RETURN n

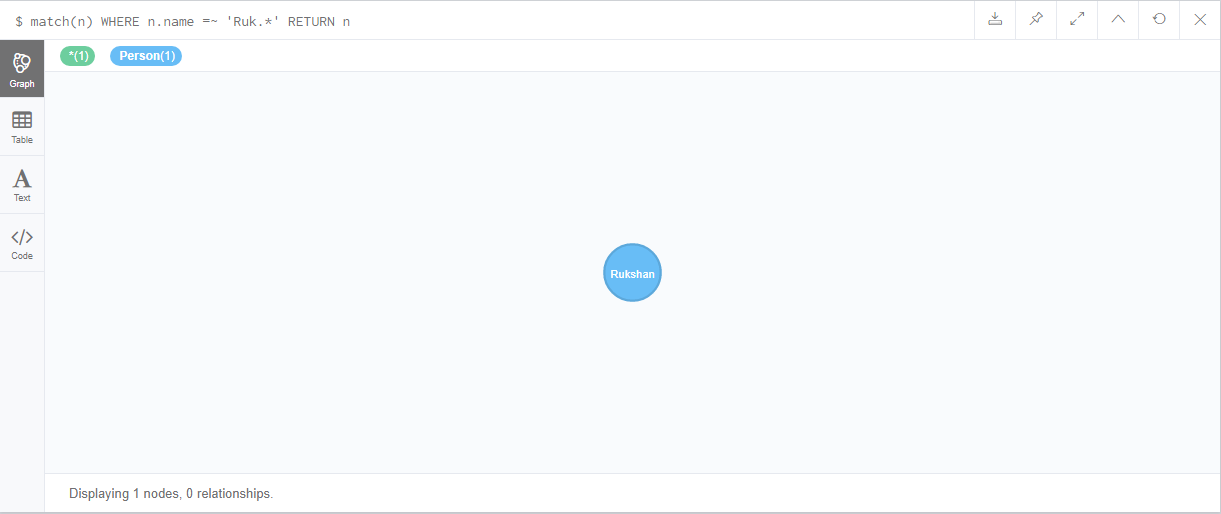


Figure 63: CASE INSENSITIVE REGULAR EXPRESSION.

-- Using Path Patters in where clause

match (n{name:'Prakash'}),(m)

WHERE m.name IN ['Sandeep', 'Rukshan'] AND (n)<--(m)

RETURN m

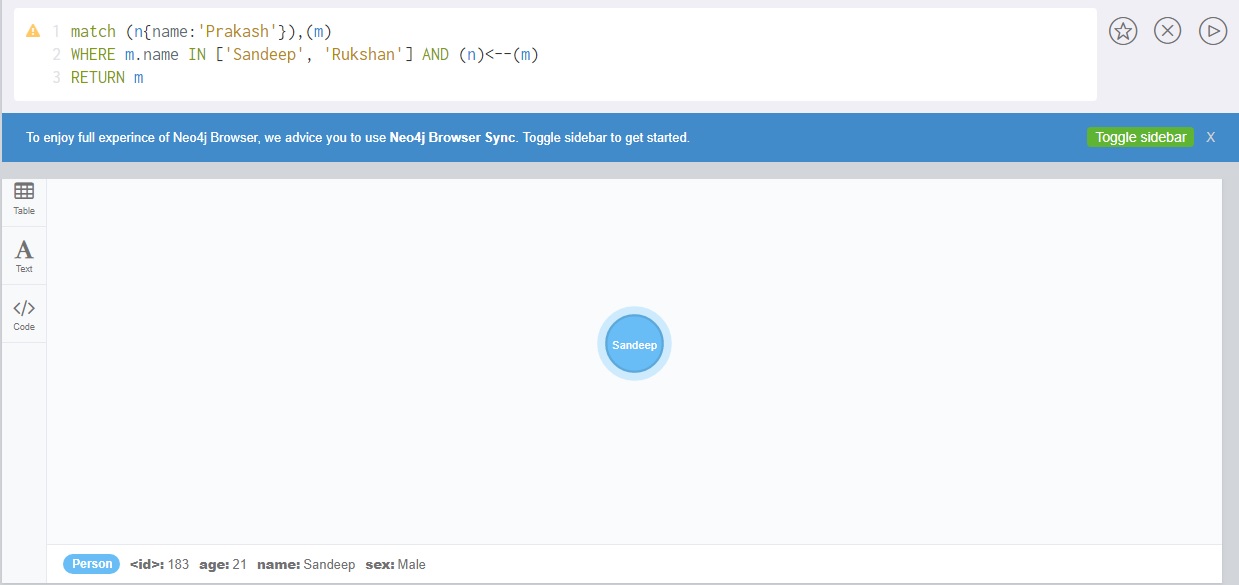


Figure 64: using path patter in where clause.

