CSY3024 – DATABASES 3

LEVEL 6

TERM I ASSIGNMENT

Diwas Lamsal

18406547

University of Northampton,

NAMI College

Video Demo Link:

https://northampton.mediaspace.kaltura.com/media/DIWAS+LAMSAL+-+18406547+-+CSY3024+%E2%80%93+DATABASES+3+Assignment+Term+I/0_kzikfm1b

2019

Table of Contents

ACRONYMS AND ABBREVIATIONS

BCNF	Boyce-Codd Normal Form
SQL	Structured Query Language
NoSQL	Not only SQL
RDB	Relational Database
ACID	Atomicity, Consistency, Isolation, Durability
JSON	JavaScript Object Notation
BSON	Binary Serialization Object Notation
CQL	Cypher Query Language
EPL	English Premier League
CSV	Comma-Separated Values
XML	Extensible Markup Language
URL	Uniform Resource Locator

PART I – LEARNING DIARY & REFLECTION

This section contains the weekly learning outcomes or reflections of the module. It briefly discusses the self-assessment of knowledge gained from this course also including any self-perceived weaknesses or strengths that were found. All the questions from lab exercises have been attempted and answered. The complete lab exercises are included in the Appendix and are referenced from each week with appropriate links.

Week 1 – Part I

Unable to attend the class, however, learned from peers that the class was a revision of previous relational databases. Normalization was revised along with introduction to other normal forms such as Boyce-Codd Normal Form (3.5NF), 4NF and 5NF. Self-studied about these different normal forms through online resources such as YouTube videos and websites to get a general idea. Brief descriptions of what was learned has been included in the appendix. (Week 1 - Part I – Appendix)

Week 1 – Part II

Started SQL Exercise which contained two parts and a total of 15 questions. The lesson involved creating database and tables for the two sets of questions which was done in XAMPP MySQL, and then entering relevant data to the tables. The questions required querying different kinds of data from the created tables. The lab exercises and other information has been included in the appendix. (Week 1 Part II – Appendix)

The lesson was fundamentally to implement previous SQL knowledge to solve real world-based problems. The question did not guide about specific ways to deal with a problem and instead, relevant techniques had to be figured by oneself. There are many ways to retrieve data from tables some of which could be more efficient or easier to write and some opposite. Different ways were tried to come up with required results. The tough part of this practical assessment was due to unfamiliarity of this kind of unguided question structure. In the past, till year II, the questions were easily structured and set up right after each lesson and it was easier to know which method can provide required results easily. Thinking about possible techniques to solve these questions was rather challenging and enjoyable experience. In order to answer the questions, different techniques were used such as sub-queries, UNIONs and INTERSECTs, and JOINs. However, only one kind of solution has been included for each question in the final source code.

Week 2

Week 3 introduced NoSQL and highlighted the need for transition from Relational Databases to NoSQL. The key advantage was found that NoSQL databases allow scaling out rather than scaling up. Other key characteristics of NoSQL such as it being non-relational, schema-less, horizontally scalable and being able to provide better fault tolerance were discussed. At this point, it was unclear as to what this new kind of language and database environment will feel and the urge to learn was high. Different categories of NoSQL were introduced which is mentioned in the appendix. All the questions from student activities have also been answered and are compiled in the appendix with relevant citation using Harvard Referencing. (Week 2 – Appendix)

Week 3

Week 3 introduced about graph databases and Neo4j in a bit more depth. The concepts of nodes, relationships, attributes and labels were discussed in a graph database. Other concepts such as how querying and indexing differs from the relational database model in graph databases were also discussed. The benefits of Neo4j and how it revolutionized a new concept in databases with the introduction of Cypher Query Language were conversed. It is now known that Cypher is a very powerful language which can perform a lot more from a lot less of code in comparison to SQL. It is easy to learn with a lot of available resources, provides good performance, and can be used to develop scalable applications in relatively smaller amount of time. At this point, more curiosity was developing for graph databases and cypher language in particular as not much of code was practiced. (Week 3 – Appendix)

Week 4

Week 4 was introduction to Cypher Query Language as the SQL for graph databases. It was expressed how Cypher was a very simple yet strong language which focuses more on the information to get rather than how to get it. Other basic syntaxes of Cypher such as creating and deleting nodes, relationships and labels were also discussed. The key points discussed in this week were the use cases of nodes, relationships, labels, variables, properties and operators. The student activities are included in the appendix. At this point, the lesson was feeling a bit overwhelming. It was also partly because some commands such as deleting nodes was not working on the computer probably because of version or other issues (could not be solved even after browsing the internet). The research question has also been answered at the bottom of week 4 appendix including relevant citations using Harvard referencing. (Week 4 – Appendix)

Week 5

Week 5 was more focused on Cypher commands and had more student activities than theoretical aspects. The upcoming weeks are almost structured in a similar way and thus have more student activities. At this point, the earlier overwhelming feeling that this new language could be very tough to learn was now gone. It was in fact feeling very easy to use and write than SQL. The overwhelming feeling in the past was due to new symbols such as ->, new ways to declare variables and a very different syntax style from SQL. However, this was solved in week 5 when a lot of code was practiced, and a fresh installation of Neo4j was used to solve the delete issue from week 5. All the activities from week 6 is included in the appendix. (Week 5 – Appendix)

Week 6

By the week 6, the commands were getting very familiar and learning new ways to deal with nodes, relationships, labels, etc. was making sense. It was very interesting to learn how to create unique nodes and relationships using MERGE. More details about what was learned from this week's class and student activities are included in the appendix. (Week 6 – Appendix)

Week 7

Week 7 mainly focused on CREATE UNIQUE commands which were kind of similar in concept to the previously learned MERGE command. It was learned that CREATE UNIQUE commands are especially used for creating unique relationships and are like a mixture of using MATCH and CREATE clauses. This command is also used to create only those nodes or relationships that have not previously been created. However, it was also mentioned that the usage of CREATE UNIQUE was removed after Cypher 3.2, we can still use the command, but the query will get demoted to using Cypher version 3.1. (Week 7 – Appendix)

Week 8

Week 8 can arguably be referred to as the most important chapter for beginning the assignment. This week involves loading CSV files into Neo4j and working with the data in CSV files. This week was basically dealt with loading different CSV files provided in the NILE to the local database. The processes involved moving the CSV files to the import directory of selected graph database and then using the LOAD CSV command in the Neo4j browser. The processes will be displayed in more detail in the Week 8 Appendix. Other lessons such as OPTIONAL MATCH, MATCH and finding shortest paths between nodes is also included.

Week 10

Although week 9 was mentioned in NILE about assignment session, week 10 was the lesson taught in the class. This week contained lessons about filtering and goes more in-depth about using the WHERE clause. Different Boolean operations as well as other ways to filter the data to be displayed were focused on. Aggregate functions and lists were also discussed briefly. The full details about the lab activities are included in the Appendix. (Week 10 - Appendix)

Self-Reflection & Summary

Self-Perceived Strengths & Weaknesses

The table below contains the weaknesses and strengths discovered while going through the course of Database 3.

Table 1 – Strengths and Weaknesses

Strengths	• Able to learn and grasp new concepts related to database • Able to think and reuse some previously learned concepts of SQL such as ways to query or manipulate data. • Able to learn and get familiar with new language like CQL relatively quickly. • Able to think out of the box in order to come up with original solutions. • Can search and find resources somewhat efficiently from the internet.

Weaknesses	• Feel overwhelmed at the beginning of a new lesson. • Sometimes trying to think out of the box leads to unnecessarily time consuming and wrong outcomes. • Introduction to new symbols usually feels overwhelming at first thinking many of it should be memorized. • The concept of importing CSV can sometimes still feel intimidating and needs more practice. • Slight overconfidence which led to skipping some lessons during the class and had to revise them later.

Summary

To summarize the main learning activities from the first term of Database III, we have to look back to Week 2 as the most important week as that was when the transition of relational databases to NoSQL was discussed. Although relational databases do a very good job at a smaller level where large data sets and big data is not necessary, in the current context where data is very interconnected to each other, they do a very poor job. Other databases called NoSQL were introduced to make up for this weakness. Key characteristics of NoSQL such as it being non-relational, schema-less, horizontally scalable and being able to provide better fault tolerance were discussed. The exercises that required research and proper Harvard referencing gave further understanding that how NoSQL works better than SQL. There could also be certain situations where SQL would still do a better job. NoSQL can be structured to fit exact user needs and can produce market ready applications sooner. It also removes the need of using JOINs which could be argued as the beauty of Relational SQL. The later weeks involved more about cypher commands and how to create and query databases in Neo4j using CQL. Certain aspects of the assignment also required further research. The upcoming chapters from the following term (Cypher 7 and 8) were studied and referred to for certain aspects of the assignment such as UNION whereas the rest was referenced from the online Neo4j documentation.

PART II – GRAPH DATABASE FOR GIVEN DATASET

Designing and Creating the Graph Database

The assignment brief comes with two other files – one CSV file that contains all the data for the graph database dataset and a text file that contains relevant information about reading the data from the provided dataset. The first step before querying the database will be to create a separate graph database for the assignment, load the CSV file to the database and then begin the querying.

Creating the Database

A new fresh database is created in order to import all the data from the CSV file. The database is empty and is ready for adding new data.

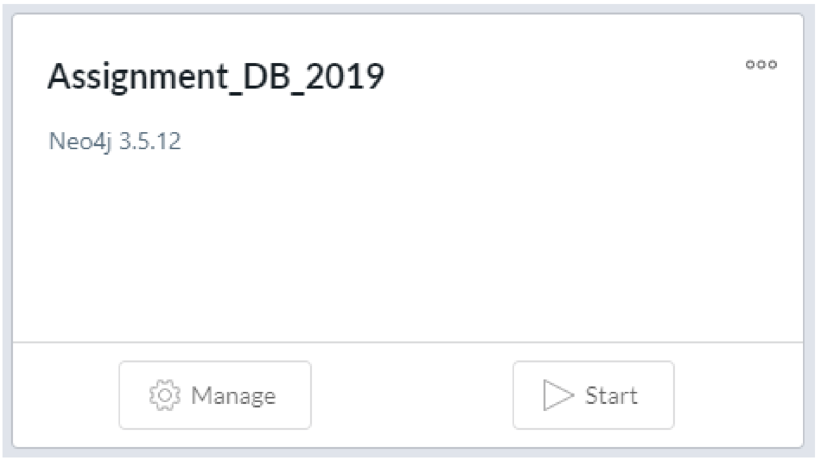
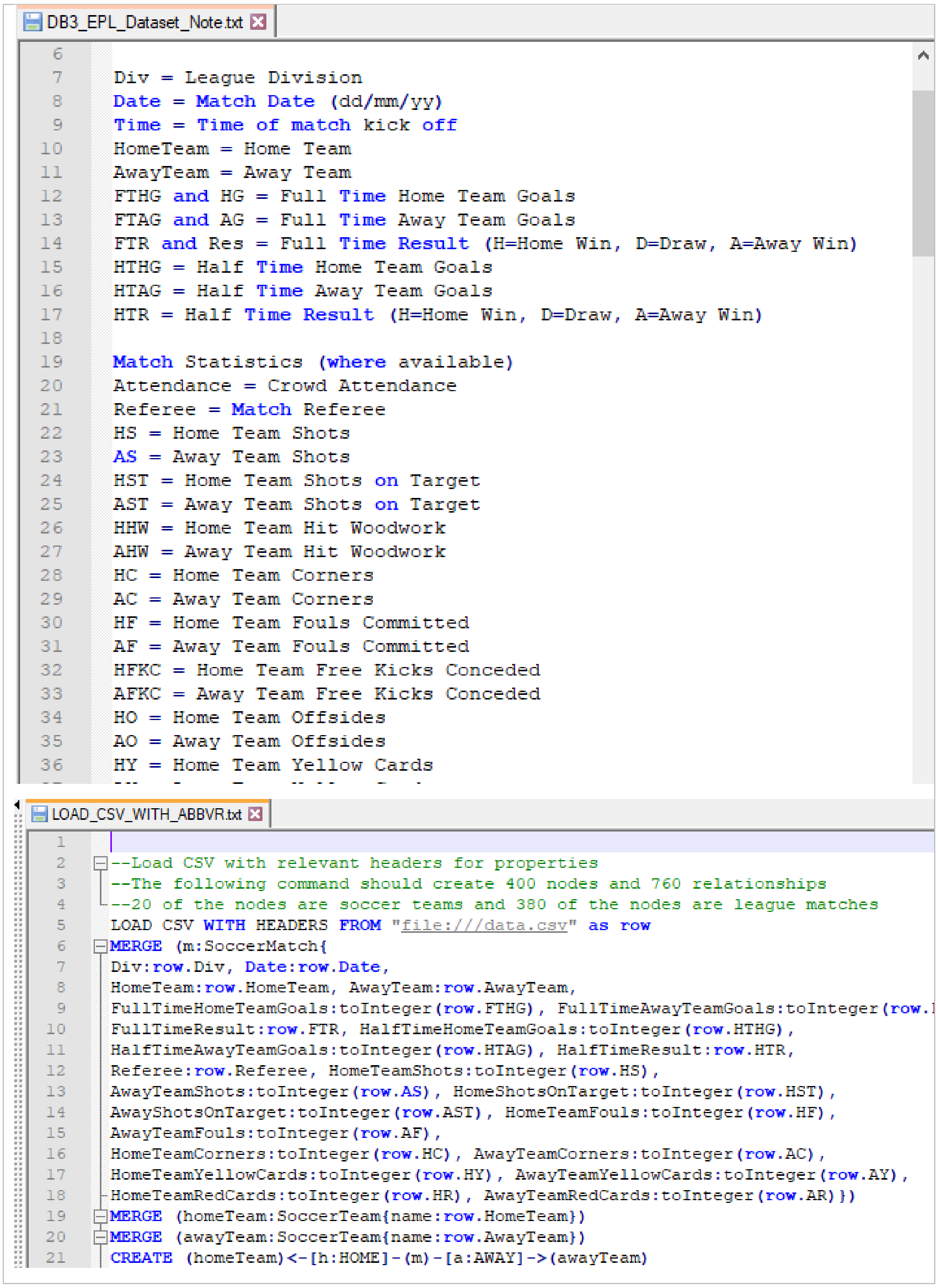


Figure 1 – New Empty Database for Assignment

Importing the CSV Data

The first step to import the CSV is to make the code for retrieving relevant headers and creating correct nodes with correct data. The LOAD CSV command and other lessons from week 9 will be very useful in this task such as copying the CSV file to the import folder in the database. The relevant information was extracted from the provided dataset note text file to set readable properties instead of Abbreviations. The CSV was loaded with headers, and the properties were set accordingly referring to the text file. As mentioned in the screenshot below, the command creates a total of 400 nodes 380 of which are the league matches and 20 of which are the teams that played those matches. The relationship is initialized while loading the CSV such that the teams are also created and for each match, the AWAY and HOME relationship determines which team is the home team and which is the away team.



```

Div = League Division
Date = Match Date (dd/mm/yy)
Time = Time of match kick off
HomeTeam = Home Team
AwayTeam = Away Team
FTHG and HG = Full Time Home Team Goals
FTAG and AG = Full Time Away Team Goals
FTR and Res = Full Time Result (H=Home Win, D=Draw, A=Away Win)
HTHG = Half Time Home Team Goals
HTAG = Half Time Away Team Goals
HTR = Half Time Result (H=Home Win, D=Draw, A=Away Win)

Match Statistics (where available)
Attendance = Crowd Attendance
Referee = Match Referee
HS = Home Team Shots
AS = Away Team Shots
HST = Home Team Shots on Target
AST = Away Team Shots on Target
HHW = Home Team Hit Woodwork
AHW = Away Team Hit Woodwork
HC = Home Team Corners
AC = Away Team Corners
HF = Home Team Fouls Committed
AF = Away Team Fouls Committed
HFKC = Home Team Free Kicks Conceded
AFKC = Away Team Free Kicks Conceded
HO = Home Team Offsides
AO = Away Team Offsides
HY = Home Team Yellow Cards
```

```

--Load CSV with relevant headers for properties
--The following command should create 400 nodes and 760 relationships
--20 of the nodes are soccer teams and 380 of the nodes are league matches
LOAD CSV WITH HEADERS FROM "file:///data.csv" as row
MERGE (m:SoccerMatch{
Div:row.Div, Date:row.Date,
HomeTeam:row.HomeTeam, AwayTeam:row.AwayTeam,
FullTimeHomeTeamGoals:toInteger(row.FTHG), FullTimeAwayTeamGoals:toInteger(row.
FullTimeResult:row.FTR, HalfTimeHomeTeamGoals:toInteger(row.HTHG),
HalfTimeAwayTeamGoals:toInteger(row.HTAG), HalfTimeResult:row.HTR,
Referee:row.Referee, HomeTeamShots:toInteger(row.HS),
AwayTeamShots:toInteger(row.AS), HomeShotsOnTarget:toInteger(row.HST),
AwayShotsOnTarget:toInteger(row.AST), HomeTeamFouls:toInteger(row.HF),
AwayTeamFouls:toInteger(row.AF),
HomeTeamCorners:toInteger(row.HC), AwayTeamCorners:toInteger(row.AC),
HomeTeamYellowCards:toInteger(row.HY), AwayTeamYellowCards:toInteger(row.AY),
HomeTeamRedCards:toInteger(row.HR), AwayTeamRedCards:toInteger(row.AR)})
MERGE (homeTeam:SoccerTeam{name:row.HomeTeam})
MERGE (awayTeam:SoccerTeam{name:row.AwayTeam})
CREATE (homeTeam)<-[h:HOME]-(m)-[a:AWAY]->(awayTeam)
```

Figure 2 – Loading the CSV with readable properties

Code for Loading the Dataset:

```
//Load CSV with relevant headers for properties
//The following command should create 400 nodes and 760 relationships
//20 of the nodes are soccer teams and 380 of the nodes are league matches
LOAD CSV WITH HEADERS FROM "file:///data.csv" AS row
MERGE (m:SoccerMatch{
Div:row.Div, Date:row.Date,
HomeTeam:row.HomeTeam, AwayTeam:row.AwayTeam,
FullTimeHomeTeamGoals:toInteger(row.FTHG),
FullTimeAwayTeamGoals:toInteger(row.FTAG),
FullTimeResult:row.FTR, HalfTimeHomeTeamGoals:toInteger(row.HTHG),
HalfTimeAwayTeamGoals:toInteger(row.HTAG), HalfTimeResult:row.HTR,
Referee:row.Referee, HomeTeamShots:toInteger(row.HS),
AwayTeamShots:toInteger(row.AS), HomeShotsOnTarget:toInteger(row.HST),
AwayShotsOnTarget:toInteger(row.AST), HomeTeamFouls:toInteger(row.HF),
AwayTeamFouls:toInteger(row.AF),
HomeTeamCorners:toInteger(row.HC), AwayTeamCorners:toInteger(row.AC),
HomeTeamYellowCards:toInteger(row.HY), AwayTeamYellowCards:toInteger(row.AY),
HomeTeamRedCards:toInteger(row.HR), AwayTeamRedCards:toInteger(row.AR)})
MERGE (homeTeam:SoccerTeam{name:row.HomeTeam})
MERGE (awayTeam:SoccerTeam{name:row.AwayTeam})
CREATE UNIQUE (homeTeam)<-[h:HOME]-(m)-[a:AWAY]->(awayTeam)
```

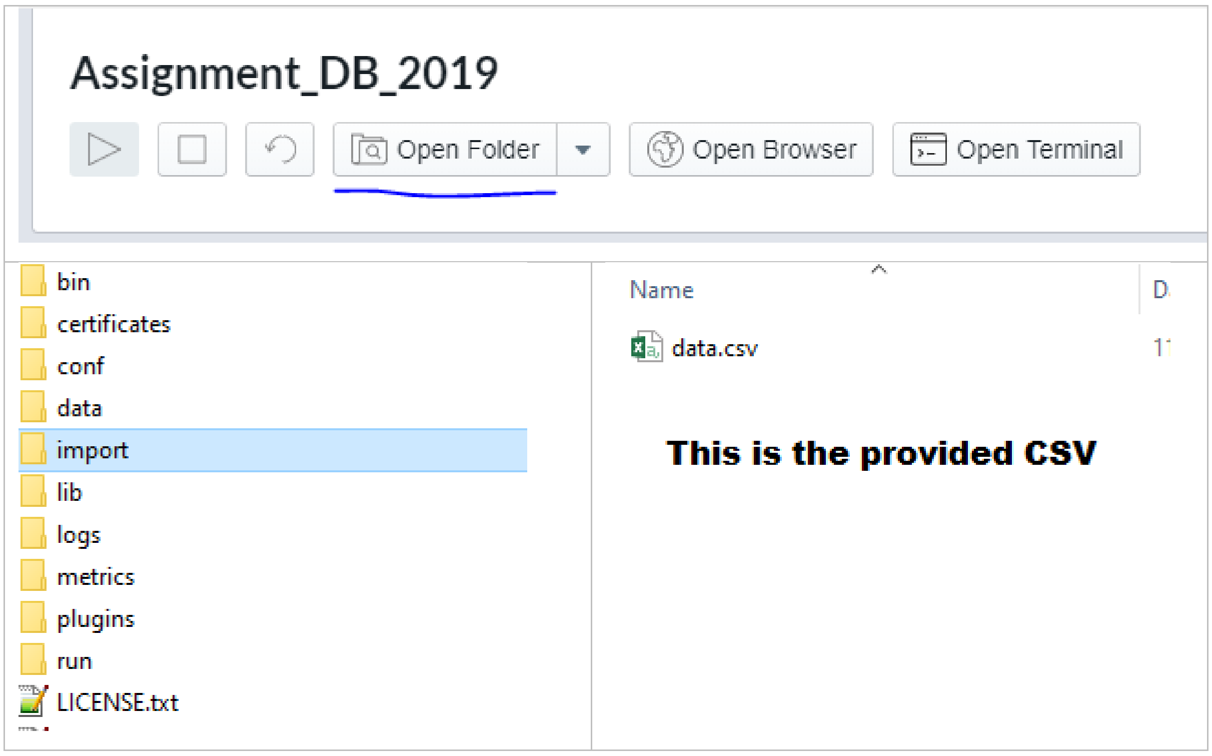
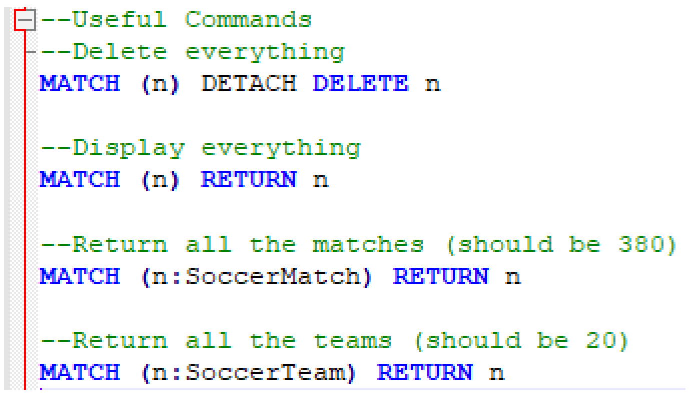


Figure 3 – Copying the CSV file to the import folder

As mentioned earlier, the CSV file included in the assignment brief was copied to the import folder of the database before LOAD operation.



```
--Useful Commands
--Delete everything
MATCH (n) DETACH DELETE n

--Display everything
MATCH (n) RETURN n

--Return all the matches (should be 380)
MATCH (n:SoccerMatch) RETURN n

--Return all the teams (should be 20)
MATCH (n:SoccerTeam) RETURN n
```

Figure 4 – Commands for efficiency

Some commands were always kept open in a text file which would allow copy-pasting and reduce time consumption as well as increase efficiency. The green texts indicate what the correct results should be while running these commands.

Database Model

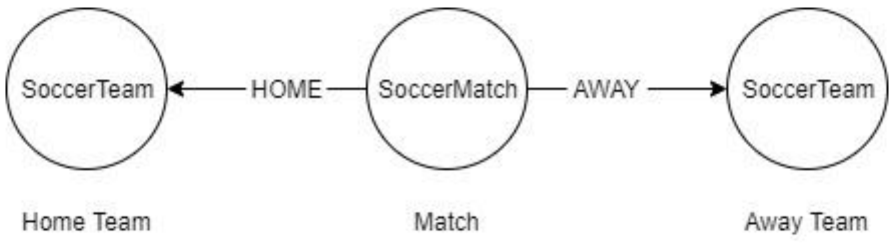


Figure 5 – Database Model

Nodes and Labels

There are basically two types of nodes that were created. The nodes are SoccerMatch node which contains all the information about the EPL games and the SoccerTeam node which contains the name of soccer teams involved in the EPL season. Both of these nodes are created when initializing the database by loading the CSV file. There are 380 SoccerMatch nodes which contain all the required details about the 380 league games. These nodes can be retrieved using the command "MATCH (n:SoccerMatch) RETURN n". The SoccerTeam nodes contain the names of each EPL team and there are 20 nodes which represent the 20 teams involved. In a similar manner, these nodes can be retrieved using the command "MATCH (n:SoccerTeam) RETURN n". These nodes have the labels SoccerMatch for all the EPL games and SoccerTeam for all the teams that have played in the EPL for provided season.

Relationships

While initializing the database from provided CSV files, a relationship was also formed between each SoccerMatch node with two SoccerTeam nodes. One SoccerTeam node would have the "HOME" relationship while the other would have "AWAY" relationship. These relationships are created based on whether the team is home team for the relevant soccer game or the away team. For example, in a match between Arsenal and Man City, if Arsenal is the home team and Man City is the away team, the match node would have "HOME" relationship with Arsenal and "AWAY" relationship with Man City. As such, for each match, there will be two relationships hence leading to 760 total relationships. This can also be confirmed by using the command "MATCH (:SoccerTeam)<-[r:HOME|:AWAY]-(:SoccerMatch) RETURN COUNT(r)".

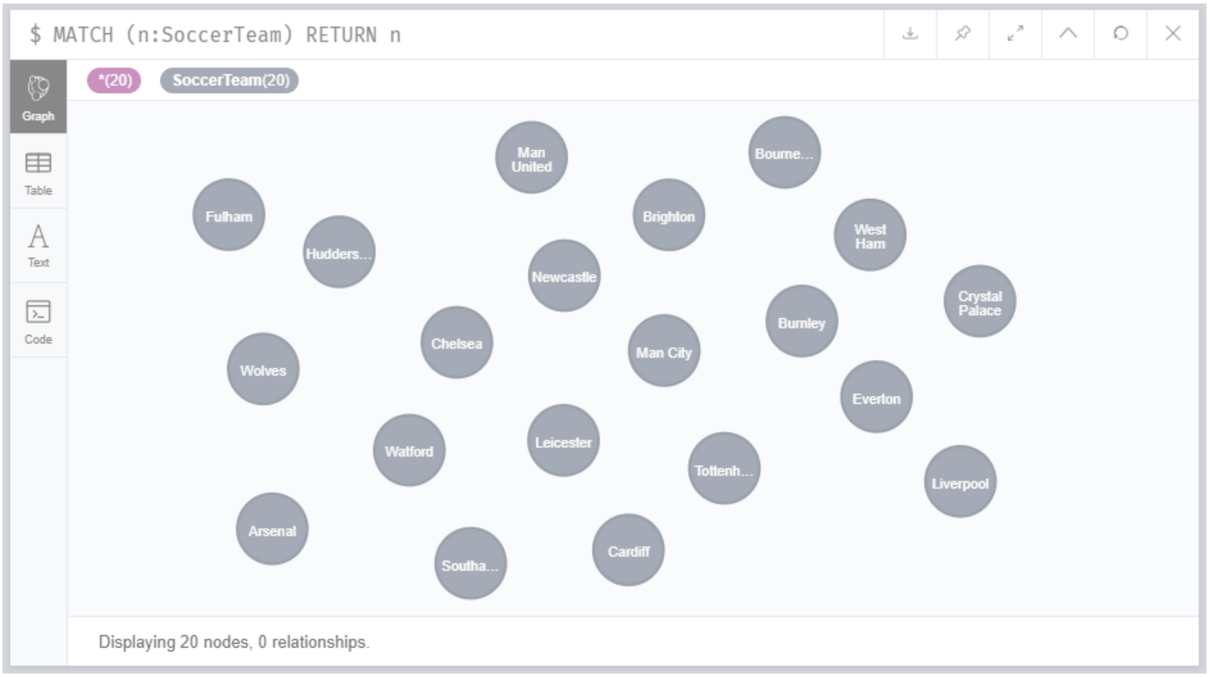
Querying the Database to Answer the Questions

1) Show all the EPL teams involved in the season.

- A simple query should be able to achieve this task. The EPL teams were created earlier using home and away relationships in every football match as SoccerTeam nodes. We can retrieve these nodes.

Answer: 20 teams displayed in the graph.

Code: MATCH (n:SoccerTeam) RETURN n



```
$ MATCH (n:SoccerTeam) RETURN n
```

Displaying 20 nodes, 0 relationships.

Figure 6 – Display all EPL Teams

This displays the 20 nodes of the involved EPL teams. These have been produced with the assumption that the EPL teams have played at least one match. The results can also be viewed in text format.

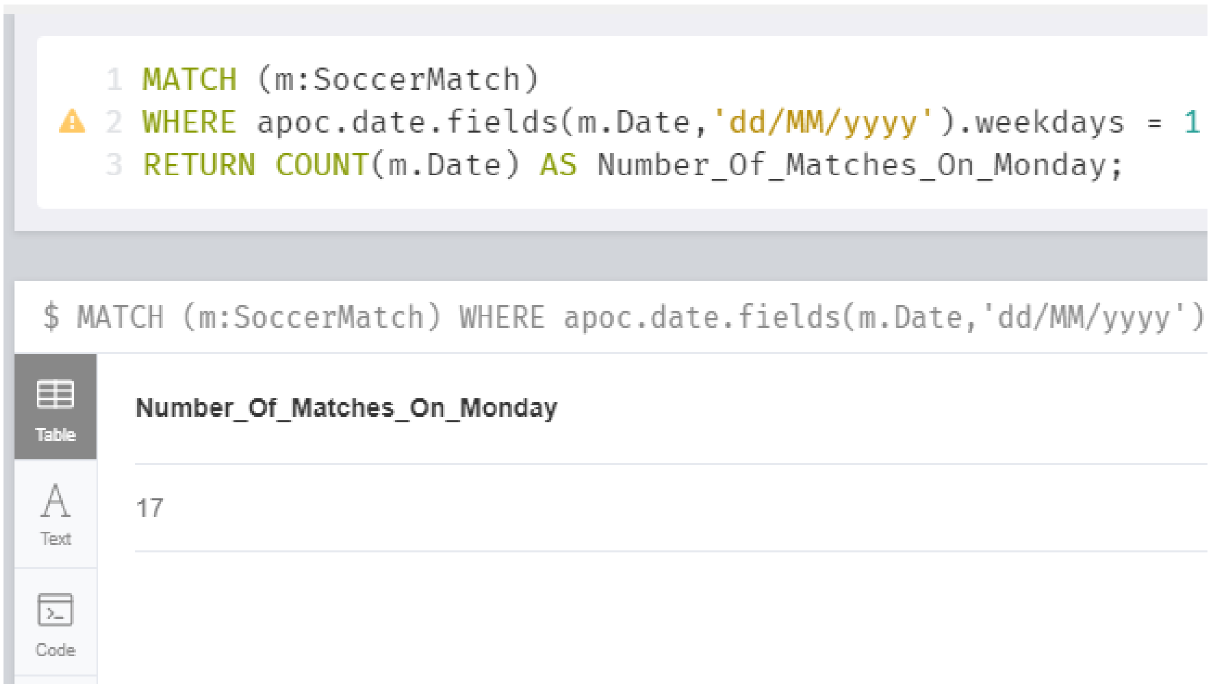
2) How many matches were played on Mondays?

- Apoc library (Neo4j Graph Database Platform, 2019) provides a way to recognize non-native date formats with the function apoc.date.fields. The first argument is the date and the second one is format. The result data contains a weekdays property. This property contains numeric values from 1-7 where 7 is Sunday. We can restrict our results to only contain data that has 1 as weekdays value (1 for Monday). We can then count the total number of matches played on Monday with the use of COUNT.

Answer: 17 matches.

Code:

```
// Get all the matches with Mondays (Weekdays as 1)
MATCH (m:SoccerMatch)
WHERE apoc.date.fields(m.Date,'dd/MM/yyyy').weekdays = 1
RETURN COUNT(m.Date) AS Number_Of_Matches_On_Monday;
```



```
MATCH (m:SoccerMatch)
WHERE apoc.date.fields(m.Date,'dd/MM/yyyy').weekdays = 1
RETURN COUNT(m.Date) AS Number_Of_Matches_On_Monday;
```

```
$ MATCH (m:SoccerMatch) WHERE apoc.date.fields(m.Date,'dd/MM/yyyy')
```

Number_Of_Matches_On_Monday
17

Figure 7 – Matches on Mondays

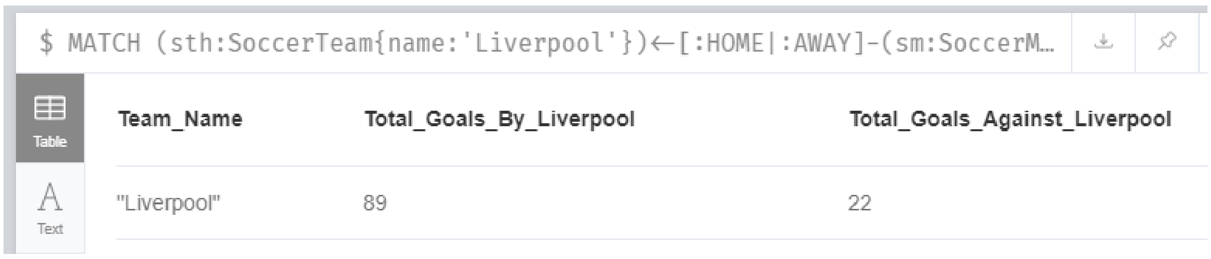
3) Display the total number of goals "Liverpool" had scored and conceded in the season.

- This can be done by first retrieving all the matches played by Liverpool. The total for all the games where Liverpool scored and conceded will then be retrieved using WITH and CASE. CASE allows selecting specific cases of when Liverpool is the home or away team and to declare necessary variables using WITH. It will be clearer from the provided screenshot.

Answer: Goals scored by Liverpool is 89 and goals conceded by Liverpool is 22.

Code:

```
//Retrieve the matches that involve Liverpool as home or away team
MATCH (liv:SoccerTeam{name:'Liverpool'})<-[:HOME|:AWAY]-(sm:SoccerMatch)
//Use with clause to rename the team name and to get the total Liverpool goals and opponent goals according to whether the team and game are home or away
WITH liv.name AS Team_Name,
(CASE WHEN sm.HomeTeam='Liverpool' THEN sm.FullTimeHomeTeamGoals ELSE
sm.FullTimeAwayTeamGoals END) AS Total_Goals_By_Team,
(CASE WHEN sm.HomeTeam='Liverpool' THEN sm.FullTimeAwayTeamGoals ELSE
sm.FullTimeHomeTeamGoals END) AS Total_Goals_Against_Team
//Display the results
//The use of SUM will provide total number of goals by Liverpool or opponents
RETURN Team_Name, SUM(Total_Goals_By_Team) AS Total_Goals_By_Liverpool,
SUM(Total_Goals_Against_Team) AS Total_Goals_Against_Liverpool
```



```
$ MATCH (sth:SoccerTeam{name:'Liverpool'})←[:HOME|:AWAY]-(sm:SoccerM…
```

Team_Name	Total_Goals_By_Liverpool	Total_Goals_Against_Liverpool
"Liverpool"	89	22

Figure 8 – Goals scored and conceded by Liverpool

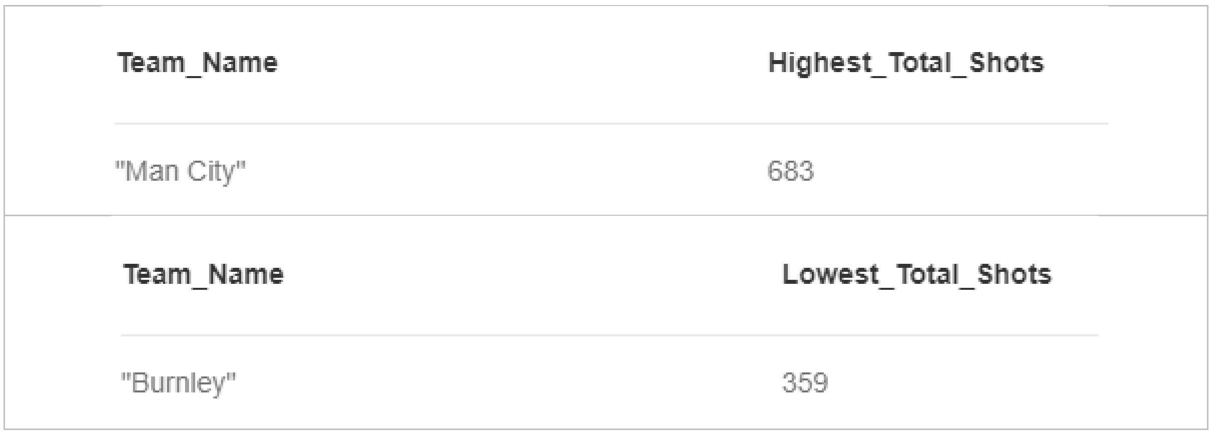
4) Which teams have the most and least shots in the season?

- As in many other questions, this question also requires addressing of the home or away team. A similar approach to the question 3 has been followed in the question 4. We can match the data according to the relationship of the team with a match, and then retrieve home team's total shots or away team's total shots accordingly. The most and least shots for a team can be retrieved separately by changing the order of data in ascending or descending order. Displaying the data in descending order and limiting the results to 1 would display the team with the greatest number of shots and vice versa for ascending.

Answer: Highest shots by Man City with 683 total shots and lowest shots by Burnley with 359 total shots.

Code:

```
//Retrieve the teams that have played the league matches
MATCH (st:SoccerTeam)<-[:HOME|:AWAY]-(sm:SoccerMatch)
//Use with clause to rename the team name and to get the total team shots according to whether
the team and game are home or away
WITH st.name AS Team_Name, (CASE WHEN sm.HomeTeam=st.name THEN
sm.HomeTeamShots ELSE sm.AwayTeamShots END) AS Total_Shots_By_Team
//Display the results with descending order for getting highest shots by a team
//The use of SUM will provide total number of shots by a particular team
//It can then be sorted by using ORDER BY clause
RETURN Team_Name, SUM(Total_Shots_By_Team) AS Highest_Total_Shots
ORDER BY Highest_Total_Shots DESC //Change to ASC to get lowest team shots
//Only display one result
LIMIT 1
```



Team_Name	Highest_Total_Shots
"Man City"	683

Team_Name	Lowest_Total_Shots
"Burnley"	359

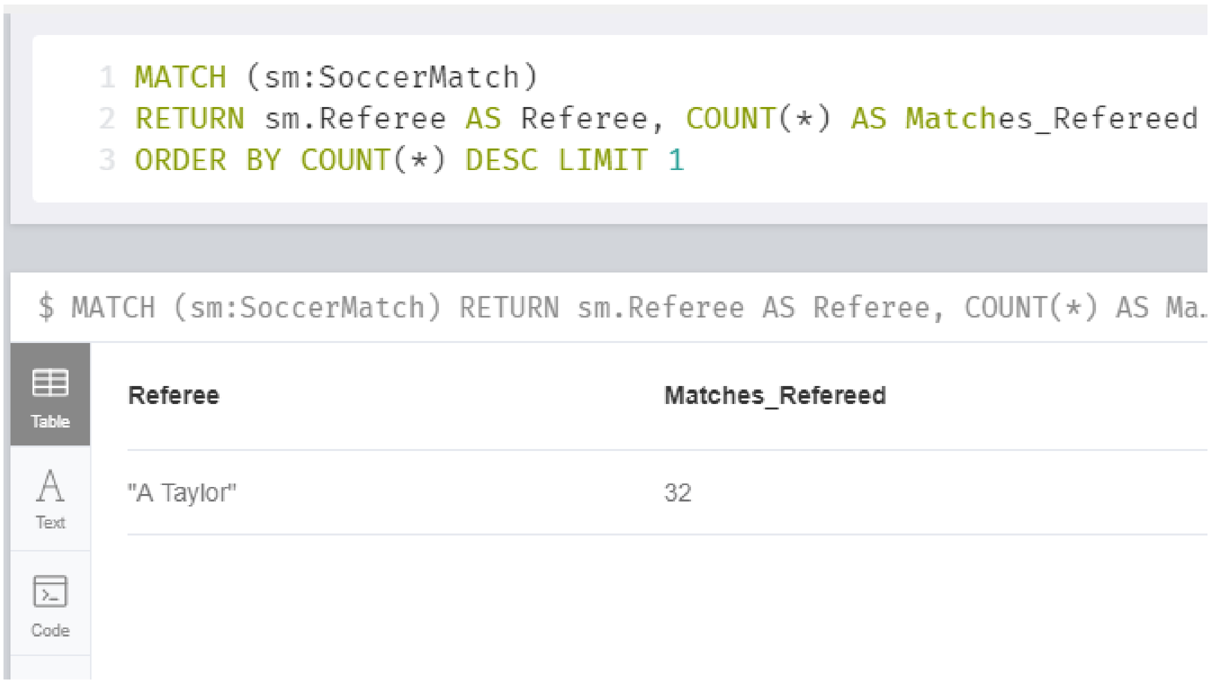
5) Who refereed the most matches?

- This can be retrieved using a simple query. First, we retrieve all the matches and referring to the cypher 6 notes, we can use COUNT function to group the number of records for each referee. This gives us a list of referee and total matches refereed by them. To find the one who refereed the most matches, we can simply order them in descending order using ORDER BY and DESC commands. The results can be limited to 1 to display only the highest record. This can be done with LIMIT.

Answer: “A Taylor” who refereed 32 matches.

Code:

```
// Get all the soccer matches
MATCH (sm:SoccerMatch)
// Group the matches by referee and their game count
RETURN sm.Referee AS Referee, COUNT(*) AS Matches_Refereed
// Order the data in descending order for referee with highest number of games
// To be displayed at the top. Limit 1 so that the highest number is only displayed.
ORDER BY COUNT(*) DESC LIMIT 1
```



```
MATCH (sm:SoccerMatch)
RETURN sm.Referee AS Referee, COUNT(*) AS Matches_Refereed
ORDER BY COUNT(*) DESC LIMIT 1
```

```
$ MATCH (sm:SoccerMatch) RETURN sm.Referee AS Referee, COUNT(*) AS Ma…
```

Referee	Matches_Refereed
"A Taylor"	32

Figure 9 – Refereed Most Matches

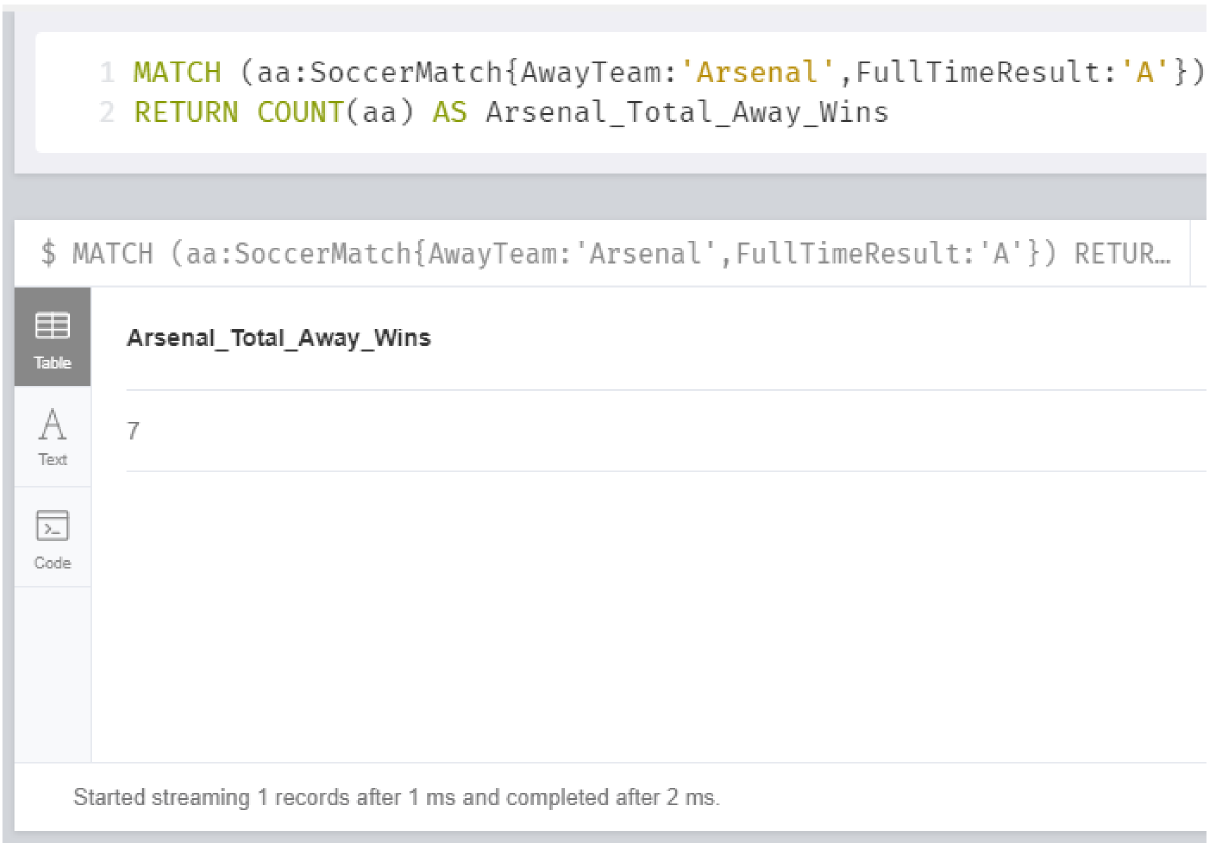
6) How many matches "Arsenal" won as the away team?

- For retrieving the results, it would be easier to first get all the soccer matches with "Arsenal" as the away team with the MATCH clause. It can then be manipulated either using WHERE clause or directly matching the nodes where the full time result of the game is won by the away team. The full-time results are represented by FullTimeResult in our database (was FTR in the CSV).

Answer: "Arsenal" won 7 matches as an away team.

Code:

```
// Match the results such that Arsenal is the away team
// And the winner of the game is the away team, A represents away win
MATCH (aa:SoccerMatch{AwayTeam:'Arsenal', FullTimeResult:'A'})
RETURN COUNT(aa) AS Arsenal_Total_Away_Wins
```



```
MATCH (aa:SoccerMatch{AwayTeam:'Arsenal',FullTimeResult:'A'})
RETURN COUNT(aa) AS Arsenal_Total_Away_Wins
```

```
$ MATCH (aa:SoccerMatch{AwayTeam:'Arsenal',FullTimeResult:'A'}) RETUR…
```

Arsenal_Total_Away_Wins
7

Started streaming 1 records after 1 ms and completed after 2 ms.

Figure 10 – Matches Arsenal Won as Away Team

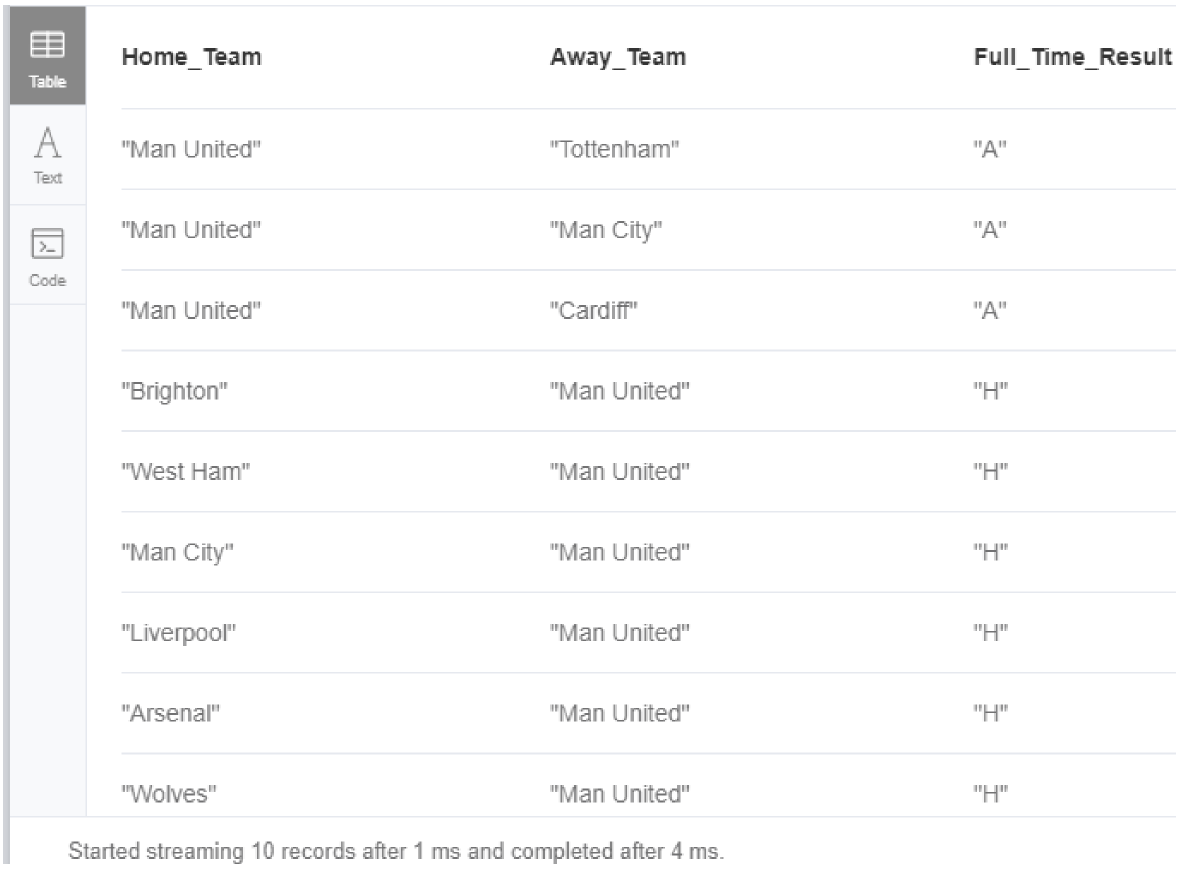
7) Display all the matches that "Man United" lost.

Although this question could seem similar to the previous one, it needs to account both the losses for Man United while being an away team, as well as the home team. There could be several approaches, but the approach used here is to retrieve two sets of results where Man United lost as the home team and the away team separately. These results could then be combined using UNION. The results can be either filtered or fully displayed.

Answer: 10 games displayed in the screenshots.