-- Filter On Patters Using Not

match (m), (n{name:'Rukshan'})

WHERE NOT (m)-->(n)

RETURN m

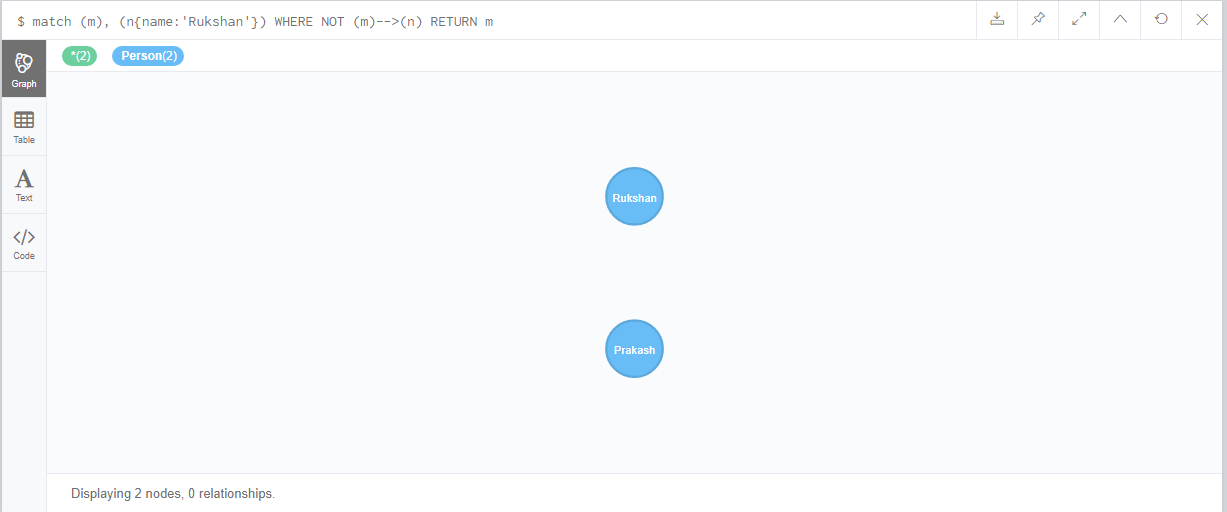


Figure 65: filtering patters using not clause on where

-- Filter On Patterns With Properties

match(n) WHERE (n)-[:KNOWS]-({name:'Prakash'})

RETURN n;

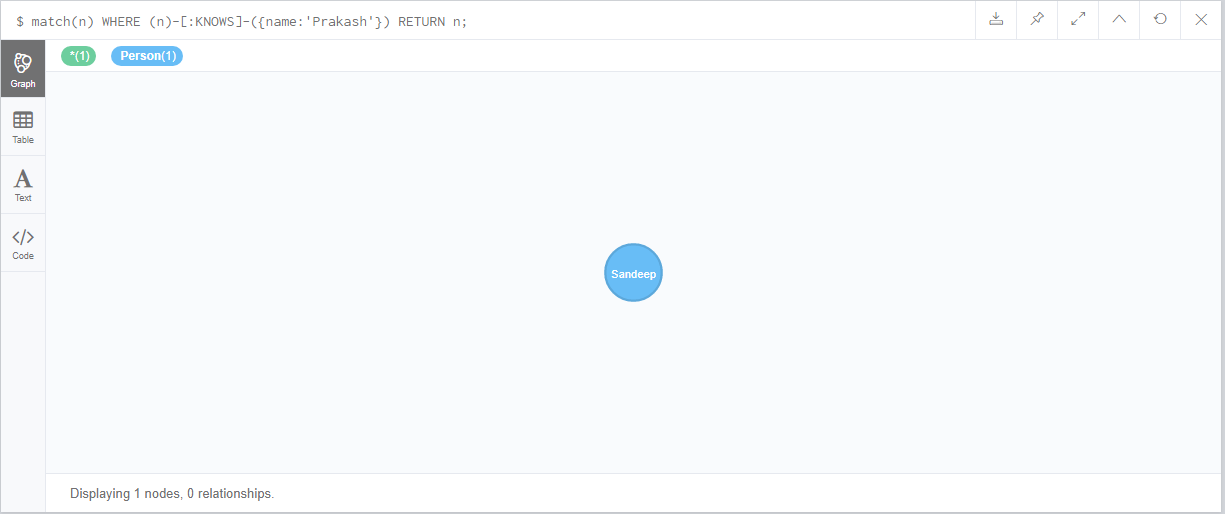


Figure 66: filtering on patter with properties.