Code Using Projection (More Readable):

```
// Retrieve the results when Man United is the home team but the away team won
MATCH (lh:SoccerMatch)
WHERE lh.HomeTeam = 'Man United' AND lh.FullTimeResult = 'A'
// Display the games where Man United lost as the home team with projection
RETURN lh.HomeTeam AS Home_Team, lh.AwayTeam AS Away_Team, lh.FullTimeResult
AS Full_Time_Result
// Combine the results with UNION
UNION
// Retrieve the results when Man United is the away team but the home team won
MATCH (la:SoccerMatch)
WHERE la.AwayTeam = 'Man United' AND la.FullTimeResult = 'H'
// Display the games where Man United lost as the away team with projection
RETURN la.HomeTeam AS Home_Team, la.AwayTeam AS Away_Team, la.FullTimeResult
AS Full_Time_Result
```



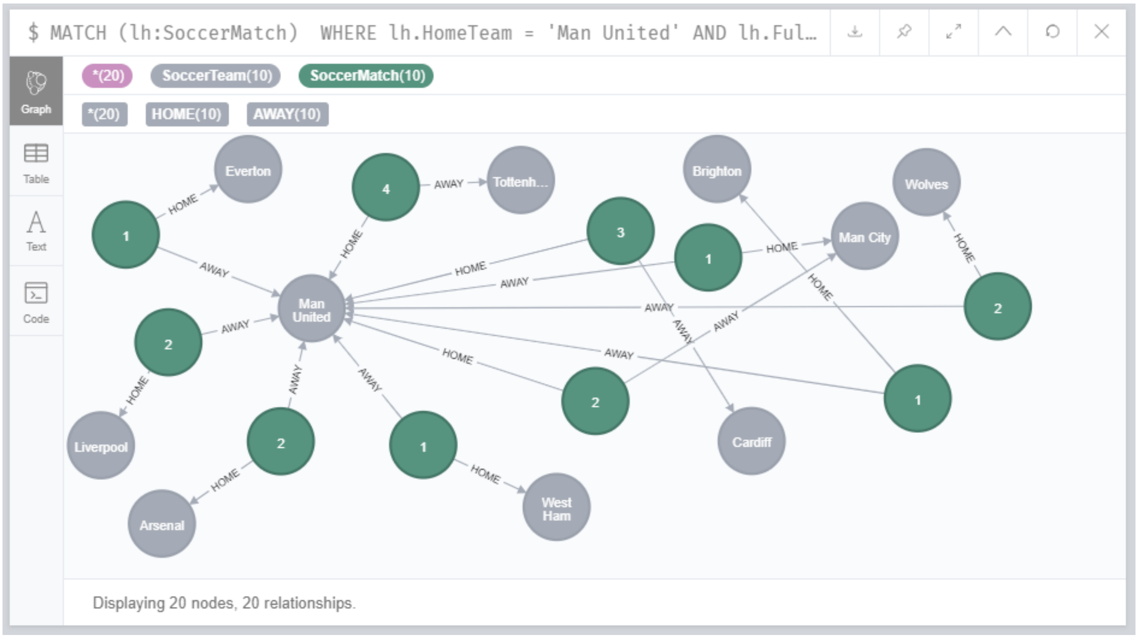
Home_Team	Away_Team	Full_Time_Result
"Man United"	"Tottenham"	"A"
"Man United"	"Man City"	"A"
"Man United"	"Cardiff"	"A"
"Brighton"	"Man United"	"H"
"West Ham"	"Man United"	"H"
"Man City"	"Man United"	"H"
"Liverpool"	"Man United"	"H"
"Arsenal"	"Man United"	"H"
"Wolves"	"Man United"	"H"

Started streaming 10 records after 1 ms and completed after 4 ms.

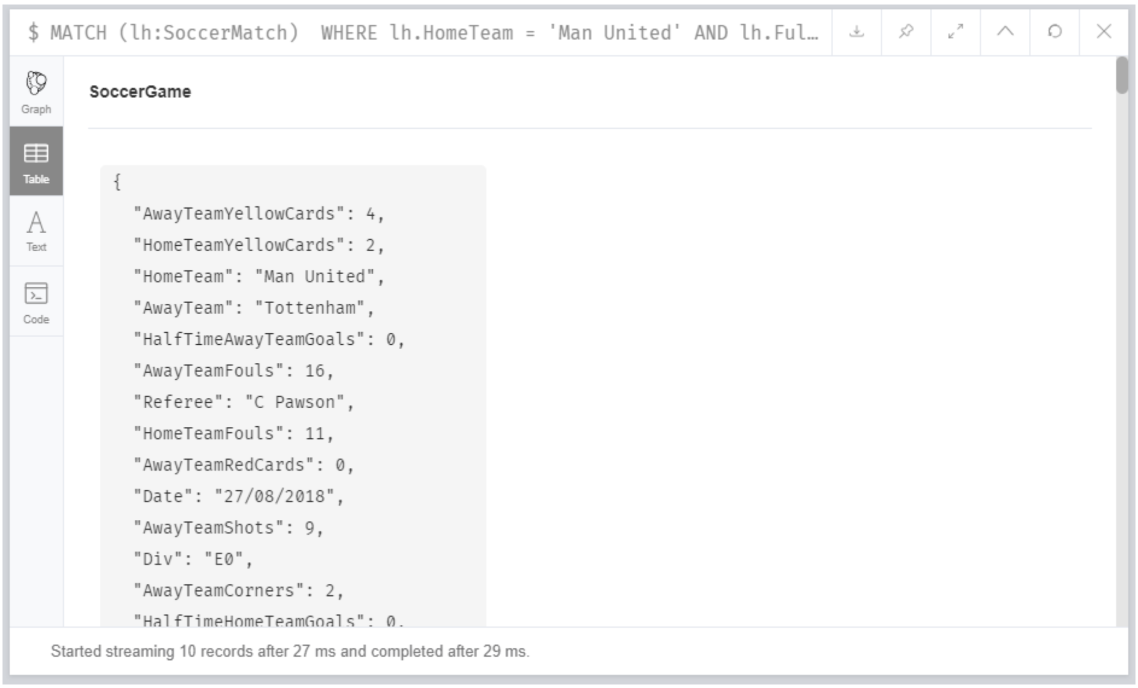
Figure 11 – All Matches that Man United Lost (Projected)

Code Without Projection (Display whole Soccer Match nodes):

```
// Retrieve the results when Man United is the home team but the away team won
MATCH (lh:SoccerMatch)
WHERE lh.HomeTeam = 'Man United' AND lh.FullTimeResult = 'A'
// Display the games where Man United lost as the home team without projection
RETURN lh AS SoccerGame // Combine the results with UNION
UNION
// Retrieve the results when Man United is the away team but the home team won
MATCH (la:SoccerMatch)
WHERE la.AwayTeam = 'Man United' AND la.FullTimeResult = 'H'
// Display the games where Man United lost as the away team without projection
RETURN la AS SoccerGame
```


```
$ MATCH (lh:SoccerMatch) WHERE lh.HomeTeam = 'Man United' AND lh.Ful…
```

Displaying 20 nodes, 20 relationships.


```
$ MATCH (lh:SoccerMatch) WHERE lh.HomeTeam = 'Man United' AND lh.Ful…
```

SoccerGame

```
{
  "AwayTeamYellowCards": 4,
  "HomeTeamYellowCards": 2,
  "HomeTeam": "Man United",
  "AwayTeam": "Tottenham",
  "HalfTimeAwayTeamGoals": 0,
  "AwayTeamFouls": 16,
  "Referee": "C Pawson",
  "HomeTeamFouls": 11,
  "AwayTeamRedCards": 0,
  "Date": "27/08/2018",
  "AwayTeamShots": 9,
  "Div": "E0",
  "AwayTeamCorners": 2,
  "HalfTimeHomeTeamGoals": 0,
```

Started streaming 10 records after 27 ms and completed after 29 ms.

Figure 12 - All Matches that Man United Lost (Not Projected)

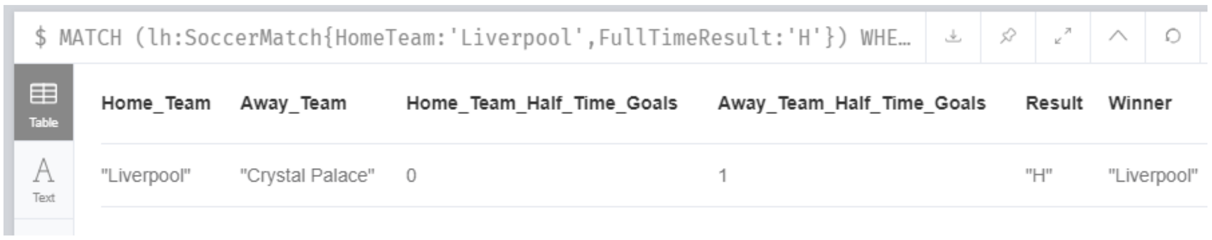
8) Display all matches that "Liverpool" won but were down in the first half.

- A similar technique to question 7 can be used for retrieving this result. Two sets of data can be retrieved using MATCH for Liverpool's home and away games. This can then be displayed together using WITH and UNION. The greater than less than operators are used however in this case to retrieve only those matches where Liverpool were down in the first half. The first half records are denoted by HalfTimeHomeTeamGoals and HalfTimeAwayTeamGoals. The home half time goals should be less than away goals when Liverpool is the home team and the exact opposite when Liverpool is the away team. Similar to the previous question, two formats of output are displayed with one being more readable and the other displaying everything.

Answer: One occurrence was found between Liverpool and Crystal Palace where Liverpool was the home team. Crystal Palace had scored one goal before the half time. The game later ended up being 4-3 in the favour of Liverpool.

Code Using Projection (More Readable):

```
//Retrieve the results when Liverpool is the Home Team and winner is the home team
MATCH (lh:SoccerMatch{HomeTeam:'Liverpool',FullTimeResult:'H'})
 //Restrict the results to get only those where away team has higher score in half time
WHERE lh.HalfTimeHomeTeamGoals < lh.HalfTimeAwayTeamGoals
//For displaying the results in more readable format
WITH lh.HomeTeam AS Home_Team, lh.AwayTeam AS Away_Team,
lh.HalfTimeHomeTeamGoals AS Home_Team_Half_Time_Goals,
lh.HalfTimeAwayTeamGoals AS Away_Team_Half_Time_Goals,
lh.FullTimeResult AS Result, 'Liverpool' AS Winner
//Display the results
RETURN Home_Team, Away_Team, Home_Team_Half_Time_Goals,
Away_Team_Half_Time_Goals, Result, Winner
//Merge the results
UNION
//Retrieve the results when Liverpool is the Away Team and winner is the away team
MATCH (la:SoccerMatch{AwayTeam:'Liverpool',FullTimeResult:'A'})
//Restrict the results to get only those where home team has higher score in half time
WHERE la.HalfTimeHomeTeamGoals > la.HalfTimeAwayTeamGoals
//For displaying the results in more readable format
WITH la.HomeTeam AS Home_Team, la.AwayTeam AS Away_Team,
la.HalfTimeHomeTeamGoals AS Home_Team_Half_Time_Goals,
la.HalfTimeAwayTeamGoals AS Away_Team_Half_Time_Goals,
la.FullTimeResult AS Result, 'Liverpool' AS Winner
//Display the results
RETURN Home_Team, Away_Team, Home_Team_Half_Time_Goals,
Away_Team_Half_Time_Goals, Result, Winner
```



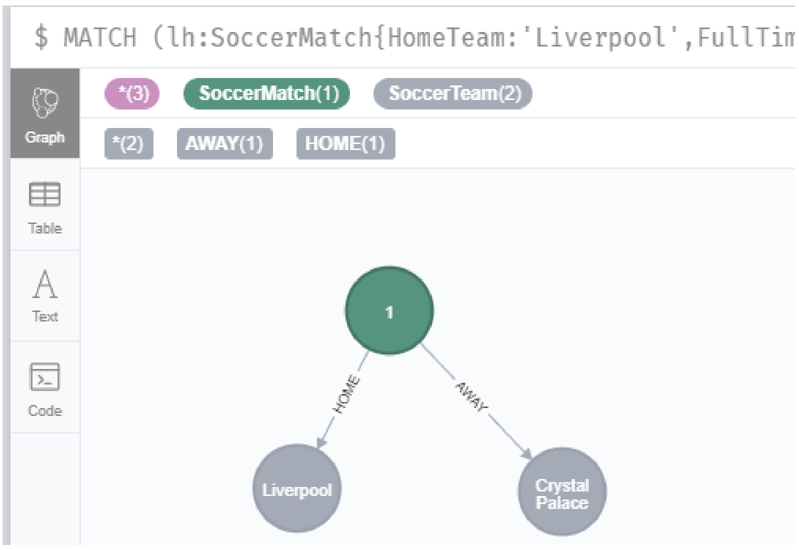
```
$ MATCH (lh:SoccerMatch{HomeTeam:'Liverpool',FullTimeResult:'H'}) WHE…
```

Home_Team	Away_Team	Home_Team_Half_Time_Goals	Away_Team_Half_Time_Goals	Result	Winner
"Liverpool"	"Crystal Palace"	0	1	"H"	"Liverpool"

Figure 13 – All Matches Liverpool Won Being Down First Half (Projected)

Code Without Projection (Display Everything):

```
//Retrieve the results when Liverpool is the Home Team and winner is the home team
MATCH (lh:SoccerMatch{HomeTeam:'Liverpool',FullTimeResult:'H'})
//Restrict the results to get only those where away team has higher score in half time
WHERE lh.HalfTimeHomeTeamGoals < lh.HalfTimeAwayTeamGoals
WITH lh AS DataNode
//Display the results
RETURN DataNode
//Merge the results
UNION
//Retrieve the results when Liverpool is the Away Team and winner is the away team
MATCH (la:SoccerMatch{AwayTeam:'Liverpool',FullTimeResult:'A'})
//Restrict the results to get only those where home team has higher score in half time
WHERE la.HalfTimeHomeTeamGoals > la.HalfTimeAwayTeamGoals
WITH la AS DataNode
//Display the results
RETURN DataNode
```



```
$ MATCH (lh:SoccerMatch{HomeTeam:'Liverpool',FullTim
```

Figure 14 - All Matches Liverpool Won Being Down First Half (Not Projected)

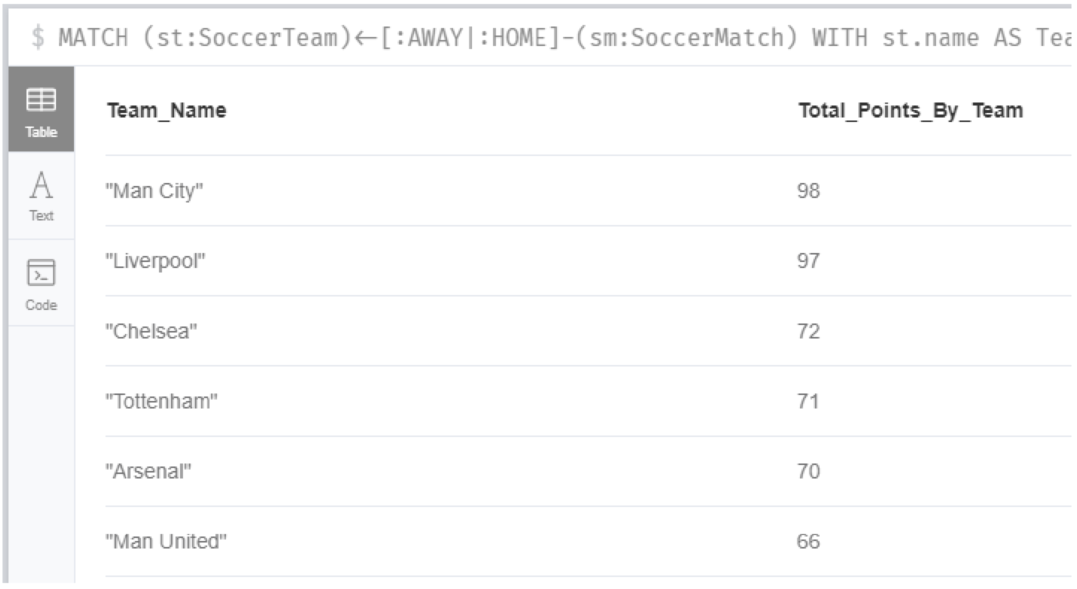
9) **Write a query to display the final ranking of all the teams based on their total points.**

- It has not been given about how to assign points to a team from a match result. We can assume the real-world scenario to calculate total points for a team based on their total wins or draws or losses. We can assume that for a win, a team will get 3 points and for a draw, a team will gain 1 point and for loss, there will be no point added.

Answer: The teams are displayed according to the ranking of their points (Man City leads with 98 points)

Code:

```
// Retrieve all the matches with all the teams involved
MATCH (st:SoccerTeam)<-[:AWAY|:HOME]-(sm:SoccerMatch)
// Use WITH clause to produce relevant names to use in the RETURN
WITH st.name AS Team_Name,
// Use CASE to filter results for whenever a team is the home team or away team
// Then supply the records with 3 for W or Win and 1 for D or Draw and none for loss
CASE
WHEN sm.HomeTeam=st.name AND sm.FullTimeResult="H" THEN 3
WHEN sm.AwayTeam=st.name AND sm.FullTimeResult="A" THEN 3
WHEN sm.FullTimeResult = "D" THEN 1
ELSE 0 END AS Points_By_Team_In_Match
// Display the results in a tabular format with readable names in Descending order
// The first team is the one with highest ranking and most points
RETURN Team_Name, SUM(Points_By_Team_In_Match) AS Total_Points_By_Team
ORDER BY Total_Points_By_Team DESC
```



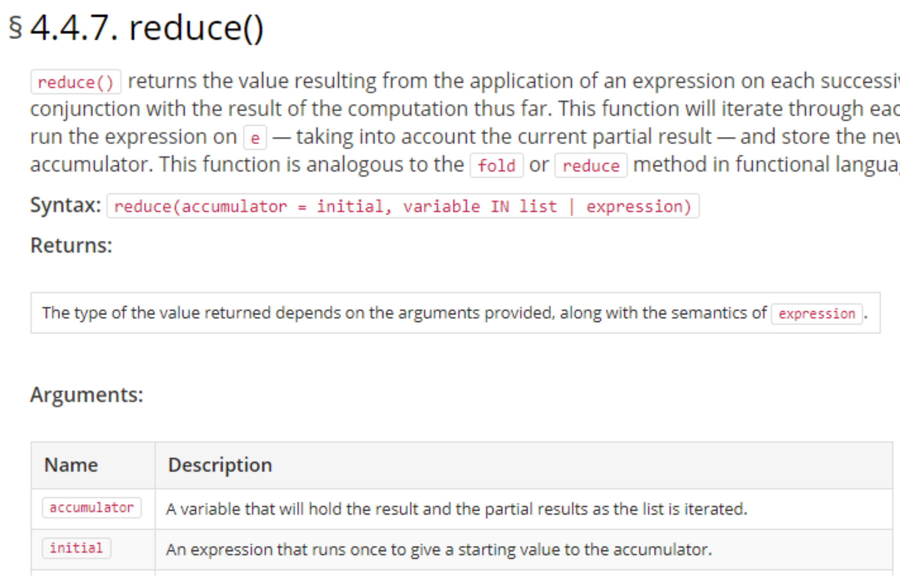
```
$ MATCH (st:SoccerTeam)←[:AWAY|:HOME]-(sm:SoccerMatch) WITH st.name AS Tea
```

Team_Name	Total_Points_By_Team
"Man City"	98
"Liverpool"	97
"Chelsea"	72
"Tottenham"	71
"Arsenal"	70
"Man United"	66

Figure 15 – Total Points by Each Team

10) Which team has drawn the most consecutive matches?

- This question seemed to be the toughest question by far and consumed a bit more of time. However, the answer was finally found after a lot of thoughts and coming across the reduce function in Neo4j documentation. It mentioned that the REDUCE function can be used on each element for a list and then return the value after applying any kind of expression on it (Neo4j.com, 2019). It gave the thought that REDUCE could be used to solve the required problem. For every game listed in order, if for one particular team, there are series of draws, these need to be counted. Every consecutive draw should add to the variable by one. There is also a possibility of having a series of consecutive draws for example a team having 3 consecutive draws, followed by a win, followed by another consecutive sets of draws. Our code should account for these conditions properly and not override the previous values unless the recently found streak is higher than the previous one. After going through all the matches for all the teams, the result can be displayed in descending order (which displays the team with highest consecutive draws at the top) and limiting the result to 1.



§ 4.4.7. reduce()

`reduce()` returns the value resulting from the application of an expression on each successive element in a list in conjunction with the result of the computation thus far. This function will iterate through each element `e` in the given list, run the expression on `e` — taking into account the current partial result — and store the new partial result in the accumulator. This function is analogous to the `fold` or `reduce` method in functional languages such as Lisp and Scala.

Syntax: `reduce(accumulator = initial, variable IN list | expression)`

Returns:

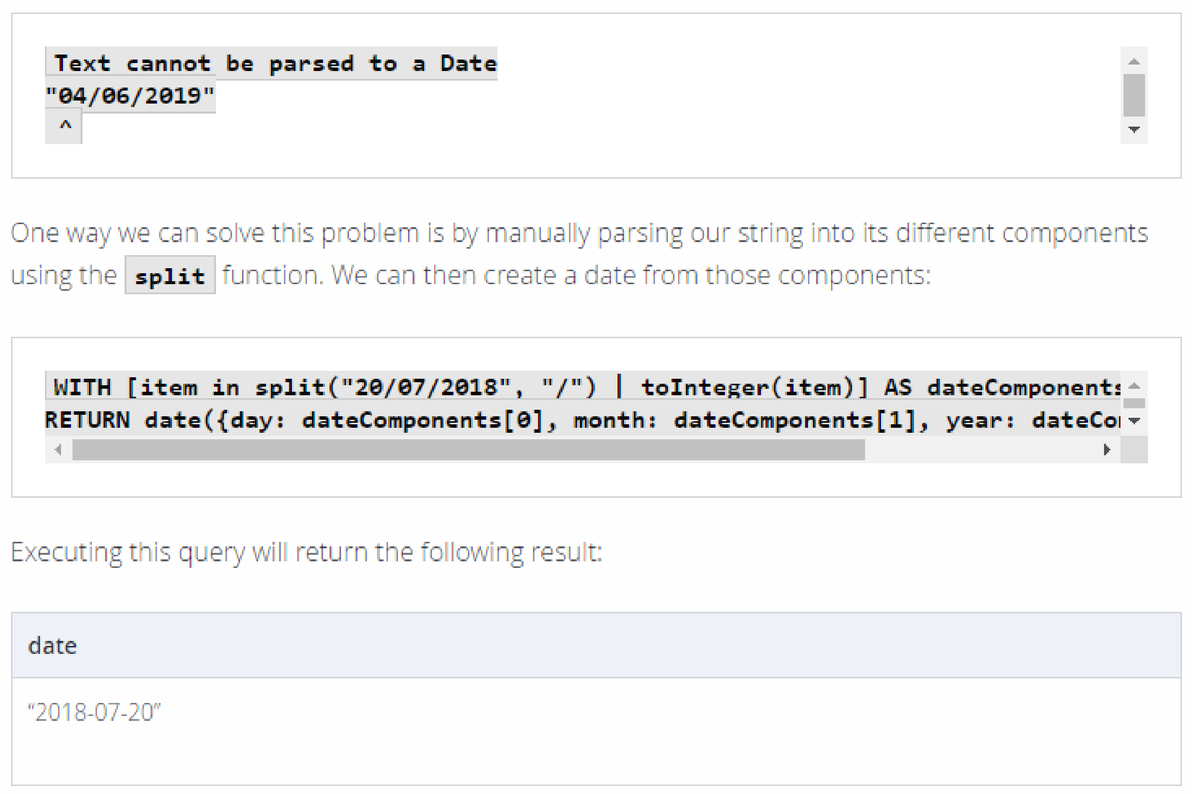
The type of the value returned depends on the arguments provided, along with the semantics of `expression`.

Arguments:

Name	Description
`accumulator`	A variable that will hold the result and the partial results as the list is iterated.
`initial`	An expression that runs once to give a starting value to the accumulator.

Figure 16 – The REDUCE function

- While working on the answers, it came to be known that the provided data type of date cannot be sorted as date but only as a string. This string value of date needed to be converted to a date data type for sorting. Neo4j website provides a way to do this. (Neo4j Graph Database Platform, 2019)



```
Text cannot be parsed to a Date
"04/06/2019"
 ^
```

One way we can solve this problem is by manually parsing our string into its different components using the `split` function. We can then create a date from those components:

```
WITH [item in split("20/07/2018", "/") | toInteger(item)] AS dateComponent
RETURN date({day: dateComponents[0], month: dateComponents[1], year: dateCo
```

Executing this query will return the following result:

date
"2018-07-20"

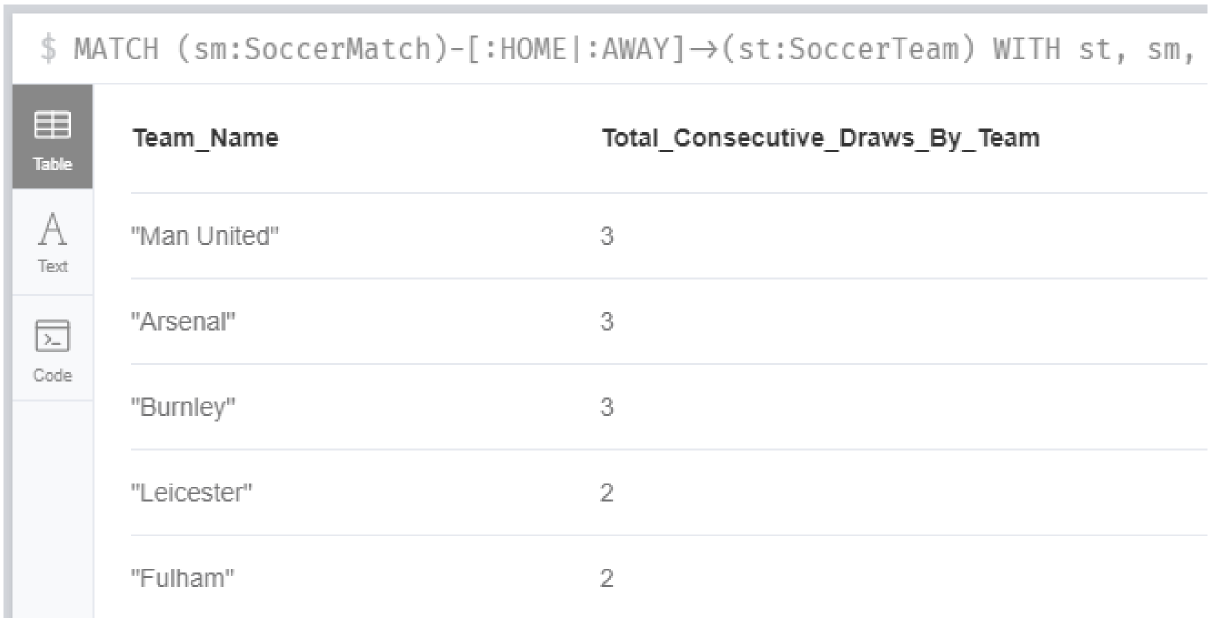
Figure 17 – Convert String to Date

Answer: Man United, Burnley and Arsenal with 3 consecutive draws for each team. If using LIMIT to restrict the output to one, Man United would be displayed.

Code:

```
// Retrieve all the matches with all the teams
MATCH
(sm:SoccerMatch)-[:HOME|:AWAY]->(st:SoccerTeam)
// Use WITH clause to produce relevant names to use in the RETURN and REDUCE
// The Final_Result will be used as a list and all the elements will be accessed to get and
// Work with the game results. The result should be a Draw and is denoted by D in our
Dataset.
// The result should be in order of date because it needs to account for the consecutive wins.
// For retrieving the date in sortable format, need to use the code provided in neo4j docs
WITH st, sm, [vals IN split(sm.Date, "/") | toInteger(vals)] AS dateVals
WITH st.name as Team_Name, sm.FullTimeResult AS Final_Result, date({day: dateVals[0],
month: dateVals[1], year: dateVals[2]}) AS game_date
// The sorting is done here according to the date of the match
ORDER BY game_date
// The RETURN part returns the team name and the max total consecutive draws of the team
RETURN Team_Name,
// The REDUCE function is used to set and manipulate the count variables
// The list contains the game results which would hold the value D for a draw
```

```
REDUCE(s = {maxConsecutiveDraws: 0, recentConsecutiveDraws: 0}, game_outcome IN
COLLECT(Final_Result) |
// For every draw result, the current consecutive draws should be incremented
// If the current value exceeds the maximum value, the maximum value should be replaced
// With the current one
  CASE WHEN game_outcome = "D"
    THEN {
// The +1 is being used here because otherwise the last element would miss and every team
// Would have one less total consecutive draw value
      maxConsecutiveDraws: CASE WHEN s.recentConsecutiveDraws + 1 >
s.maxConsecutiveDraws THEN s.recentConsecutiveDraws + 1 ELSE
s.maxConsecutiveDraws END,
      recentConsecutiveDraws: s.recentConsecutiveDraws + 1
    }
// If the game is not a draw, recent values is changed to 0 but maximum value is set to
// Whatever was held previously
    ELSE {maxConsecutiveDraws: s.maxConsecutiveDraws, recentConsecutiveDraws: 0}
  END
  ).maxConsecutiveDraws AS Total_Consecutive_Draws_By_Team
// Order the results in descending order to have the highest values first
ORDER BY Total_Consecutive_Draws_By_Team DESC
```



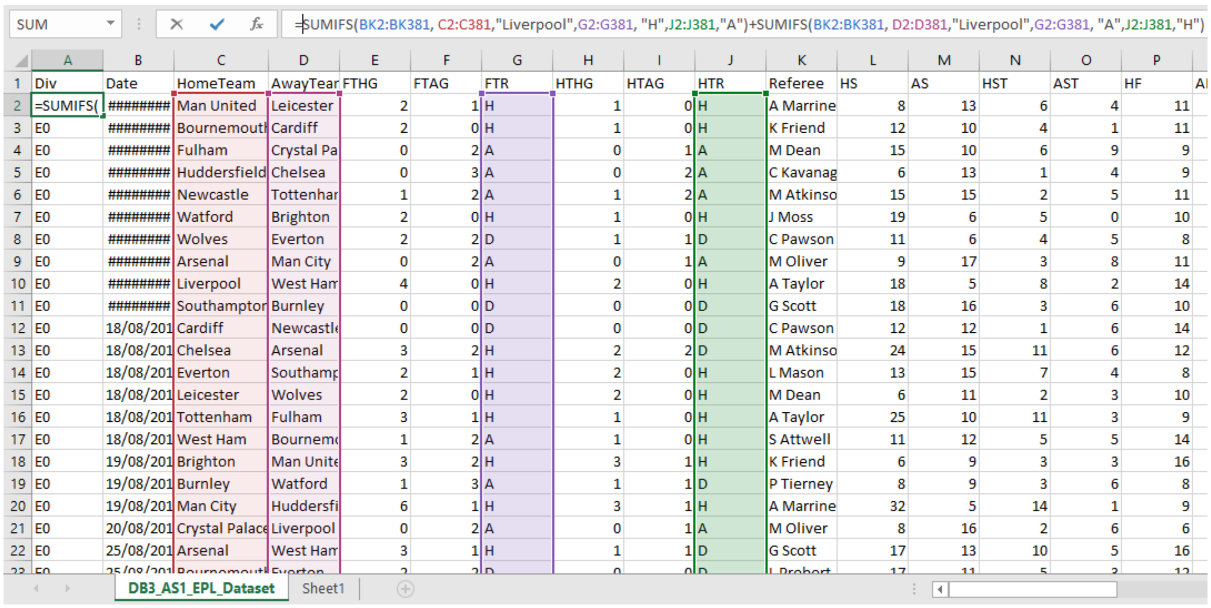
```
$ MATCH (sm:SoccerMatch)-[:HOME|:AWAY]→(st:SoccerTeam) WITH st, sm,
```

Team_Name	Total_Consecutive_Draws_By_Team
"Man United"	3
"Arsenal"	3
"Burnley"	3
"Leicester"	2
"Fulham"	2

Figure 18 – Total Consecutive Draws by Team

Verifying Results

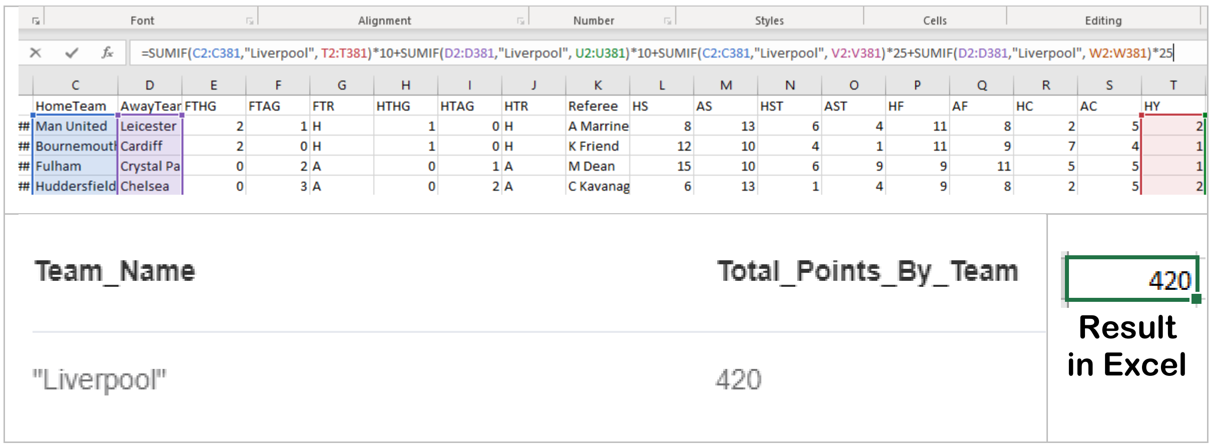
The results obtained were verified by directly querying the CSV file using Excel and Microsoft support website on how to use the Excel SUMIF commands. (Support.office.com, 2019)



```
=SUMIFS(BK2:BK381, C2:C381,"Liverpool",G2:G381, "H",J2:J381,"A")+SUMIFS(BK2:BK381, D2:D381,"Liverpool",G2:G381, "A",J2:J381,"H")
```

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	Div	Date	HomeTeam	AwayTear	FTHG	FTAG	FTR	HTHG	HTAG	HTR	Referee	HS	AS	HST	AST	HF
2	=SUMIFS(	########	Man United	Leicester	2	1	H	1	0	H	A Marrine	8	13	6	4	11
3	E0	########	Bournemoutl	Cardiff	2	0	H	1	0	H	K Friend	12	10	4	1	11
4	E0	########	Fulham	Crystal Pa	0	2	A	0	1	A	M Dean	15	10	6	9	9
5	E0	########	Huddersfield	Chelsea	0	3	A	0	2	A	C Kavanag	6	13	1	4	9
6	E0	########	Newcastle	Tottenhar	1	2	A	1	2	A	M Atkinso	15	15	2	5	11
7	E0	########	Watford	Brighton	2	0	H	1	0	H	J Moss	19	6	5	0	10
8	E0	########	Wolves	Everton	2	2	D	1	1	D	C Pawson	11	6	4	5	8
9	E0	########	Arsenal	Man City	0	2	A	0	1	A	M Oliver	9	17	3	8	11
10	E0	########	Liverpool	West Har	4	0	H	2	0	H	A Taylor	18	5	8	2	14
11	E0	########	Southamptor	Burnley	0	0	D	0	0	D	G Scott	18	16	3	6	10
12	E0	18/08/201	Cardiff	Newcastl	0	0	D	0	0	D	C Pawson	12	12	1	6	14
13	E0	18/08/201	Chelsea	Arsenal	3	2	H	2	2	D	M Atkinso	24	15	11	6	12
14	E0	18/08/201	Everton	Southamp	2	1	H	2	0	H	L Mason	13	15	7	4	8
15	E0	18/08/201	Leicester	Wolves	2	0	H	2	0	H	M Dean	6	11	2	3	10
16	E0	18/08/201	Tottenham	Fulham	3	1	H	1	0	H	A Taylor	25	10	11	3	9
17	E0	18/08/201	West Ham	Bournem	1	2	A	1	0	H	S Attwell	11	12	5	5	14
18	E0	19/08/201	Brighton	Man Unite	3	2	H	3	1	H	K Friend	6	9	3	3	16
19	E0	19/08/201	Burnley	Watford	1	3	A	1	1	D	P Tierney	8	9	3	6	8
20	E0	19/08/201	Man City	Huddersfi	6	1	H	3	1	H	A Marrine	32	5	14	1	9
21	E0	20/08/201	Crystal Palace	Liverpool	0	2	A	0	1	A	M Oliver	8	16	2	6	6
22	E0	25/08/201	Arsenal	West Har	3	1	H	1	1	D	G Scott	17	13	10	5	16
23	E0	25/08/201	Bournemoutl	Everton	2	2	D	0	0	D	L Probert	17	11	5	3	12

SUMIFS(BK2:BK381, C2:C381,"Liverpool",G2:G381, "H",J2:J381,"A")+SUMIFS(BK2:BK381, D2:D381,"Liverpool",G2:G381, "A",J2:J381,"H")



```
=SUMIF(C2:C381,"Liverpool", T2:T381)*10+SUMIF(D2:D381,"Liverpool", U2:U381)*10+SUMIF(C2:C381,"Liverpool", V2:V381)*25+SUMIF(D2:D381,"Liverpool", W2:W381)*25
```

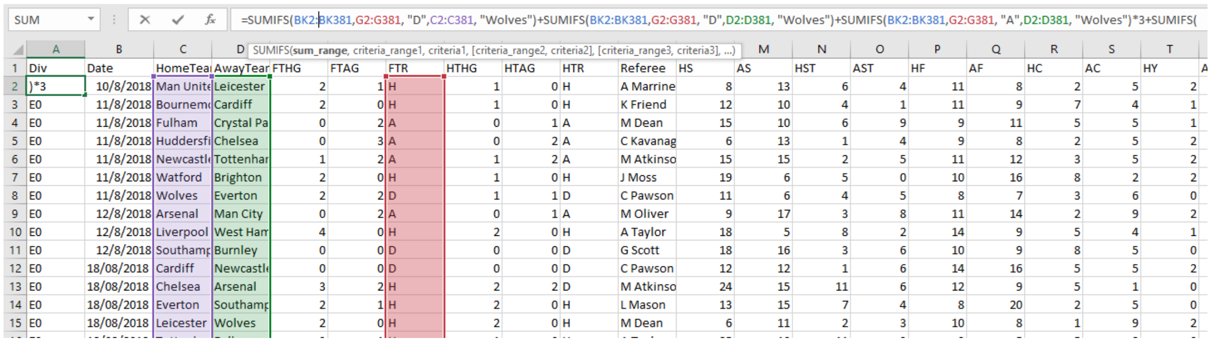
	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
	HomeTeam	AwayTear	FTHG	FTAG	FTR	HTHG	HTAG	HTR	Referee	HS	AS	HST	AST	HF	AF	HC	AC	HY
##	Man United	Leicester	2	1	H	1	0	H	A Marrine	8	13	6	4	11	8	2	5	2
##	Bournemoutl	Cardiff	2	0	H	1	0	H	K Friend	12	10	4	1	11	9	7	4	1
##	Fulham	Crystal Pa	0	2	A	0	1	A	M Dean	15	10	6	9	9	11	5	5	1
##	Huddersfield	Chelsea	0	3	A	0	2	A	C Kavanag	6	13	1	4	9	8	2	5	2

Team_Name	Total_Points_By_Team
"Liverpool"	420

420

Result in Excel

SUMIF(C2:C381,"Liverpool", T2:T381)*10+SUMIF(D2:D381,"Liverpool", U2:U381)*10+SUMIF(C2:C381,"Liverpool", V2:V381)*25+SUMIF(D2:D381,"Liverpool", W2:W381)*25



SUM

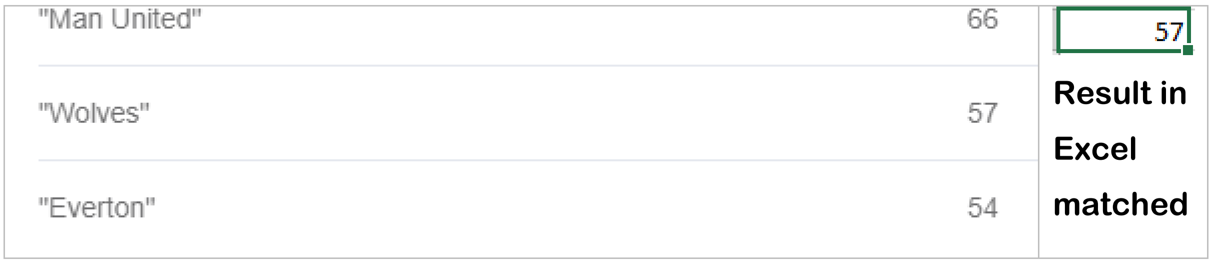
=SUMIFS(BK2:BK381,G2:G381, "D",C2:C381, "Wolves")+SUMIFS(BK2:BK381,G2:G381, "D",D2:D381, "Wolves")+SUMIFS(BK2:BK381,G2:G381, "A",D2:D381, "Wolves")*3+SUMIFS(

SUMIFS(**sum_range**, criteria_range1, criteria1, [criteria_range2, criteria2], [criteria_range3, criteria3], ...)

	A	B	C	D								M	N	O	P	Q	R	S	T
1	Div	Date	HomeTeai	AwayTear	FTHG	FTAG	FTR	HTHG	HTAG	HTR	Referee	HS	AS	HST	AST	HF	AF	HC	AC	HY
2	)*3	10/8/2018	Man Unite	Leicester	2	1	H	1	0	H	A Marrine	8	13	6	4	11	8	2	5	2
3	E0	11/8/2018	Bournem	Cardiff	2	0	H	1	0	H	K Friend	12	10	4	1	11	9	7	4	1
4	E0	11/8/2018	Fulham	Crystal Pa	0	2	A	0	1	A	M Dean	15	10	6	9	9	11	5	5	1
5	E0	11/8/2018	Huddersfi	Chelsea	0	3	A	0	2	A	C Kavanag	6	13	1	4	9	8	2	5	2
6	E0	11/8/2018	Newcastl	Tottenhar	1	2	A	1	2	A	M Atkinso	15	15	2	5	11	12	3	5	2
7	E0	11/8/2018	Watford	Brighton	2	0	H	1	0	H	J Moss	19	6	5	0	10	16	8	2	2
8	E0	11/8/2018	Wolves	Everton	2	2	D	1	1	D	C Pawson	11	6	4	5	8	7	3	6	0
9	E0	12/8/2018	Arsenal	Man City	0	2	A	0	1	A	M Oliver	9	17	3	8	11	14	2	9	2
10	E0	12/8/2018	Liverpool	West Han	4	0	H	2	0	H	A Taylor	18	5	8	2	14	9	5	4	1
11	E0	12/8/2018	Southamp	Burnley	0	0	D	0	0	D	G Scott	18	16	3	6	10	9	8	5	0
12	E0	18/08/2018	Cardiff	Newcastl	0	0	D	0	0	D	C Pawson	12	12	1	6	14	16	5	5	2
13	E0	18/08/2018	Chelsea	Arsenal	3	2	H	2	2	D	M Atkinso	24	15	11	6	12	9	5	1	0
14	E0	18/08/2018	Everton	Southamp	2	1	H	2	0	H	L Mason	13	15	7	4	8	20	2	5	0
15	E0	18/08/2018	Leicester	Wolves	2	0	H	2	0	H	M Dean	6	11	2	3	10	8	1	9	2

=SUMIFS(BK2:BK381,G2:G381, "D",C2:C381, "Wolves")+SUMIFS(BK2:BK381,G2:G381, "D",D2:D381, "Wolves")+SUMIFS(BK2:BK381,G2:G381, "A",D2:D381, "Wolves")*3+SUMIFS(BK2:BK381,G2:G381, "H", C2:C381, "Wolves")*3

Tested and confirmed the total points by Wolves is 57. This was also tested and confirmed for other football teams and the results were consistent.



"Man United"	66
"Wolves"	57
"Everton"	54

57

Result in Excel matched

APPENDIX I – REFERENCES

Neo4j APOC Library

Neo4j Graph Database Platform. (2019). *Neo4j APOC Library - Neo4j Graph Database Platform*. [online] Available at: https://neo4j.com/developer/neo4j-apoc/ [Accessed 6 Dec. 2019].

Neo4j REDUCE Function

Neo4j.com. (2019). *4.4. List functions - Chapter 4. Functions*. [online] Available at: https://neo4j.com/docs/cypher-manual/current/functions/list/#functions-reduce [Accessed 6 Dec. 2019].

Neo4j String to Date

Neo4j Graph Database Platform. (2019). *Neo4j: Convert string to date - Neo4j Graph Database Platform*. [online] Available at: https://neo4j.com/developer/kb/neo4j-string-to-date/ [Accessed 6 Dec. 2019].

Excel using SUMIFS

Support.office.com. (2019). *Sum values based on multiple conditions*. [online] Available at: https://support.office.com/en-us/article/sum-values-based-on-multiple-conditions-e610ae0f-4d27-480c-9119-eb644f1e847e#:~:targetText=The%20first%20step%20is%20to,the%20function%20requires%20as%20input. [Accessed 6 Dec. 2019].

Normalization 3.5NF, 4NF and 5NF

Questionsolves.com. (2019). *1NF, 2NF, 3NF, BCNF, 4NF and 5NF in Database Normalization*. [online] Available at: http://www.questionsolves.com/Website-Content/Normalization.php [Accessed 6 Dec. 2019].

Convert Relational to Graph Databases (Journal)

De Virgilio, R., Maccioni, A., & Torlone, R. (2013). Converting relational to graph databases. First International Workshop on Graph Data Management Experiences and Systems - GRADES '13. Available from DOI:10.1145/2484425.2484426 [Accessed 20 Dec. 2019].

Graph Databases replace Relational (Website)

Gillin, P. (2017). *Graph databases are hot, but can they break relational's grip? - SiliconANGLE*. [online] Available at: https://siliconangle.com/2017/12/02/graph-databases-hot-can-break-relationals-grip/ [Accessed 20 Dec. 2019].

dan111 (2012). *Comparison of Relational Databases and Graph Databases*. [online] Stack Overflow. Available at: https://stackoverflow.com/questions/13046442/comparison-of-relational-databases-and-graph-databases [Accessed 20 Dec. 2019].

Neo4j Importing CSV Files (Official Documentation)

Neo4j – Importing CSV Data into Neo4j. (2019). *CSV Import Guide: How to Use LOAD CSV Command for Neo4j*. [online] Available at: https://neo4j.com/developer/guide-import-csv/ [Accessed 20 Dec. 2019].

Neo4j – Importing CSV Files Neo4j Desktop and Sandbox. (2019). *Importing CSV Files: Neo4j Desktop and Sandbox - Neo4j Graph Database Platform*. [online] Available at: https://neo4j.com/developer/kb/import-csv-locations/ [Accessed 20 Dec. 2019].

APPENDIX II – LAB EXERCISES

WEEK 1 - Part I

Boyce-Codd Normal Form (3.5NF):

It requires the tables to be at least in third normal form. The main rule for 3.5NF is for any kind of dependency from A to B, A should not be non-prime if B is prime. Or in other terms, B should be the super key of the table in such case.

4NF:

The first rule states that the table should at least be in BCNF. The other rule is there should not be more than one multi-valued dependency. Multi-valued dependency could occur when for any particular value of A, in a dependency A to B, several values of B exist. Only tables with more than two columns are considered for possibility of multi-valued dependency.

5NF:

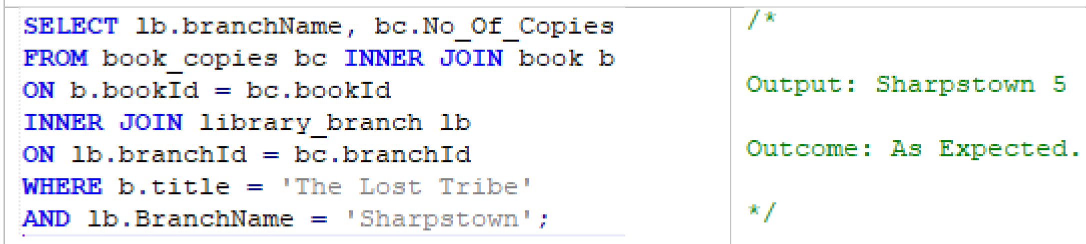
As in previous normal forms, the 5NF requires the table to be at least in fourth normal form. The other rule is that any tables should be decomposed as furthest as possible without losing any original data formed when re-joining the tables. It could also be referred to as furthest lossless decomposition. (Questionsolves.com, 2019)

WEEK 1 – Part II

The source code for activities is included here. The comments in the source code after each question infer as to whether the output and expected outcomes match and hence verifying whether the answer is correct.

PART I

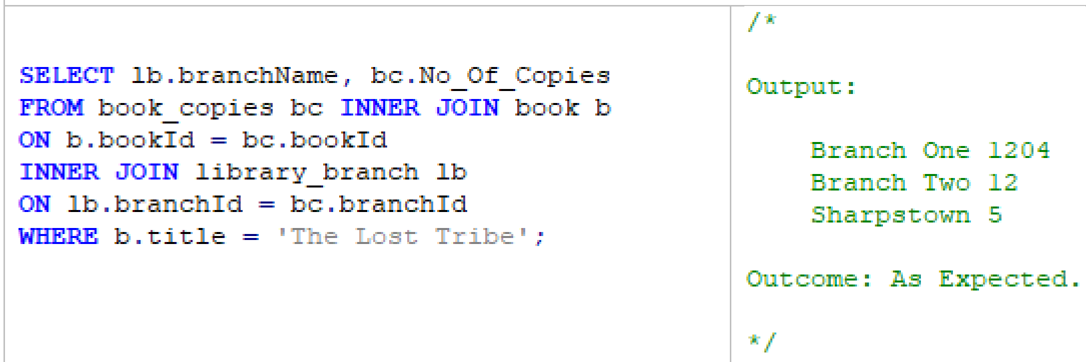
Question: How many copies of the book titled The Lost Tribe are owned by the library branch whose name is "Sharpstown"?



```sql
SELECT lb.branchName, bc.No_Of_Copies
FROM book_copies bc INNER JOIN book b
ON b.bookId = bc.bookId
INNER JOIN library_branch lb
ON lb.branchId = bc.branchId
WHERE b.title = 'The Lost Tribe'
AND lb.BranchName = 'Sharpstown';
```

```
/*

Output: Sharpstown 5

Outcome: As Expected.

*/
```

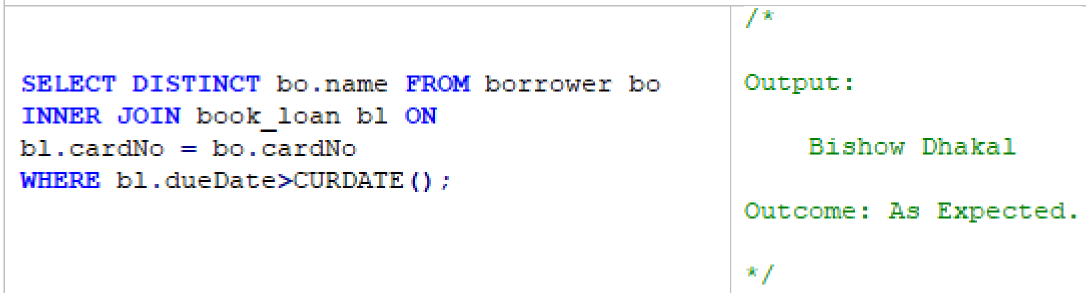
Question: How many copies of the book titled The Lost Tribe are owned by each library branch?



```sql
SELECT lb.branchName, bc.No_Of_Copies
FROM book_copies bc INNER JOIN book b
ON b.bookId = bc.bookId
INNER JOIN library_branch lb
ON lb.branchId = bc.branchId
WHERE b.title = 'The Lost Tribe';
```

```
/*

Output:

    Branch One 1204
    Branch Two 12
    Sharpstown 5

Outcome: As Expected.

*/
```

Question: Retrieve the names of all borrowers who do not have any books checked out.



```sql
SELECT DISTINCT bo.name FROM borrower bo
INNER JOIN book_loan bl ON
bl.cardNo = bo.cardNo
WHERE bl.dueDate>CURDATE();
```

```
/*

Output:

    Bishow Dhakal

Outcome: As Expected.

*/
```

Question: For each book that is loaned out from the "Sharpstown" branch and whose DueDate is today, retrieve the book title, the borrower's name, and the borrower's address.

```sql
SELECT b.title, bo.name, bo.address
FROM book_loan bl INNER JOIN book b
ON b.bookId = bl.bookId
INNER JOIN library_branch lb
ON lb.branchId = bl.branchId
INNER JOIN borrower bo
ON bo.cardNo = bl.cardNo
WHERE lb.branchName = 'Sharpstown'
AND bl.dueDate = CURDATE();
```

```
/*

Output: NULL

Outcome: As Expected.

*/
```

Question: For each library branch, retrieve the branch name and the total number of books loaned out from that branch.

```sql
SELECT lb.branchName, COUNT(bl.branchId)
FROM library_branch lb
INNER JOIN book_loan bl ON
lb.branchId = bl.branchId
GROUP BY bl.branchId;
```

```
Output:

    Branch One 1
    Branch Two 9
    Sharpstown 2

Outcome: As Expected.
```

Question: Retrieve the names, addresses, and number of books checked out for all borrowers who have more than five books checked out.

```sql
SELECT br.name, br.address, COUNT(bl.cardNo)
FROM borrower br
INNER JOIN book_loan bl ON
br.cardNo = bl.cardNo
WHERE COUNT(bl.cardNo)>5
GROUP BY bl.cardNo;
```

```
/*

Output: NULL

Outcome: Pending.

*/
```

Question: For each book authored (or co-authored) by "Stephen King", retrieve the title and the number of copies owned by the library branch whose name is "Central"

```sql
SELECT b.title, bc.No_Of_Copies,
lb.BranchName FROM library_branch lb
INNER JOIN book_copies bc ON
bc.branchId = lb.branchId
INNER JOIN book b ON
b.bookId = bc.bookId
INNER JOIN book_authors ba ON
ba.bookId = b.bookId
WHERE ba.authorName = 'STEPHEN KING'
AND lb.branchName = 'CENTRAL';
```

```
/*

Output: NULL

Outcome: As Expected.

*/
```

PART II

Question: Retrieve the names of employees in department 5 who work more than 10 hours per week on the 'ProductX' project.

```sql
SELECT e.fname, e.minit, e.lname FROM
employee e INNER JOIN works_on wo ON
e.ssn = wo.essn
WHERE wo.hours > 10
AND e.fname IN (
    SELECT e.fname FROM employee e
    INNER JOIN department d ON
    e.dno = d.dnumber
    INNER JOIN project p ON
    p.dnum = d.dnumber
    WHERE d.dname = 'DEPARTMENT 5'
    AND p.pname = 'ProductX'
);
```

```
/*

Output:

    Hari Bahadur Aryal
    Shyam Nath Ghale

Outcome: As Expected.

*/
```

Question: For each project, list the project name and the total hours per week (by all employees) spent on that project.

```sql
SELECT p.pname, SUM(wo.hours)
FROM project p
INNER JOIN works_on wo ON
p.pnumber = wo.pno
GROUP BY wo.pno;
```

```
Output:

    ProductX 45
    ProductY 7
    ProductZ 15
    ProductXYZ 20

Outcome: As Expected.
```

Question: Retrieve the names of employees who work on every project.

```sql
SELECT e.fname, e.minit, e.lname
FROM employee e
WHERE NOT EXISTS
    (SELECT p.pnumber FROM project p
     WHERE p.pnumber NOT IN
        (SELECT wo.pno FROM works_on wo
         WHERE wo.essn = e.ssn));
```

```
/*

Output: Rima Kumari Upreti

Outcome: As Expected.

*/
```

Question: Retrieve the names of employees who do not work on any project.

```sql
SELECT DISTINCT e.fname, e.minit, e.lname
FROM employee e
LEFT JOIN works_on wo ON
e.ssn = wo.essn
WHERE wo.essn IS NULL;
```

```
/*

Output: Rabindra Nath

Outcome: As Expected.

*/
```

Question: Find the names and addresses of employees who work on at least one project located in Houston but whose department has no location in Houston.

```sql
SELECT DISTINCT e.fname, e.minit,
e.lname, e.address FROM employee e
INNER JOIN department d ON
d.dnumber = e.dno
INNER JOIN dept_locations dl ON
d.dnumber = dl.dnumber
INNER JOIN project p ON
p.dnum = d.dnumber
INNER JOIN works_on wo ON
wo.pno = p.pnumber AND wo.essn = e.ssn
WHERE p.plocation = 'HOUSTON' AND
dl.dlocation != 'HOUSTON';
```

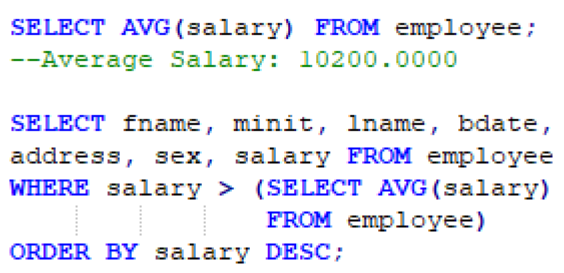
```
/*

Output:

Ram K Thapa NY, USA
Hari B Aryal AR,  Texas
Shyam B.K. TK, Kathmandu

Outcome: As Expected.

*/
```

Question: List the last names of department managers who have no dependents.