-- Filter On Relationship type

match (n)-[r]->(m)

WHERE n.name='Sandeep' AND type(r)=~'K.\*'

RETURN n;

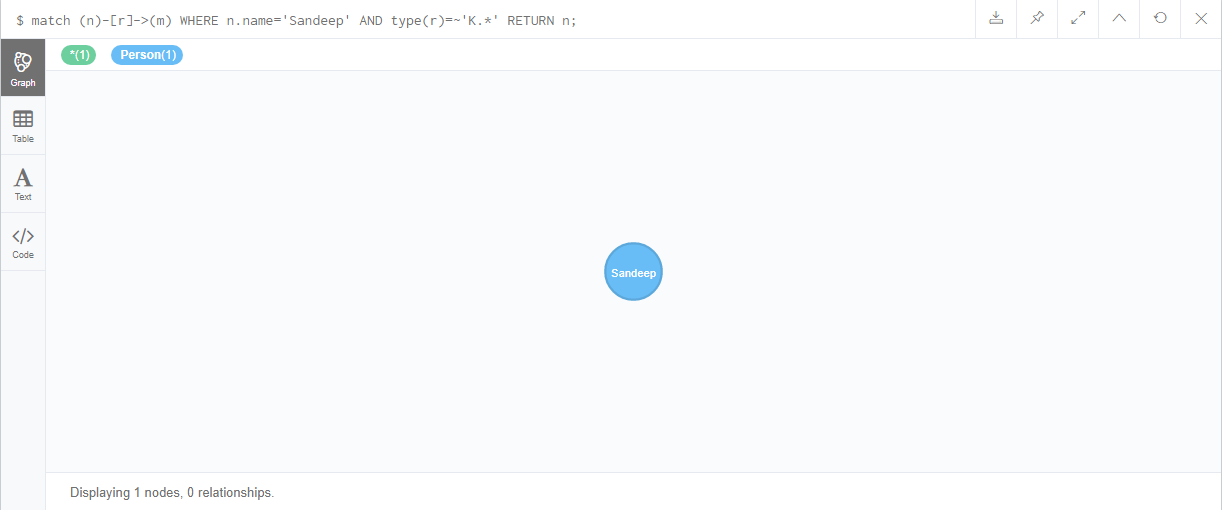


Figure 67: filter on relationship type.

-- List

match (n)

WHERE n.name IN['Rukshan', 'Prakash']

RETURN n;