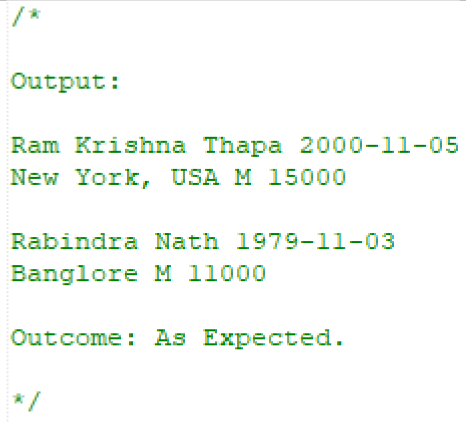
```sql
SELECT DISTINCT e.lname
FROM employee e
INNER JOIN department d
ON d.dnumber = e.dno
LEFT JOIN dependent de ON
e.ssn = de.essn
WHERE de.essn IS NULL
AND e.ssn IN (SELECT mgrssn
              FROM department);
```

```
/*

Output: Ghale

Outcome: As Expected.

*/
```

Question: Find details of those employees whose salary is > the average salary for all employees. Output salary in descending order

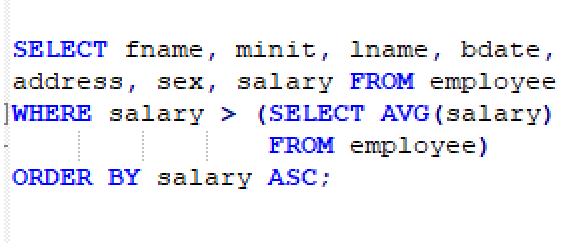


```
SELECT AVG(salary) FROM employee;
--Average Salary: 10200.0000

SELECT fname, minit, lname, bdate,
address, sex, salary FROM employee
WHERE salary > (SELECT AVG(salary)
                FROM employee)
ORDER BY salary DESC;
```

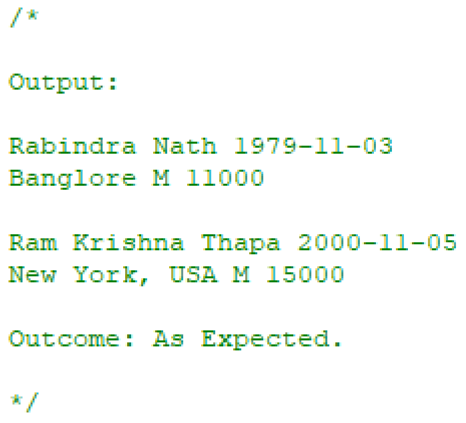


```
/*

Output:

Ram Krishna Thapa 2000-11-05
New York, USA M 15000

Rabindra Nath 1979-11-03
Banglore M 11000

Outcome: As Expected.

*/
```

Question: Find details of those employees whose salary is > the average salary for all employees. Output salary in ascending order



```
SELECT fname, minit, lname, bdate,
address, sex, salary FROM employee
WHERE salary > (SELECT AVG(salary)
                FROM employee)
ORDER BY salary ASC;
```



```
/*

Output:

Rabindra Nath 1979-11-03
Banglore M 11000

Ram Krishna Thapa 2000-11-05
New York, USA M 15000

Outcome: As Expected.

*/
```

WEEK 2

Student activities from week 2 contained some questions to be answered. The answers are listed below in order.

- NoSQL is a different approach of storing and dealing with data which is quite different from the traditional RDB approach wherein the data was stored in tables.

-The major reason to pick NoSQL was to have horizontal scaling instead of vertical scaling that relational databases allow. The ACID principle was difficult to follow in a relational database and NoSQL was introduced for proper availability, consistency and partition tolerance (CAP Theorem).

- The different kinds of NoSQL data stores:

  - **Key-Value**: Which is designed to handle key/value stores. The data model used for key-value pairs is a collection of pairs of keys and values. Examples: DynamoDB (Amazon), Voldemort (LinkedIn), etc.
  - **Document-Based**: Which can model more complex data such as objects unlike the key-value pairs. Documents similar to JSON, named BSON are stored in this model. The most familiar example is MongoDB.
  - **Column-Based**: It stores data in a rather similar way to traditional RDBMS, however, the data is stored in columns instead of rows. Common examples include Bigtable (Google) and HBase (Apache).
  - **Graph-Based**: Focuses more on modelling the data which includes nodes, properties or relationships. The graph-based databases will be discussed more in upcoming classes with the introduction to Neo4j and Cypher. Examples: Neo4j, FlockDB.

- Comparison of Relational Databases and NoSQL Databases

	Relational Databases	**NoSQL Databases**
Data Structure	The data is stored in tables where it needs to be properly structured and defined in a schema.	Stored in the four types of data stores discussed earlier which is non-relational and schema-less. Data can be unstructured.

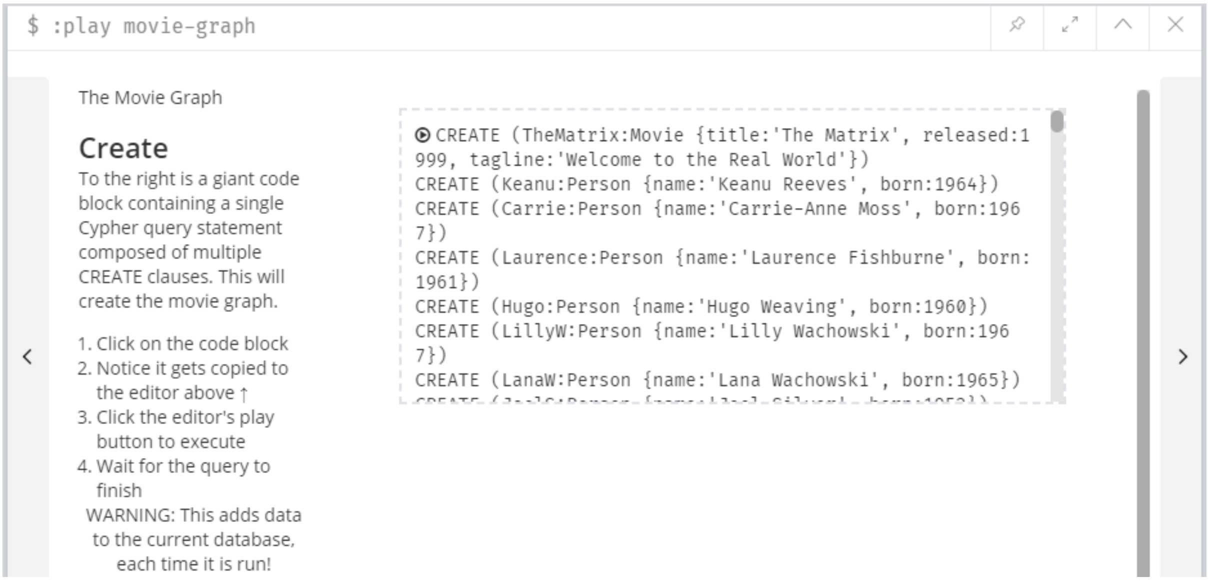
Scalability	Vertically scalable (example: increasing number of floors)	Horizontally scalable (example: increasing number of buildings)
Performance	Perhaps poor performance than NoSQL as NoSQL can be structured to fit exact user needs and can produce market ready applications sooner.	The performance mostly relies on the type of data and data store. Better performance than traditional RDBMS as it removes the need of SQL JOINs.
Data Integrity	Provides better data integrity	Inferior in data integrity
Query Languages	SQL	CQL, BSON, etc.

Question: Challenges faced by traditional firms that use relational databases in adopting NoSQL databases.

- Although NoSQL and graph databases in particular provide more effective and efficient way to store and connect data in the current context where data are very interconnected to each other, it could be a problem to convert the existing relational database systems to graph database systems. It could be very tough to convert a large system with a lot of data to a graph database as it would require a lot of memory and processing power. Although with the help of APIs or CSV exports, data could be transferred to traditional firms, it is not sure whether the datatypes will be exactly the same and transferring complex types will not be directly possible. There will also be a lot of sensitive and personal information involved which should not be allowed to leak or get deleted and thus they have to make sure it works. However, there have been some papers written about potential solutions to this problem. This paper by Roberto and co. discusses about a comprehensive approach to solving this problem. They propose a solution for automatic migration of database from relational to graph database systems. Their feasibility report highlights the success of their experiments that there is no data loss and the queries are translated efficiently (De Virgilio, et al., 2013).

WEEK 3

The activities from this week involved starting up Neo4j and getting familiar with the User Interface. The Neo4j’s movie graph was created and manipulated to learn about different interactions. Personally, this new kind of database structure was found to be remarkable and provided a broader view for databases for example the possible ways of how this can be implemented to a very large dataset. Gave an understanding of how big tech giants such as LinkedIn could potentially provide relevant job suggestions according to skills and interests of a person with the possible use of graph database and how many big companies such as eBay, Walmart and Adobe use Neo4j. Some screenshots from the movie database practice will be included below.



```
$ :play movie-graph
```

The Movie Graph

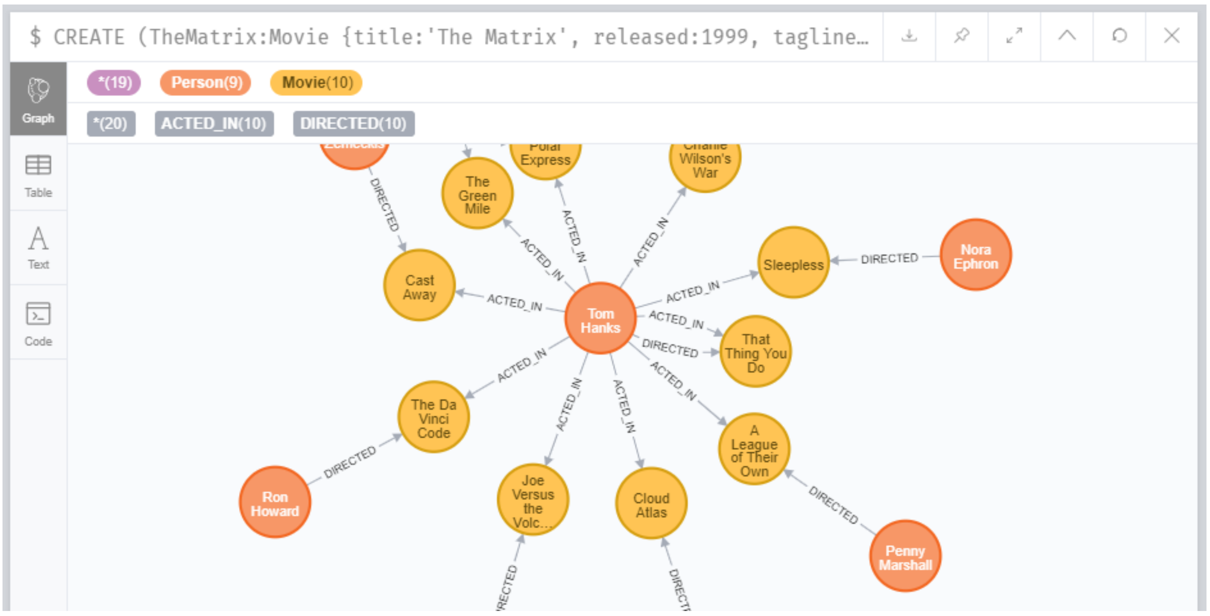
Create

To the right is a giant code block containing a single Cypher query statement composed of multiple CREATE clauses. This will create the movie graph.

1. Click on the code block
2. Notice it gets copied to the editor above ↑
3. Click the editor's play button to execute
4. Wait for the query to finish

WARNING: This adds data to the current database, each time it is run!

```
CREATE (TheMatrix:Movie {title:'The Matrix', released:1
999, tagline:'Welcome to the Real World'})
CREATE (Keanu:Person {name:'Keanu Reeves', born:1964})
CREATE (Carrie:Person {name:'Carrie-Anne Moss', born:196
7})
CREATE (Laurence:Person {name:'Laurence Fishburne', born:
1961})
CREATE (Hugo:Person {name:'Hugo Weaving', born:1960})
CREATE (LillyW:Person {name:'Lilly Wachowski', born:196
7})
CREATE (LanaW:Person {name:'Lana Wachowski', born:1965})
```



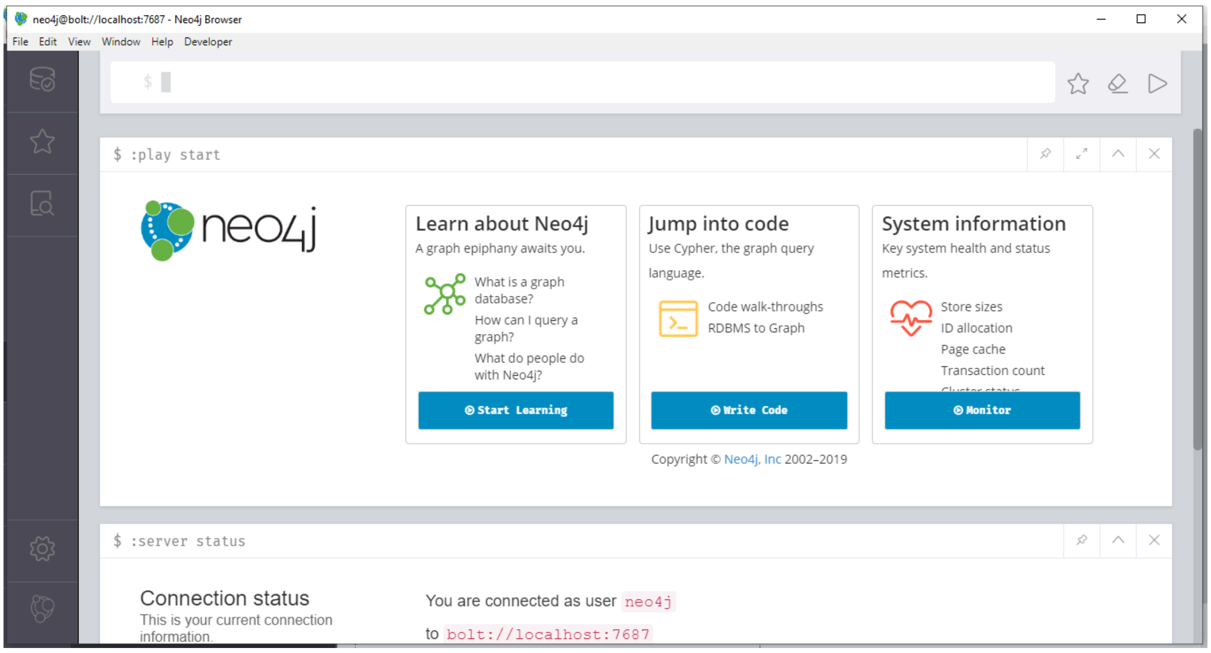
```
$ CREATE (TheMatrix:Movie {title:'The Matrix', released:1999, tagline…
```

WEEK 4

The student activities in week 4 involved practicing the Cypher syntaxes introduced in the slides. The slides introduced different commands such as creating a node, relationship, and several other clauses. The exercise needed clearing the full database using the DETACH DELETE command and then creating the nodes and relationships from the code provided in the lesson. Nodes such as lecturer and module and TEACHES relationship between these nodes were created, manipulated and deleted for getting familiar with Cypher.

Exercises:

- Starting the Neo4j database:


neo4j@bolt://localhost:7687 - Neo4j Browser
File Edit View Window Help Developer
Start Learning
Write Code
Monitor

```
$ :play start
```

Learn about Neo4j
A graph epiphany awaits you.
What is a graph database?
How can I query a graph?
What do people do with Neo4j?

Jump into code
Use Cypher, the graph query language.
Code walk-throughs
RDBMS to Graph

System information
Key system health and status metrics.
Store sizes
ID allocation
Page cache
Transaction count

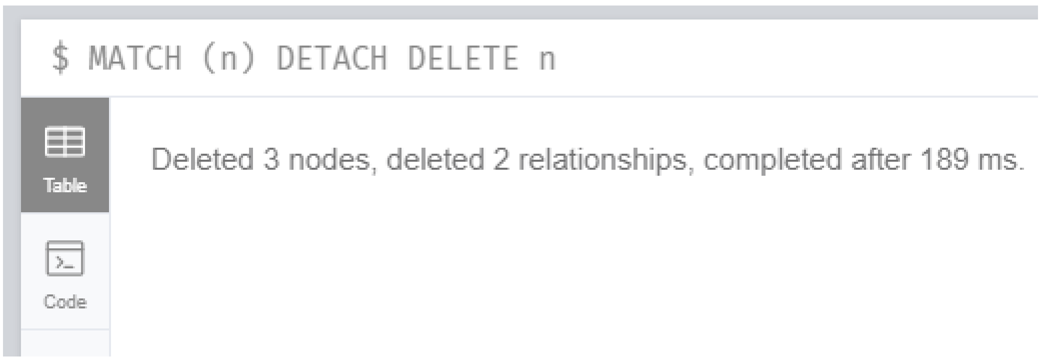
Copyright © Neo4j, Inc 2002–2019

```
$ :server status
```

Connection status
This is your current connection information.

You are connected as user `neo4j`
to `bolt://localhost:7687`

- Clearing the database using DETACH DELETE:

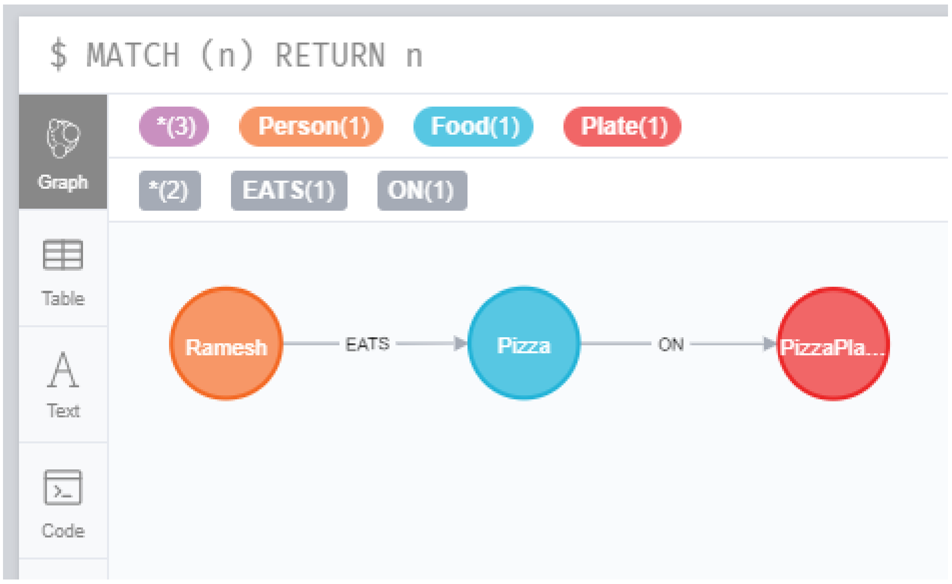


```
$ MATCH (n) DETACH DELETE n
```

```
Deleted 3 nodes, deleted 2 relationships, completed after 189 ms.
```

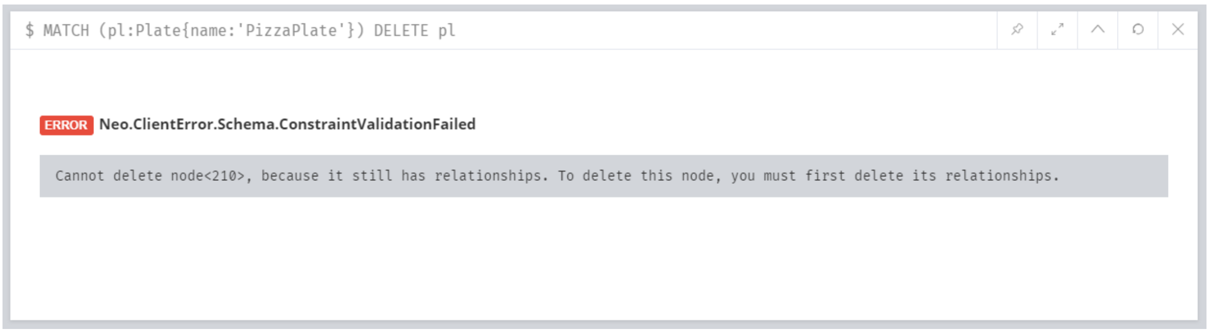
- Creating simple nodes and relationships:

```
CREATE (p:Person{name:'Ramesh',age:20})

CREATE (f:Food{name:'Pizza'})

CREATE (pl:Plate{name:'PizzaPlate'})
MATCH (p:Person{name:'Ramesh'}),(f:Food{name'Pizza'}),(pl:Plate{name:'PizzaPlate'})
CREATE (p)-[e:EATS]→(f)-[o:ON]→(pl)
```



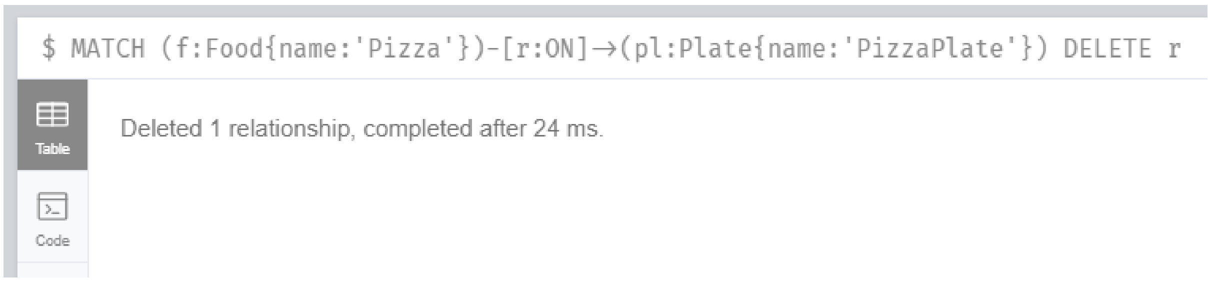
```
$ MATCH (n) RETURN n
```

- Deleting some nodes and relationships:



```
$ MATCH (pl:Plate{name:'PizzaPlate'}) DELETE pl

ERROR Neo.ClientError.Schema.ConstraintValidationFailed

Cannot delete node<210>, because it still has relationships. To delete this node, you must first delete its relationships.
```

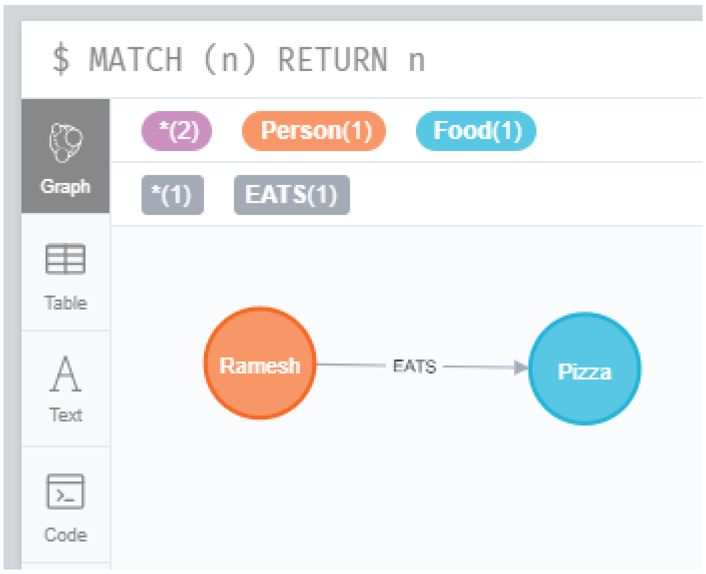
From this, came to confirm that nodes cannot be deleted without first removing any existing relationships. So, need to remove the ON relationship before removing this node.



```
$ MATCH (f:Food{name:'Pizza'})-[r:ON]→(pl:Plate{name:'PizzaPlate'}) DELETE r
```

```
Deleted 1 relationship, completed after 24 ms.
```

Now after running the above code, the node gets deleted and this is the final outcome.



```
$ MATCH (n) RETURN n
```

Question: Does the emergence of graph databases mean the death of relational databases? do some research and justify your answer. Cite the resources using Harvard reference style.

- As in the current world, data is more and more connected to each other, graph databases provide a very efficient and effective approach to store data and connect it to each other, and systems can benefit with the ability to scale to huge datasets (De Virgilio, et al., 2013). While graph databases could be very useful for such large datasets, it could be time consuming and costly to setup for smaller scale usages. There are also other benefits of using relational databases such as it being much faster while operating on high number of rows and the queries could be faster (but longer to write) (dan111, 2012). There are also several existing systems or businesses that use relational database. Transferring to a graph database could be costly and sometimes too risky for these businesses. It is also true that for the systems with high number of uniform and densely populated tables, relational systems perform better (Gillin, 2017).

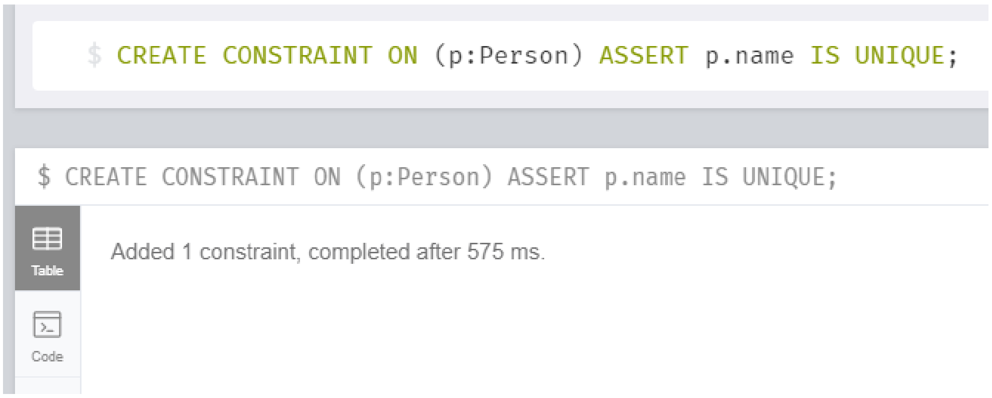
WEEK 5

Week 5 was more focused on Cypher commands and had more student activities than theoretical aspects. The upcoming weeks are almost structured in a similar way and thus have more student activities.

The lab activities started from creating and dropping constraints which need to be created before creating any nodes, followed by indexes. The latter part included the introduction of SET clause which allowed to update or add labels or properties. The SET clause is also found to be useful when copying properties from nodes or relationships. Similarly, another clause REMOVE can be used to remove properties or labels. The final part asked the question about whether resetting a label would update the label. The answer would be no as in order to re SET a label, it should first be deleted and then added again as it cannot be updated.

- Creating, verifying and dropping constraints

Constraints can be used to enforce some rules. It is to be noted that constraints need to be defined before creating nodes.

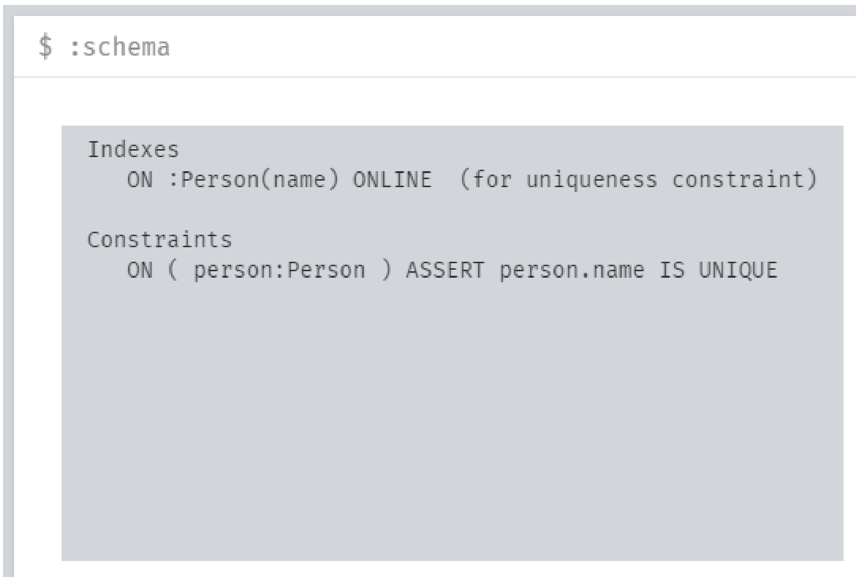


```
$ CREATE CONSTRAINT ON (p:Person) ASSERT p.name IS UNIQUE;
```

```
$ CREATE CONSTRAINT ON (p:Person) ASSERT p.name IS UNIQUE;
```

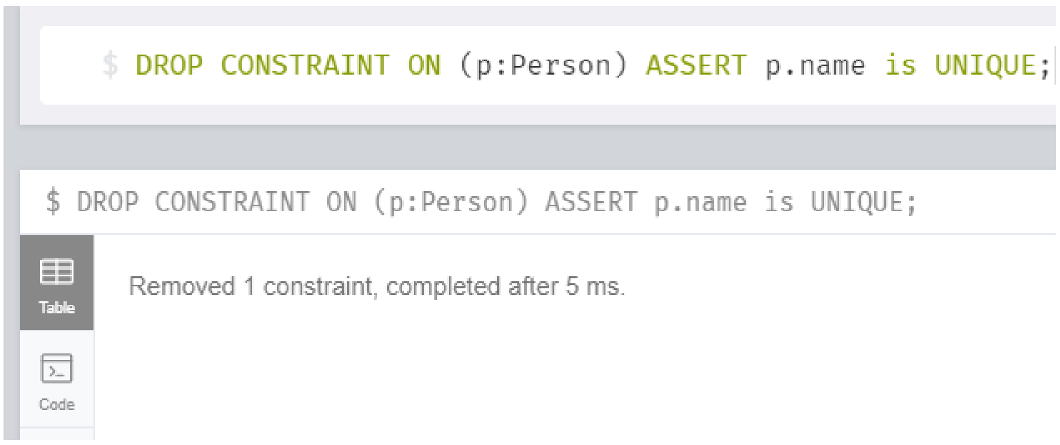
```
Added 1 constraint, completed after 575 ms.
```

Constraints can be verified using :schema command.



```
$ :schema

Indexes
   ON :Person(name) ONLINE  (for uniqueness constraint)

Constraints
   ON ( person:Person ) ASSERT person.name IS UNIQUE
```

Constraints can be dropped in a similar way by replacing CREATE with DROP.



```
$ DROP CONSTRAINT ON (p:Person) ASSERT p.name is UNIQUE;

$ DROP CONSTRAINT ON (p:Person) ASSERT p.name is UNIQUE;
```

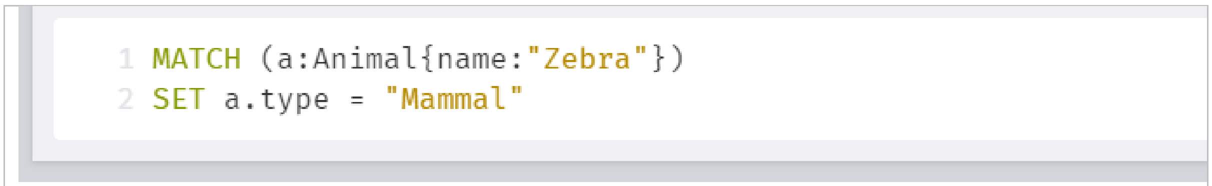
Removed 1 constraint, completed after 5 ms.

- Creating and dropping indexes

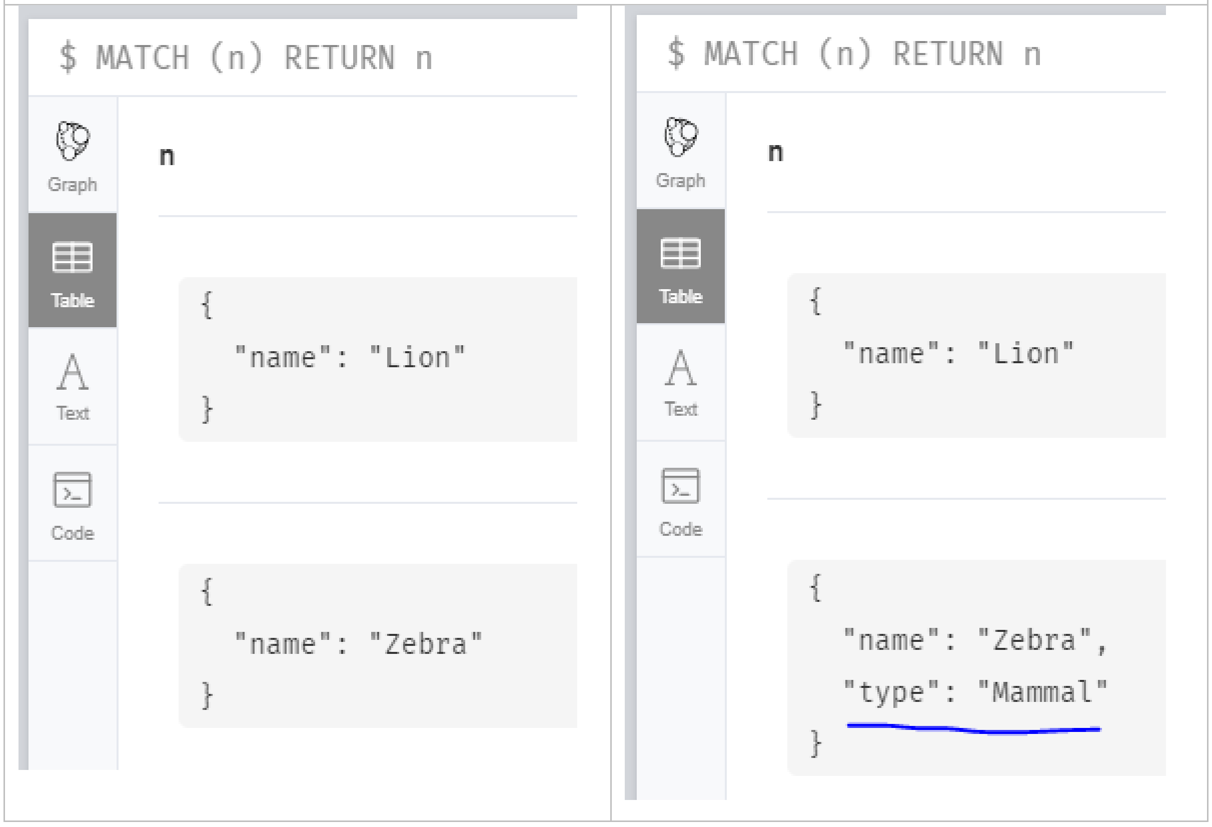
```
CREATE INDEX ON :Person(rollno)

DROP INDEX ON :Person(rollno)
```

- Setting attributes

The SET command can be used to update an existing label on a node or to add new properties. Properties can also be added using maps.



```
MATCH (a:Animal{name:"Zebra"})
SET a.type = "Mammal"
```



```
$ MATCH (n) RETURN n
```

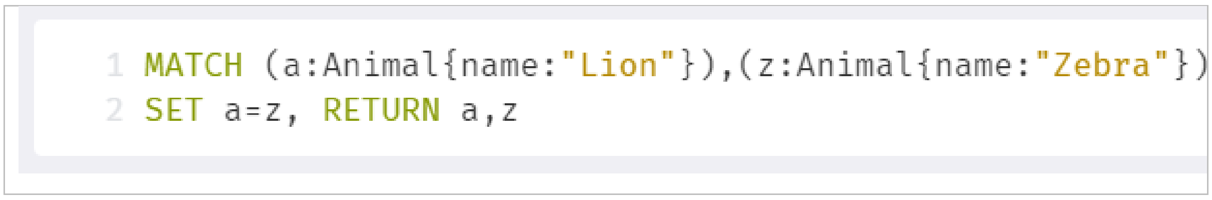
```
n
{
  "name": "Lion"
}
{
  "name": "Zebra"
}
```

```
$ MATCH (n) RETURN n
```

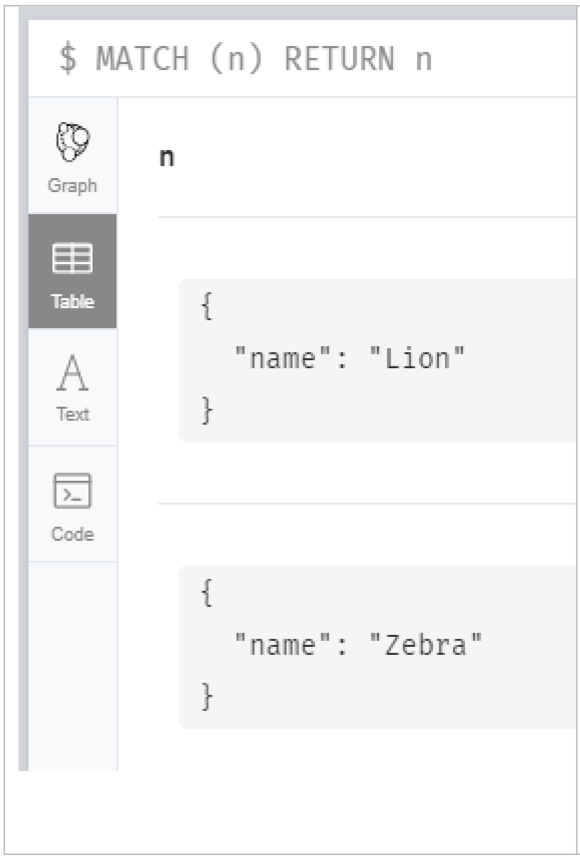
```
n
{
  "name": "Lion"
}
{
  "name": "Zebra",
  "type": "Mammal"
}
```

- Copying properties

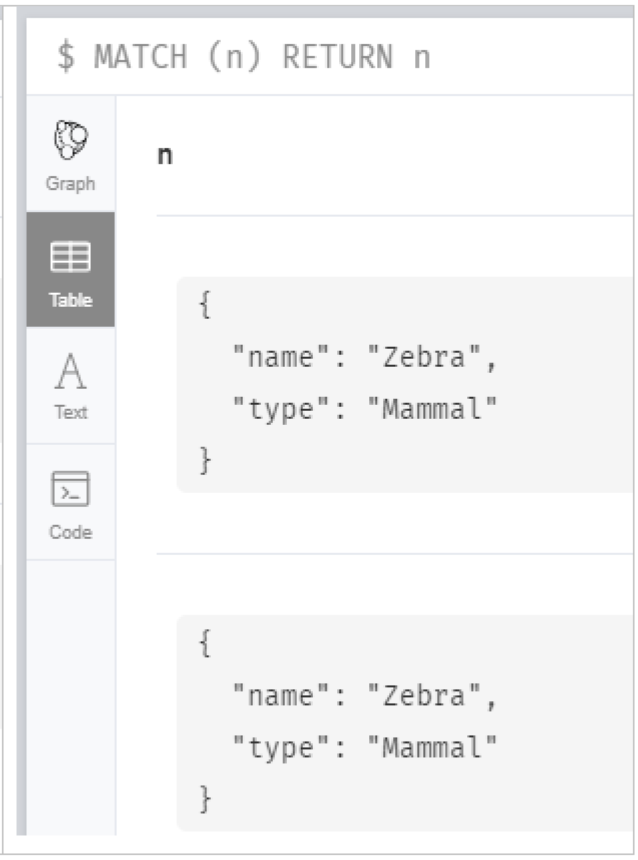
Properties can be copied from a node to other or even among relationships. This removes existing properties from the receiving end.



```
MATCH (a:Animal{name:"Lion"}),(z:Animal{name:"Zebra"})
SET a=z, RETURN a,z
```



```
$ MATCH (n) RETURN n

n

{
  "name": "Lion"
}

{
  "name": "Zebra"
}
```

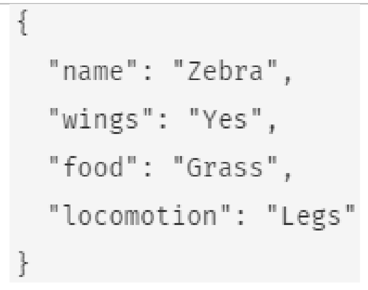


```
$ MATCH (n) RETURN n

n

{
  "name": "Zebra",
  "type": "Mammal"
}

{
  "name": "Zebra",
  "type": "Mammal"
}
```

Properties can be copied to relationships in a similar manner.

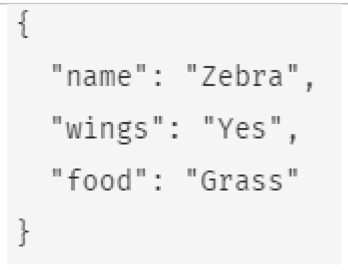
- Remove properties

Properties can be removed either by setting them to NULL or by using REMOVE command.



```
{
  "name": "Zebra",
  "wings": "Yes",
  "food": "Grass",
  "locomotion": "Legs"
}
```

```
MATCH (z:Animal
{name:"Zebra"})
REMOVE z.locomotion
RETURN z
```

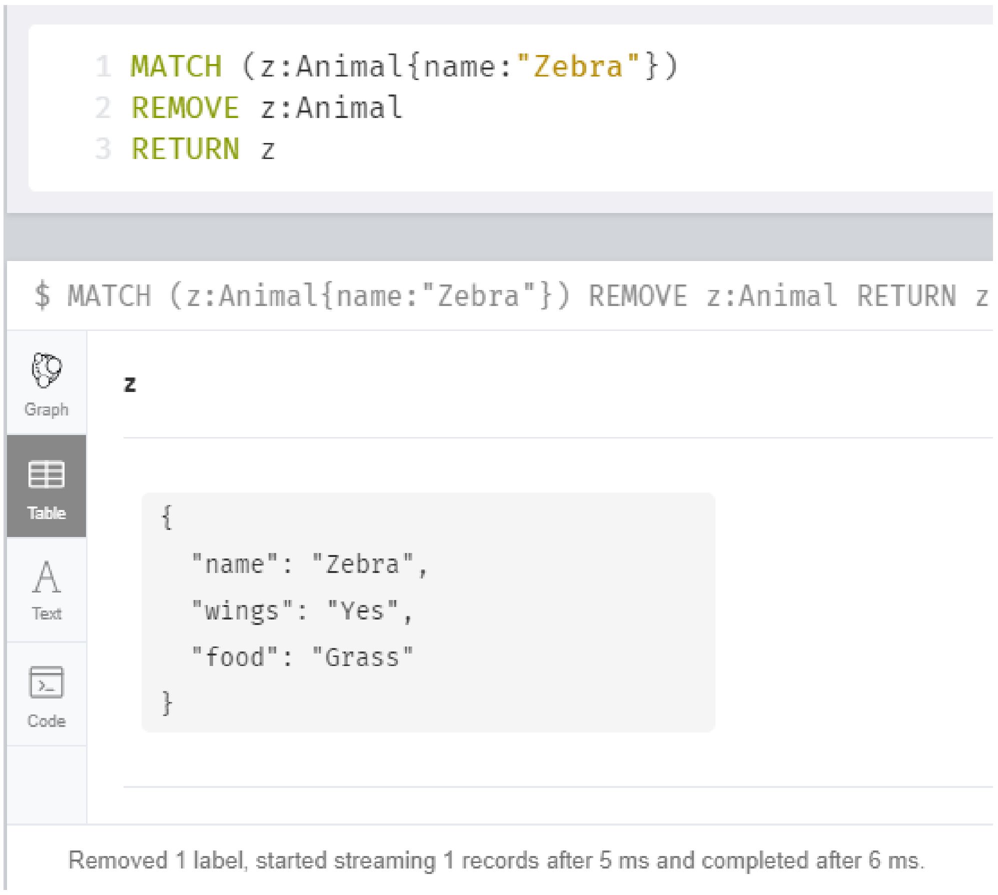


```
{
  "name": "Zebra",
  "wings": "Yes",
  "food": "Grass"
}
```

- Removing, adding or updating labels

Labels can be removed from nodes or relationships using REMOVE clause. However, we can still retrieve the node using existing properties.

For example, we can try removing the Animal label from Zebra.



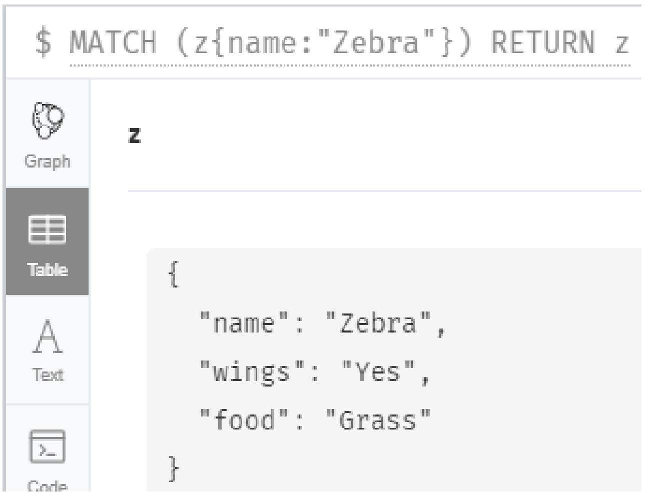
```
MATCH (z:Animal{name:"Zebra"})
REMOVE z:Animal
RETURN z
```

```
$ MATCH (z:Animal{name:"Zebra"}) REMOVE z:Animal RETURN z
```

```
z

{
  "name": "Zebra",
  "wings": "Yes",
  "food": "Grass"
}
```

Removed 1 label, started streaming 1 records after 5 ms and completed after 6 ms.

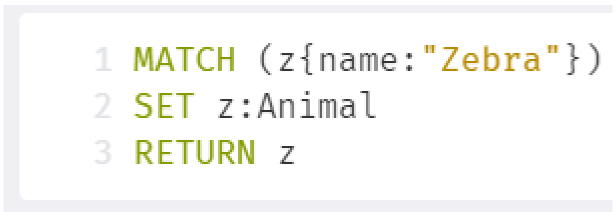
We can still retrieve the Zebra.



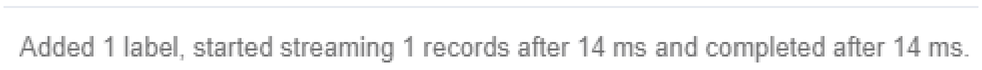
```
$ MATCH (z{name:"Zebra"}) RETURN z
```

```
z

{
  "name": "Zebra",
  "wings": "Yes",
  "food": "Grass"
}
```

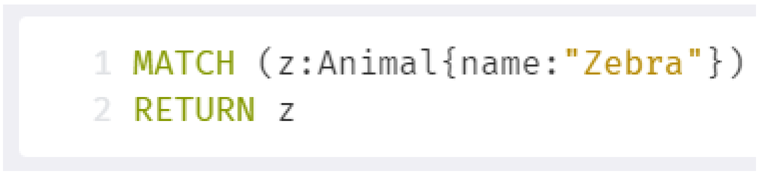
Labels can be added using SET clause and then verified using MATCH command. It is worth noting that labels cannot directly be updated using the SET clause and instead, need to be removed first using the REMOVE clause and then set in order to get updated.



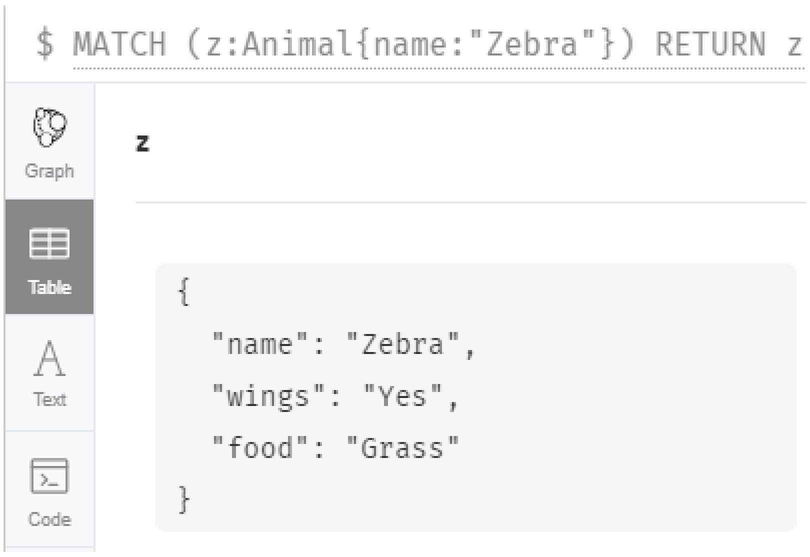
```
MATCH (z{name:"Zebra"})
SET z:Animal
RETURN z
```



Added 1 label, started streaming 1 records after 14 ms and completed after 14 ms.



```
MATCH (z:Animal{name:"Zebra"})
RETURN z
```



```
$ MATCH (z:Animal{name:"Zebra"}) RETURN z
```

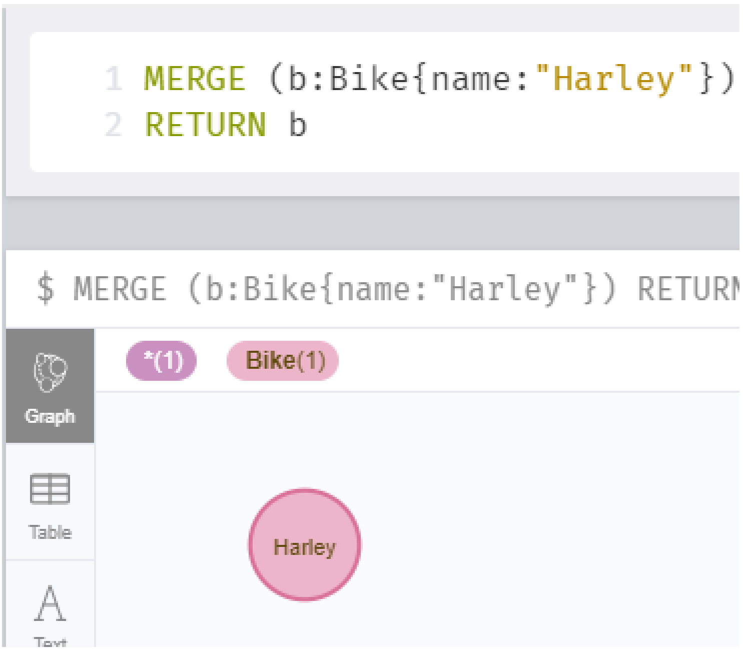
z

```
{
  "name": "Zebra",
  "wings": "Yes",
  "food": "Grass"
}
```

WEEK 6

- The MERGE operation.