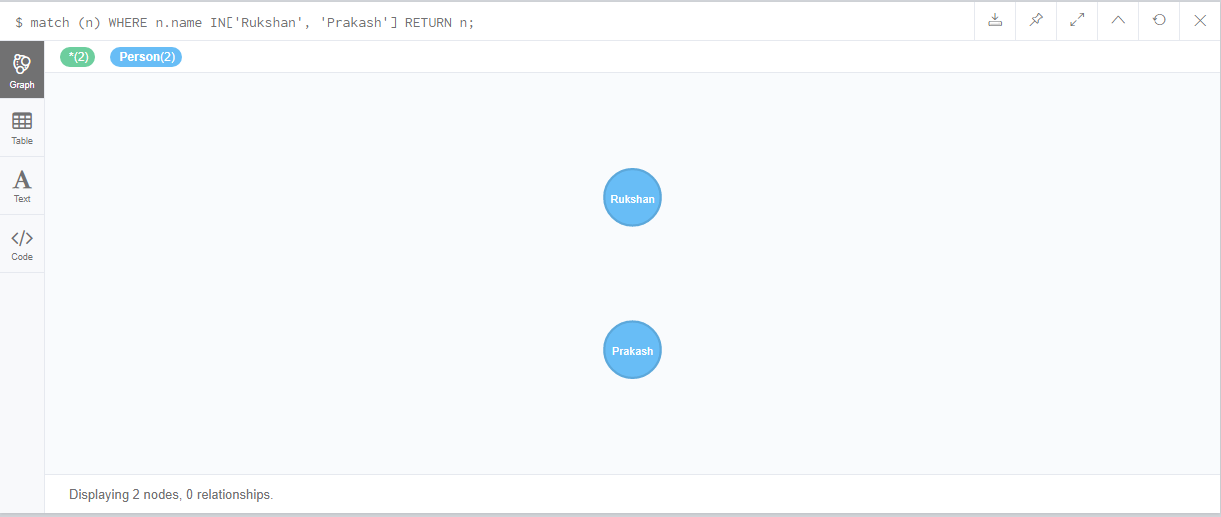


Figure 68: filtering with the list in where clause.

-- Missing properties and values

match (n)

WHERE n.sex = 'Male'

RETURN n;

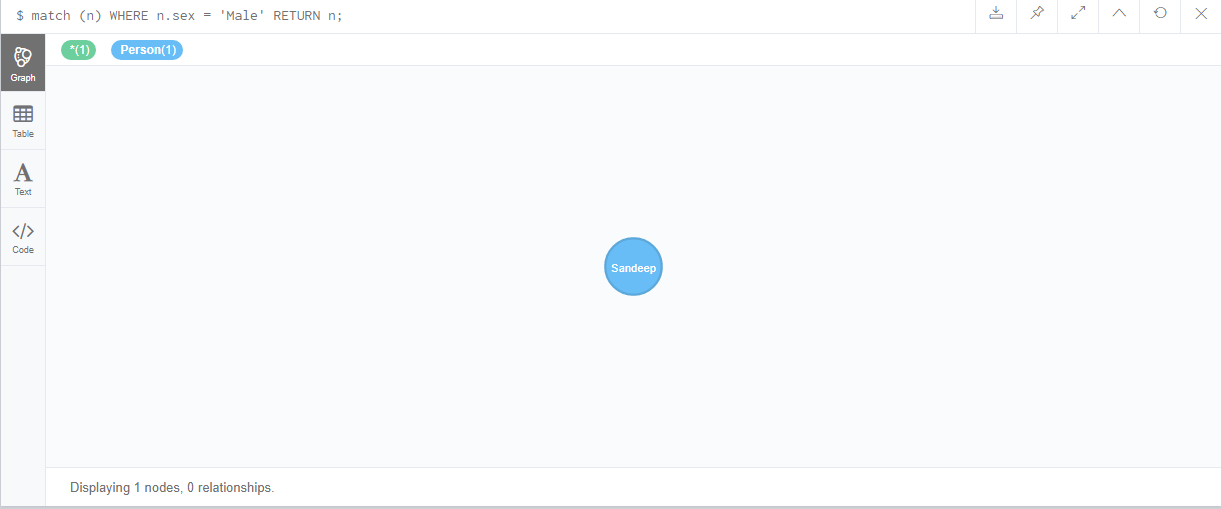


Figure 69: filtering with the matching property.

/\*-- Logs --\*/

Completing this exercise author know there are tons of way that can be used to get the required result. Where clause can be used in labels, nodes, relationship. Filtering are done in various ways, to match string there are lot of function like to match starting of the string there is STARTS WITH so wise for match ending of the string there is END WITH and there is also CONTAINS function to match anywhere of the strings. And there is also a regular expression to match strings.

## PART II

-- Loading club.csv file

LOAD CSV WITH HEADERS FROM ("file:///club.csv") AS row

create(n:Club {name:'row.club'};

--Loading result.csv file and making relationship between home and away club

LOAD CSV WITH HEADERS FROM ("file:///result.csv") AS row

match(a:Club{name:row.home}),(b:Club{name:row.away}) Merge (a)-[r:played\_with]->(b);

--Loading attributes to relation

LOAD CSV WITH HEADERS FROM ("file:///result.csv") AS row

match(a:Club{name:row.home})-[r:played\_with]-(b:Club{name:row.away})

set r+={status:row.status,home:row.home, away:row.away,score:[row.score\_\_001, row.score\_\_002], time:row.time, id:row.id, link:row.link };

-------------- Queires -----------------------

-- Display total number of matches played. ----

match(a:Club)-[r:played\_with{status:'played'}]->(b:Club) return count(r) AS Total\_Match\_Played;