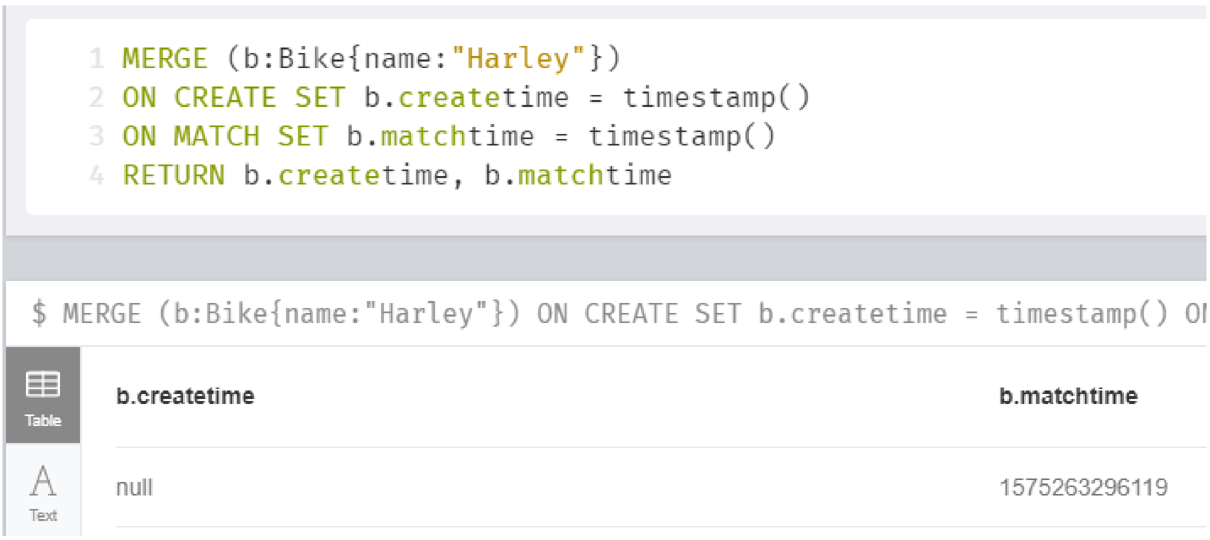
The MERGE command either creates an element if it does not exist or matches the existing pattern if the element exists. If there is no node available in the database, and MERGE command is attempted on a node, it creates the node. Running the command again does not create another node unlike the CREATE command.



```
MERGE (b:Bike{name:"Harley"})
RETURN b
```

```
$ MERGE (b:Bike{name:"Harley"}) RETURN
```

- ON CREATE SET and ON MATCH SET



```
MERGE (b:Bike{name:"Harley"})
ON CREATE SET b.createtime = timestamp()
ON MATCH SET b.matchtime = timestamp()
RETURN b.createtime, b.matchtime
```

```
$ MERGE (b:Bike{name:"Harley"}) ON CREATE SET b.createtime = timestamp() O
```

b.createtime	b.matchtime
null	1575263296119

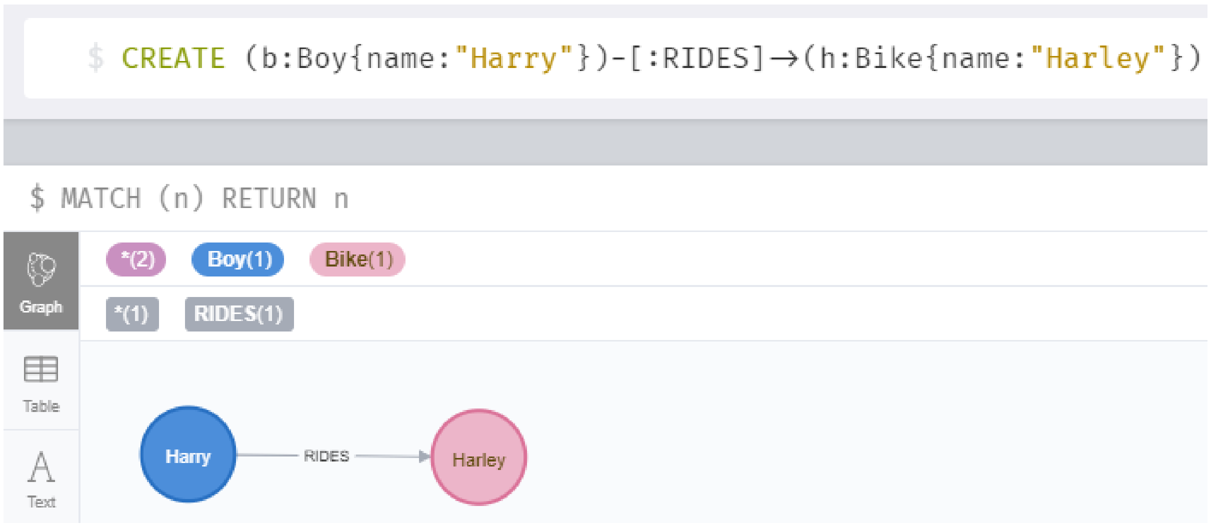
Here, the create time is null because we created the node already and the MERGE command found the node. If the node had not been created, it would be created and the match time would have been null, making the create time have some value.

- MERGE with relationships

Eight cases were discussed about using MERGE with relationships. These cases discussed individually about what the outcomes would be when using MERGE on different states or kinds of nodes. The command sometimes causes the nodes to be duplicated or have unnecessarily redundant data (for example: when nodes exist but no relationship exists, two new nodes are created and then the relationship added to them) and thus should be used by knowing the outcomes. Using MATCH and MERGE together tends to solve this problem and should be used to prevent duplication of nodes. Using MATCH also allows for bi-directional relationships.

- The eight cases:

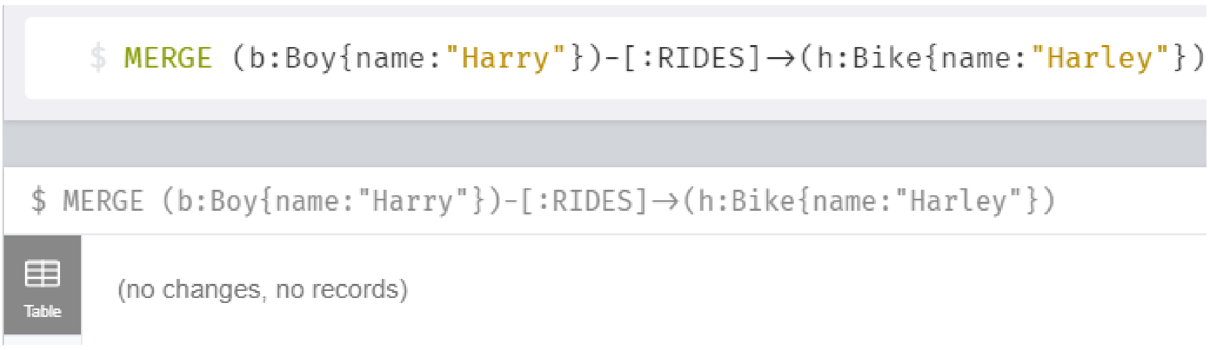
1. Two nodes exist and a relationship exists between them.



```
$ CREATE (b:Boy{name:"Harry"})-[:RIDES]→(h:Bike{name:"Harley"})
```

```
$ MATCH (n) RETURN n
```

Using merge in this state does not change anything.

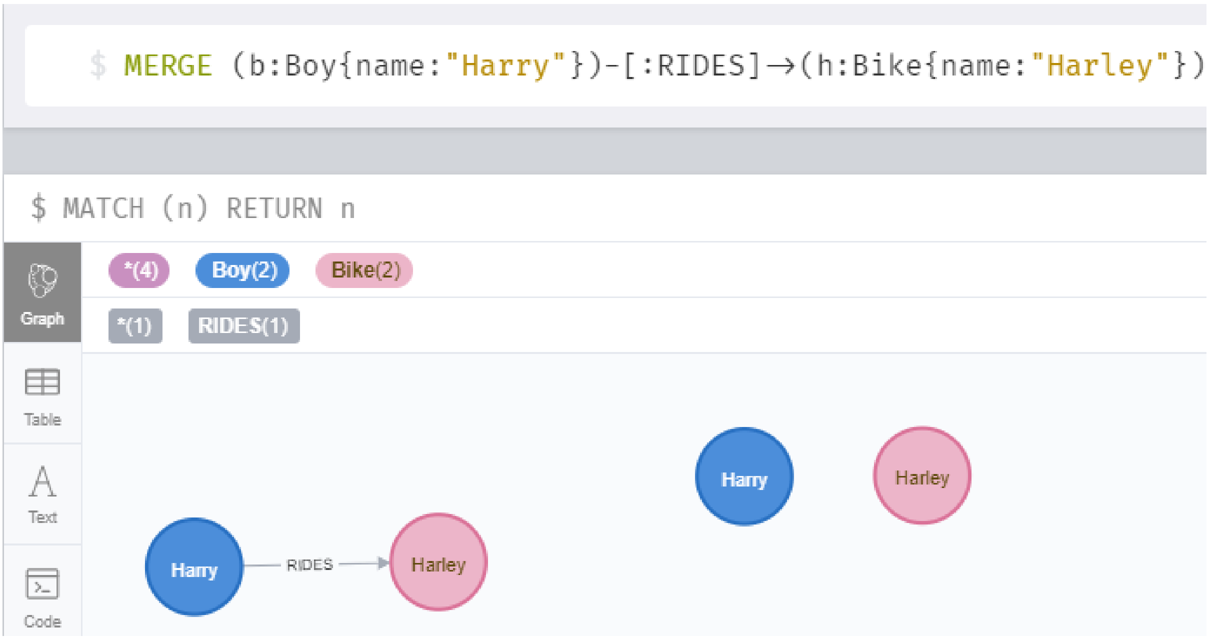


```
$ MERGE (b:Boy{name:"Harry"})-[:RIDES]→(h:Bike{name:"Harley"})
```

```
$ MERGE (b:Boy{name:"Harry"})-[:RIDES]→(h:Bike{name:"Harley"})
```

(no changes, no records)

2. Nodes exist but there is not a relationship between them.

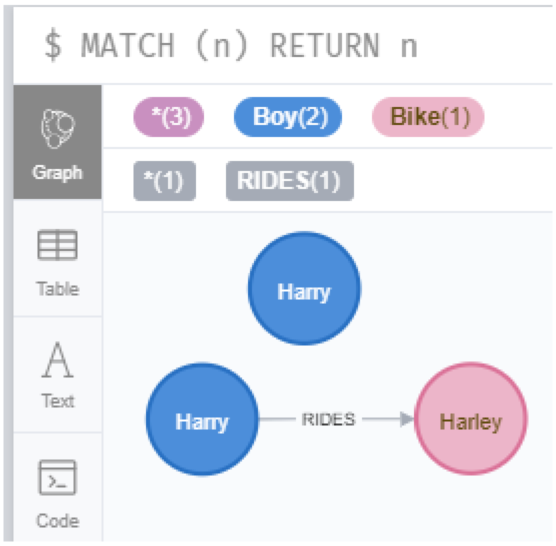


```
$ MERGE (b:Boy{name:"Harry"})-[:RIDES]→(h:Bike{name:"Harley"})
```

```
$ MATCH (n) RETURN n
```

The nodes are duplicated and then the relationship is added.

3. Single node exists and there is no relationship.



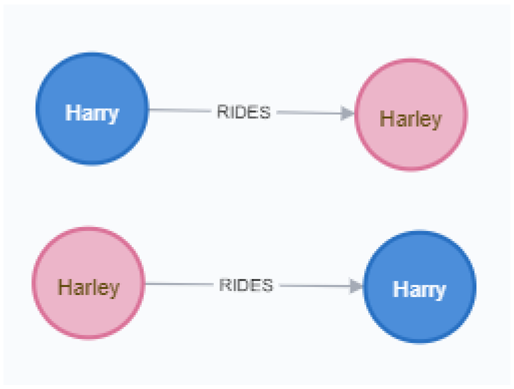
```
$ MATCH (n) RETURN n
```

The existing node is duplicated and then the relationship is created.

4. A node exists with UNIQUE constraint applied to it.

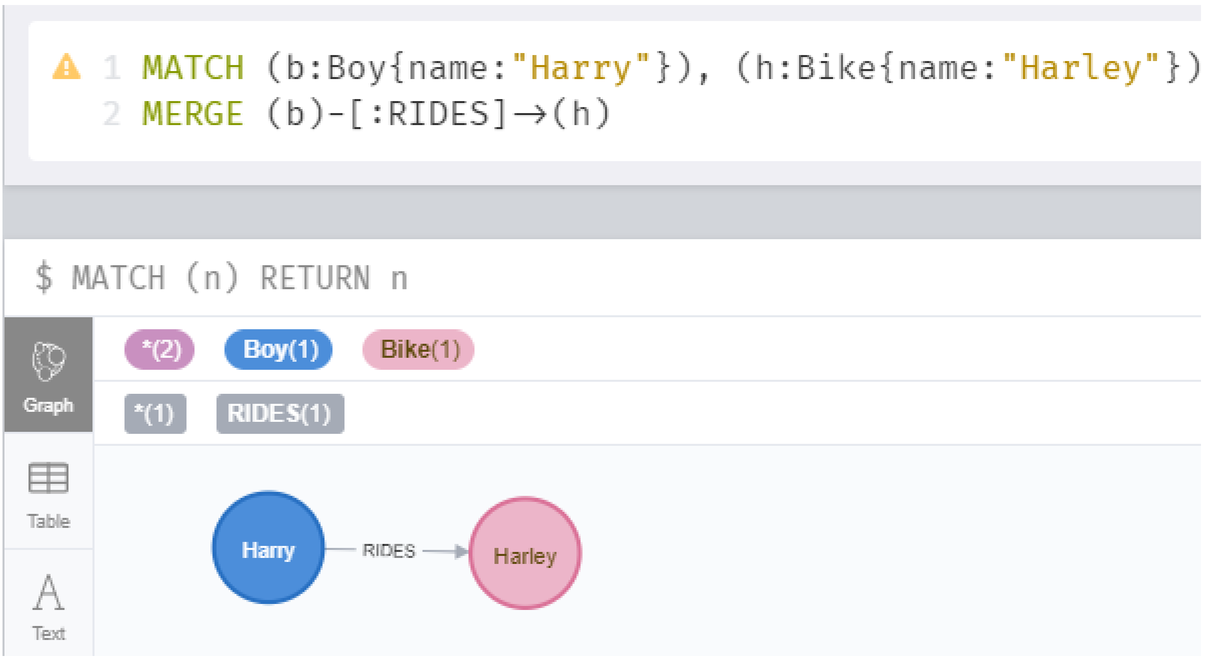
The duplication of nodes with unique constraints is not allowed and thus attempting this throws an error mentioning the failure to validate the unique constraint.

5. Two nodes and relationship between them exists and an attempt is made to add a direction towards opposite direction.



This creates two new nodes with the relationship towards the opposite direction.

6. Applying a MATCH to case 2.



```
MATCH (b:Boy{name:"Harry"}), (h:Bike{name:"Harley"})
MERGE (b)-[:RIDES]→(h)
```

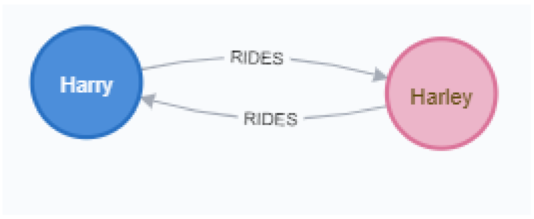
```
$ MATCH (n) RETURN n
```

This solves the issue of duplication from case 2.

7. Two nodes exist in a relationship and defining an un-directional relationship

This does not change anything. If there is a match found for a relationship, the command does not do anything.

8. Two nodes exist in a relationship and defining an opposite directional relationship (Attempting case 5 with MATCH)



This creates the relationship within the nodes unlike in case 5 where two new nodes were created.

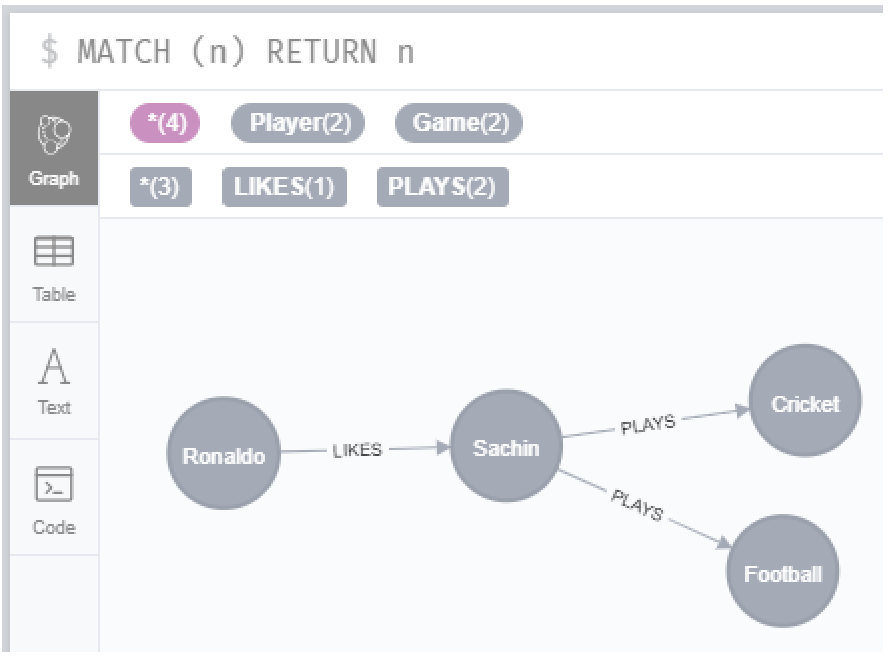
It can clearly be seen how use of MATCH can prevent duplication.

WEEK 7

- CREATE UNIQUE

The CREATE UNIQUE command should create what is missing, meaning it is like a mix of using both MATCH and CREATE commands at the same time. It generally requires less lines of code than MERGE commands and thus is faster to write. It is a general guidance to use MERGE command for preventing any nodes from duplicating and CREATE UNIQUE command for preventing any relationships from duplicating.

```
MERGE (p:Player {name: 'Ronaldo'})

MATCH (ap:Player {name: 'Ronaldo'})
CREATE UNIQUE (ap)-[r:LIKES]→(b:Player {name:'Sachin'})-
[r1:PLAYS]→(g:Game{name: "Cricket"})

MATCH (ap:Player {name: 'Ronaldo'})
CREATE UNIQUE (ap)-[r:LIKES]→(b:Player {name:'Sachin'})-
[r1:PLAYS]→(g:Game{name: "Football"})

MATCH (n) RETURN n
```



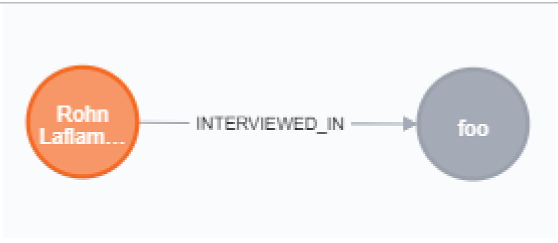
```
$ MATCH (n) RETURN n
```

By running the commands, two Player nodes are created who will have PLAYS relationship and LIKES relationship between each other. Using this command, no node or relationships are ever duplicated.

Running the provided code with some aspects modified.

```
CREATE (e:Episode{title:"foo"})
MERGE (p:Person{name:"Rohn Laflamme"})
ON CREATE SET p.mame = "Ron Laflamme",
p.state="CREATED"
ON MATCH SET p.state = "MATCHED"
CREATE UNIQUE (e)<-[:INTERVIEWED_IN]-(p)
```

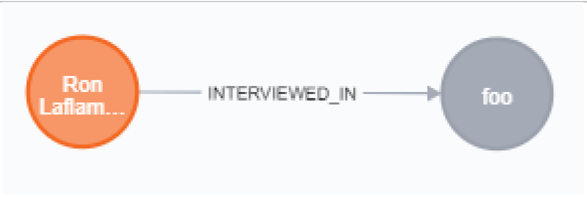
```
{
  "title": "foo"
}
```



```
{
  "name": "Ron Laflamme",
  "state": "CREATED"
}
```

Now running the other part of the code with existing Ron Laflamme node (instead of Lynn Rose).

```
MATCH (n) DETACH DELETE n

CREATE (p:Person{name:"Ron Laflamme"})
CREATE (e:Episode{title:"foo"})
MERGE (p:Person{name:"Ron Laflamme"})
MERGE (e:Episode{title:"foo"})
ON CREATE SET p.name = "Ron Laflamme", p.state = "CREATED"
ON MATCH SET p.state = "MATCHED"
CREATE UNIQUE (e)<-[:INTERVIEWED_IN]-(p)
```



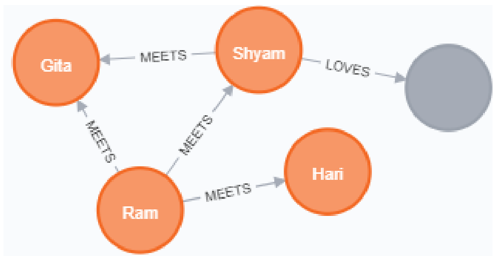
```
"name": "Ron Laflamme",
"state": "MATCHED"
```

Creating nodes if they are missing using the provided code

```
CREATE (:Person{name:"Ram"})
CREATE (:Person{name:"Shyam"})
CREATE (:Person{name:"Hari"})
CREATE (:Person{name:"Gita"})

MATCH (p1:Person{name:"Ram"}), (p2:Person{name:"Shyam"})
CREATE (p1)-[:MEETS]→(p2)

MATCH (p1:Person{name:"Ram"}), (p2:Person{name:"Hari"})
CREATE (p1)-[:MEETS]→(p2)

MATCH (p1:Person{name:"Ram"}), (p2:Person{name:"Gita"})
CREATE (p1)-[:MEETS]→(p2)

MATCH (p1:Person{name:"Shyam"}), (p2:Person{name:"Gita"})
CREATE (p1)-[:MEETS]→(p2)

MATCH (p1:Person{name:"Shyam"})
CREATE UNIQUE (p1)-[:LOVES]→(someone)
```

Verifying the New Node "someone"

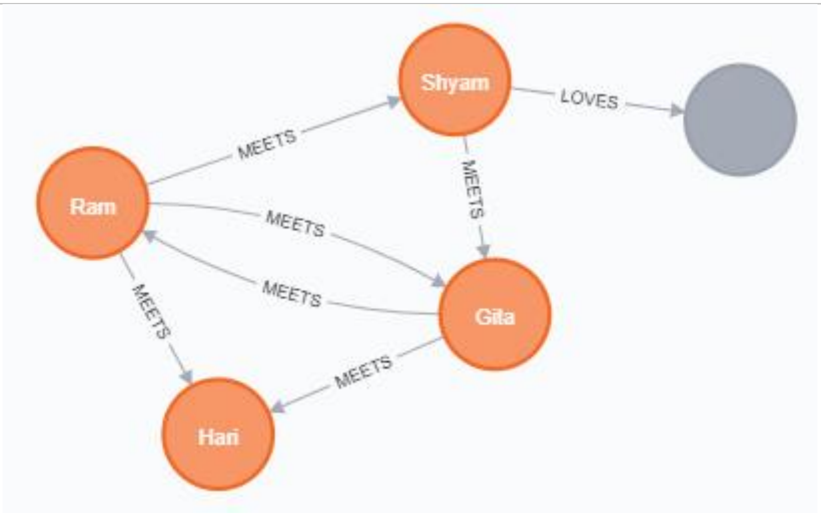
```
MATCH p=(n)-[:MEETS|LOVES]-()
RETURN p
```



The node in grey is the someone node that we created from the last line. Values can also be initialized to the newly created node for example by specifying the node label or properties while creating the new node. The someone can be replaced by b:Dog{name: "Bruno") which would create the LOVES relationship from Shyam to a Dog named Bruno.

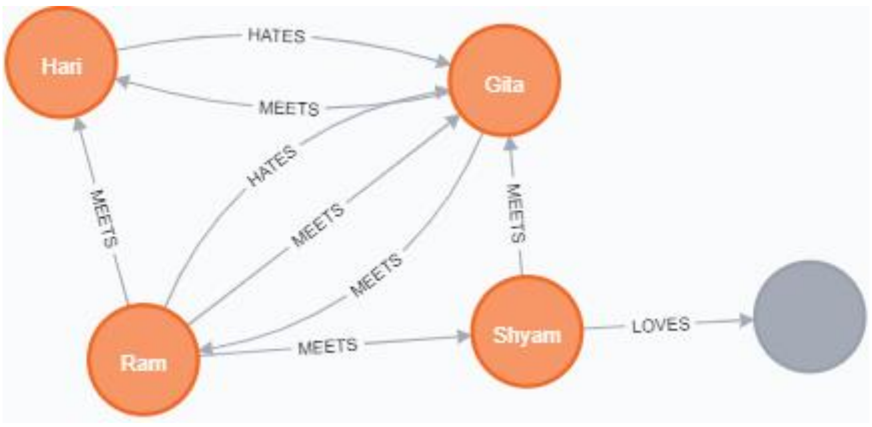
Similarly, relationships can also be created if they are missing like we did with nodes, labels along with properties. In the same dataset we created earlier (without replacing someone node to Dog), we can try to create a new relationship using CREATE UNIQUE.

```
MATCH (p:Person{name:"Gita"}), (rest)
WHERE rest.name IN ['Hari','Ram']
CREATE UNIQUE (p)-[rel:MEETS]→(rest)
```



New MEETS relationships are created from Gita to Hari and Ram.