-- Display details of all matches involved “Arsenal FC” ------

match(a:Club{name:'Arsenal FC'})-[r:played\_with{status:'played'}]-(b:Club) return r AS Arsenal\_FC\_MATCHES;

-- Display the number of matches “Liverpool FC” has lost. ---

match(a:Club{name:'Liverpool FC'})-[r:played\_with]->(b:Club)

WHERE r.score[0]<r.score[1] return r as result

UNION

match(a:Club)-[r:played\_with]->(b:Club{name:'Liverpool FC'})

WHERE r.score[0]>r.score[1] return r as result

---- Club name with the Home score

match(a:Club)-[r:played\_with]->(b:Club)

return a.name AS Club, sum(toInteger(head(r.score))) AS Home\_Score

--- Club name with the Away score

match(a:Club)<-[r:played\_with]-(b:Club)

return a.name AS Club, sum(toInteger(last(r.score))) AS Away\_Score

---- Display top five teams that have best scoring power. --

LOAD CSV WITH HEADERS FROM ("file:///club.csv") AS row

match(a:Club{name:row.club})-[r:played\_with]->(b:Club)

WITH a,sum(toInteger(head(r.score))) AS Home\_score

OPTIONAL MATCH (a)<-[r:played\_with]-(b:Club)

WITH a,Home\_score, sum(toInteger(last(r.score))) AS Away\_score

RETURN a.name, Home\_score+Away\_score AS Total\_Score

ORDER BY Total\_Score DESC LIMIT 5

--- Display top five teams that have poorest defending.

LOAD CSV WITH HEADERS FROM ("file:///club.csv") AS row

match(a:Club{name:row.club})-[r:played\_with]->(b:Club)

WITH a,sum(toInteger(last(r.score))) AS Home\_concede

OPTIONAL MATCH (a)<-[r:played\_with]-(b:Club)

WITH a,Home\_concede, sum(toInteger(head(r.score))) AS Away\_concede

RETURN a.name, Home\_concede+Away\_concede AS Total\_concede

ORDER BY Total\_concede DESC LIMIT 5

-- Display top five teams that have best winning records.

LOAD CSV WITH HEADERS FROM ("file:///club.csv") AS row

match(a:Club{name:row.club})-[r:played\_with]->(b:Club)

WHERE r.score[0]>r.score[1]

WITH a, count(r) as result1

match(a:Club)<-[r:played\_with]-(b:Club)

WHERE r.score[0]<r.score[1]

WITH a,result1, count(r) as result2

return a.name AS CLUB, result1+result2 AS WINNING\_RECORDS

ORDER BY WINNING\_RECORDS DESC LIMIT 5;

-- Display top five teams with best home winning records.

LOAD CSV WITH HEADERS FROM ("file:///club.csv") AS row

match(a:Club{name:row.club})-[r:played\_with]->(b:Club)

WHERE r.score[0]>r.score[1]

RETURN a.name, count(r) AS HOME\_WINNIG\_RECORDS

ORDER BY HOME\_WINNIG\_RECORDS DESC LIMIT 5;

-- Display top five teams with worst home losing recording.

LOAD CSV WITH HEADERS FROM ("file:///club.csv") AS row

match(a:Club{name:row.club})-[r:played\_with]->(b:Club)

WHERE r.score[0]<r.score[1]

RETURN a.name, count(r) AS WORST\_HOME\_RECORD

ORDER BY WORST\_HOME\_RECORD DESC LIMIT 5;

-- Which teams had most “draw”?

LOAD CSV WITH HEADERS FROM ("file:///club.csv") AS row

match(a:Club{name:row.club})-[r:played\_with]->(b:Club)

WHERE r.score[0]=r.score[1]

return a.name AS CLUB, count(r) AS DRAWS

ORDER BY DRAWS DESC ;

-- 10) display the team with most consecutive “wins”.

MATCH (a:Club {name:'Arsenal FC'})-[r:played\_with]-(:Club)

WITH ((CASE a.name WHEN r.home THEN 1 ELSE -1 END) \* (TOINT(r.score[0]) - TOINT(r.score[1]))) > 0 AS win, r

ORDER BY TOINT(r.time)

RETURN REDUCE(s = {max: 0, curr: 0}, w IN COLLECT(win) |

CASE WHEN w

THEN {

max: CASE WHEN s.max < s.curr + 1 THEN s.curr + 1 ELSE s.max END,

curr: s.curr + 1}

ELSE {max: s.max, curr: 0}

END

).max AS result;