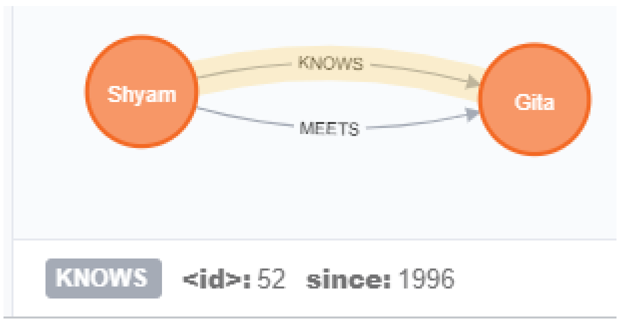
We can also create multiple relationships or nodes by separating them with a comma. For example, in the same command above, amending the third line to make it CREATE UNIQUE (p)-[rel:MEETS]->(rest), (rest)-[rel2:HATES]->(p) would create another two HATES relationships from Ram and Hari to Gita.



Exercise

The question has been amended to match the created nodes. Need to create a KNOWS relationship between Scott (replaced by Shyam) and Gary (replaced by Gita) with a property since:1996.

```
MATCH (s:Person{name:"Shyam"}), (g:Person{name:"Gita"})
CREATE UNIQUE (s)-[rel:KNOWS{since:1996}]→(g)
RETURN (s)-[rel]→(g)
```



```
KNOWS <id>: 52 since: 1996
```

- FOREACH

The bottom portion of this lesson contains information about the FOREACH clause. It is said that the command is used to update data in a list.

Exercises

The dataset we have used previously does not contain an attribute called marked. According to the question, we can add marked attributes to all the Person nodes connected to Ram or in terms of natural language, all the people that Ram meets.

```
MATCH p=(r:Person{name:"Ram"})-[:MEETS]→(e)
FOREACH (per IN nodes(p)|SET per.marked=TRUE)
```

```
{
  "marked": true,
  "name": "Ram"
}
```

```
{
  "marked": true,
  "name": "Shyam"
}
```

```
{
  "marked": true,
  "name": "Hari"
}
```

```
{
  "marked": true,
  "name": "Gita"
}
```

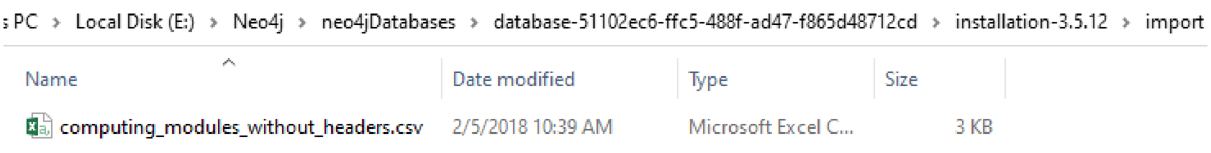
```
{

}
```

We can verify the results are working as expected as all the nodes that Ram had a MEETS relationship with have been updated to contain a property marked and set to true. While the other relationship KNOWS created with the blank “somebody” node was not updated. The FOREACH command iterates through all the found nodes and updates the property.

WEEK 8

As mentioned in the reflection, week 8 was mostly about loading data from CSV files and working with the previously learned commands on them. It also included some extra commands such as OPTIONAL MATCH.

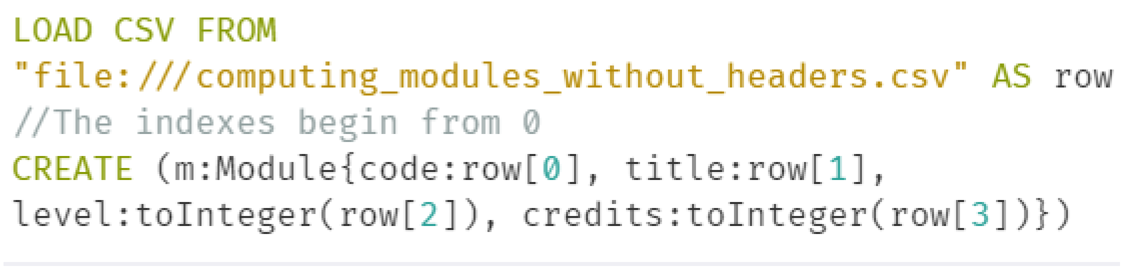
The first approach was to load the CSV file without using headers. The file computing_modules_without_headers.csv had to be imported to the database. The code was already provided.



s PC › Local Disk (E:) › Neo4j › neo4jDatabases › database-51102ec6-ffc5-488f-ad47-f865d48712cd › installation-3.5.12 › import

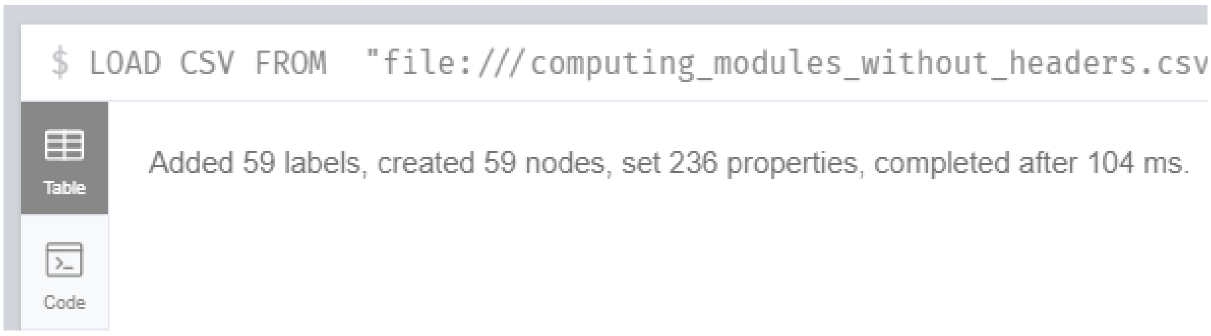
Name	Date modified	Type	Size
computing_modules_without_headers.csv	2/5/2018 10:39 AM	Microsoft Excel C...	3 KB

After moving the file to the relevant import directory, the provided command was run in the browser.



```
LOAD CSV FROM
"file:///computing_modules_without_headers.csv" AS row
//The indexes begin from 0
CREATE (m:Module{code:row[0], title:row[1],
level:toInteger(row[2]), credits:toInteger(row[3])})
```

This creates the Module nodes for every row of data that exists in the CSV file and hence, 59 nodes are created according to the 59 rows present in our CSV file.



```
$ LOAD CSV FROM  "file:///computing_modules_without_headers.csv
```

Added 59 labels, created 59 nodes, set 236 properties, completed after 104 ms.

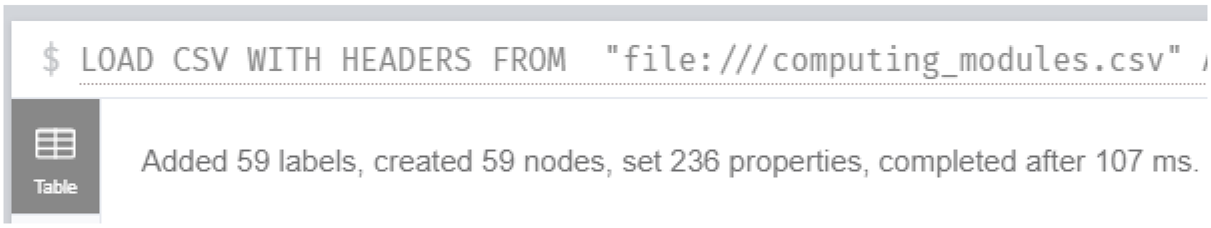
Using MATCH (n) RETURN n will display all of these freshly created nodes.

The next lesson was about importing files with headers. Headers are the information or names of each columns. We can directly refer to the header name instead of array indexes by using headers.

We are now loading a different CSV file on our clean database which contains headers.

```
LOAD CSV WITH HEADERS FROM
"file:///computing_modules.csv" AS row
//The indexes or keys are now headers of the CSV file
CREATE (m:Module{code:row.code, title:row.title,
level:toInteger(row.level), credits:toInteger(row.credits)})
```

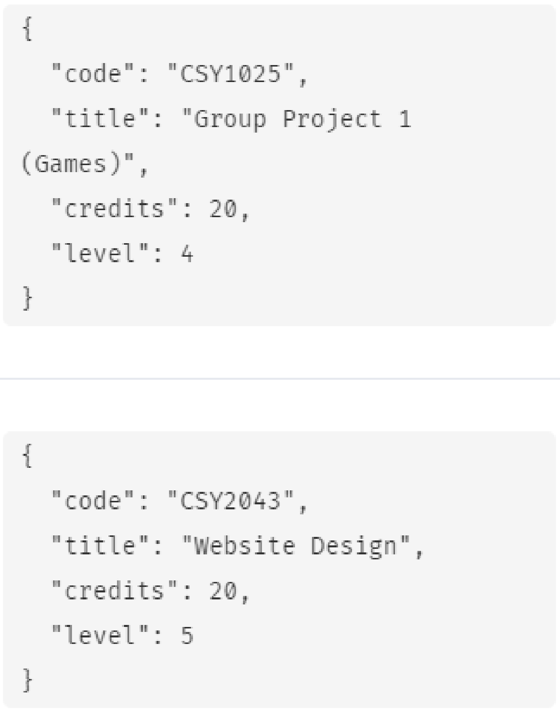
This also creates 59 nodes.



```
$ LOAD CSV WITH HEADERS FROM "file:///computing_modules.csv"
```

```
Added 59 labels, created 59 nodes, set 236 properties, completed after 107 ms.
```

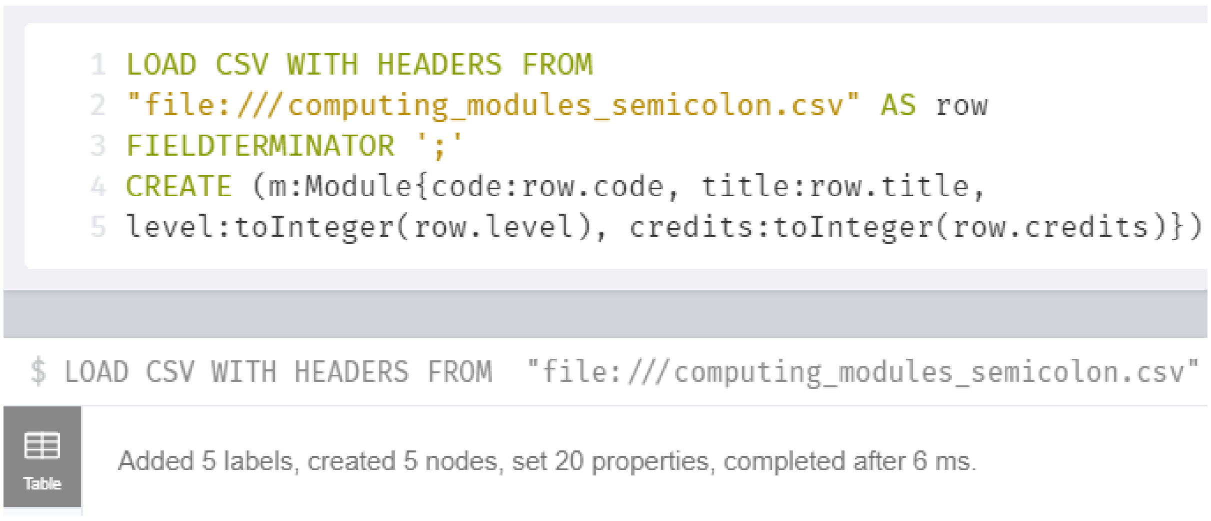
The properties and labels are set accordingly.



```
{
  "code": "CSY1025",
  "title": "Group Project 1
(Games)",
  "credits": 20,
  "level": 4
}
```

```
{
  "code": "CSY2043",
  "title": "Website Design",
  "credits": 20,
  "level": 5
}
```

We can also use a different field delimiter which has been set for the CSV file as some CSV files could contain a separate kind of delimiter than a comma and for example a space or semicolon. We can simply use FIELDTERMINATOR command followed by our delimiter.



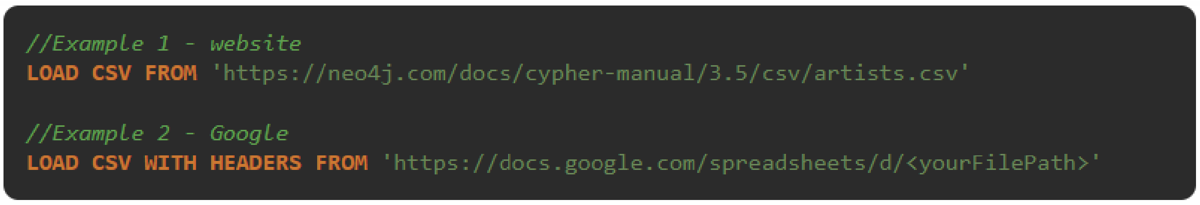
```
LOAD CSV WITH HEADERS FROM
"file:///computing_modules_semicolon.csv" AS row
FIELDTERMINATOR ';'
CREATE (m:Module{code:row.code, title:row.title,
level:toInteger(row.level), credits:toInteger(row.credits)})
```

```
$ LOAD CSV WITH HEADERS FROM  "file:///computing_modules_semicolon.csv"
```

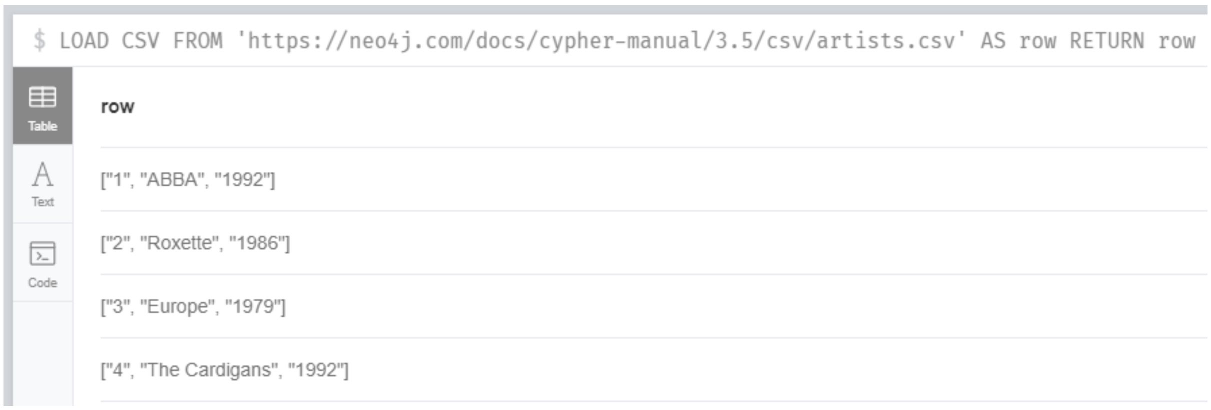
```
Added 5 labels, created 5 nodes, set 20 properties, completed after 6 ms.
```

Sometimes files such as XML and JSON have to be converted to a CSV file before loading them into the database. There are several online tools available that can convert these files into CSV files.

CSV files from online directories or URLs can also be loaded into the datasets. Different examples have been provided in the Neo4j documentation pages as well as in many other sources in the Internet. The screenshot below was taken from the Neo4j official guide to loading CSV files (Neo4j – Importing CSV Data into Neo4j, 2019). We can load similar files from the Internet directly. It is also mentioned that the permissions for using external files should be granted before loading the CSV files like this. More information regarding how to do this has also been given in another Neo4j official page with the Neo4j Sandbox. (Neo4j – Importing CSV Files Neo4j Desktop and Sandbox, 2019) The information regarding loading from different places such as GitHub, Google Drives and online websites is provided here.



```
//Example 1 - website
LOAD CSV FROM 'https://neo4j.com/docs/cypher-manual/3.5/csv/artists.csv'

//Example 2 - Google
LOAD CSV WITH HEADERS FROM 'https://docs.google.com/spreadsheets/d/<yourFilePath>'
```

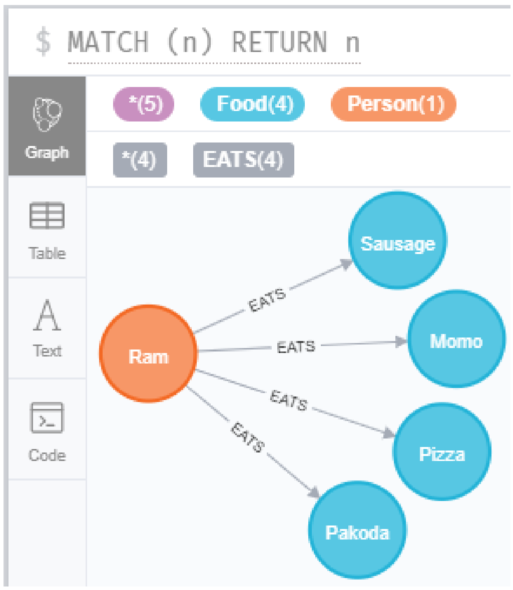


```
$ LOAD CSV FROM 'https://neo4j.com/docs/cypher-manual/3.5/csv/artists.csv' AS row RETURN row
```

row
["1", "ABBA", "1992"]
["2", "Roxette", "1986"]
["3", "Europe", "1979"]
["4", "The Cardigans", "1992"]

We can run the provided example and display the retrieved data by returning the rows.

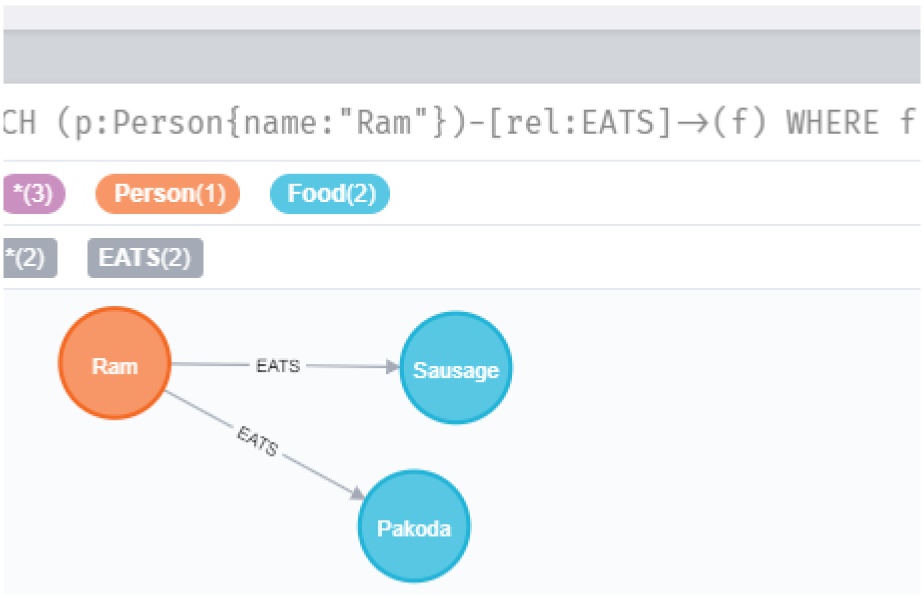
The latter part of this lesson contains more about the MATCH clause. The clause has been used previously in earlier lessons and is thus familiar. The MATCH clause is used for searching the data in database using provided information. As previously used, and extensively used in the assignment itself, it is very clear about the concepts of using MATCH clause. As provided in the chapter, MATCH clauses can be used to retrieve nodes by relationships or nodes or just simply retrieve all the nodes in the graph. We can further extend the usage of MATCH clause with the use of WHERE clause to filter the data to get more precise results, aggregate functions such as SUM to calculate SUM, MIN, MAX, AVG and COUNT to get required data, use the RETURN clause to display the data retrieved, use the ORDER BY clause to order the data in ascending or descending order according to specified column or use LIMIT clause to limit the number of rows displayed.



```
$ MATCH (n) RETURN n
```

A separate dataset has been created where Ram has EATS relationship with four different kinds of food. The foods have a property called number. This is not related to a more realistic result but done to highlight the usage of MATCH clause along with other operations such as WHERE or LIMIT.

```
MATCH (p:Person{name:"Ram"})-[rel:EATS]→(f)
WHERE f.number>4
RETURN p, rel, f
LIMIT 2
```



```
CH (p:Person{name:"Ram"})-[rel:EATS]→(f) WHERE f
```

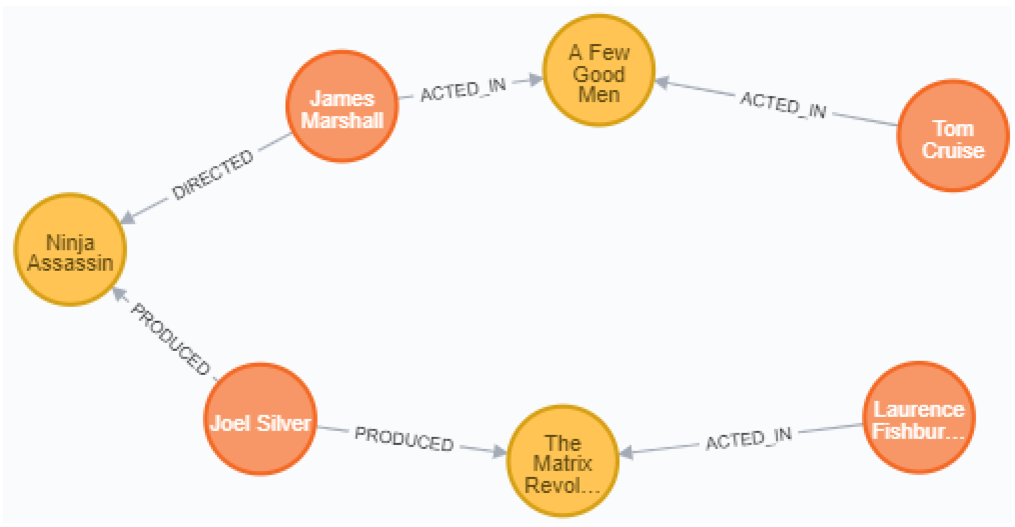
Here we are using the WHERE clause to restrict the results with only food that have number property value higher than 4 and only 2 foods or two relationships are displayed using the LIMIT clause. There could be several realistic uses of these commands.

Another function shortestPath can be used to find the shortest path between nodes. The exercises from this chapter require loading the movie database to test the shortest path function.

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

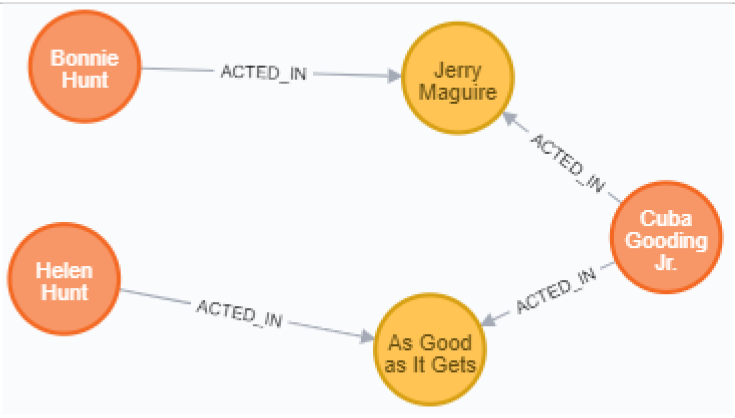
```
MATCH (tom:Person{name:"Tom Cruise"}),
(laurence:Person{name:"Laurence Fishburne"}),
p=shortestPath((tom)-[*..15]-(laurence))
RETURN p
```

The results are same as seen in the provided lesson.



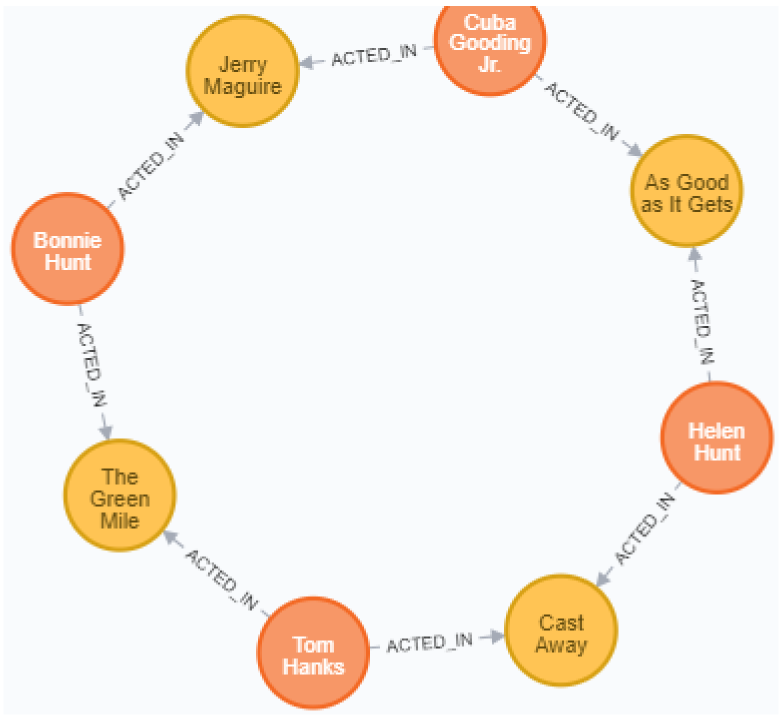
Similarly, using predicates (WHERE clause).

```
MATCH (helen:Person{name:"Helen Hunt"}),
(bonnie:Person{name:"Bonnie Hunt"}),
p=shortestPath((helen)-[*]-(bonnie))
WHERE NONE (r IN rels(p) WHERE type(r) = "MOTHER")
RETURN p
```



Finding all the shortest paths between two nodes.

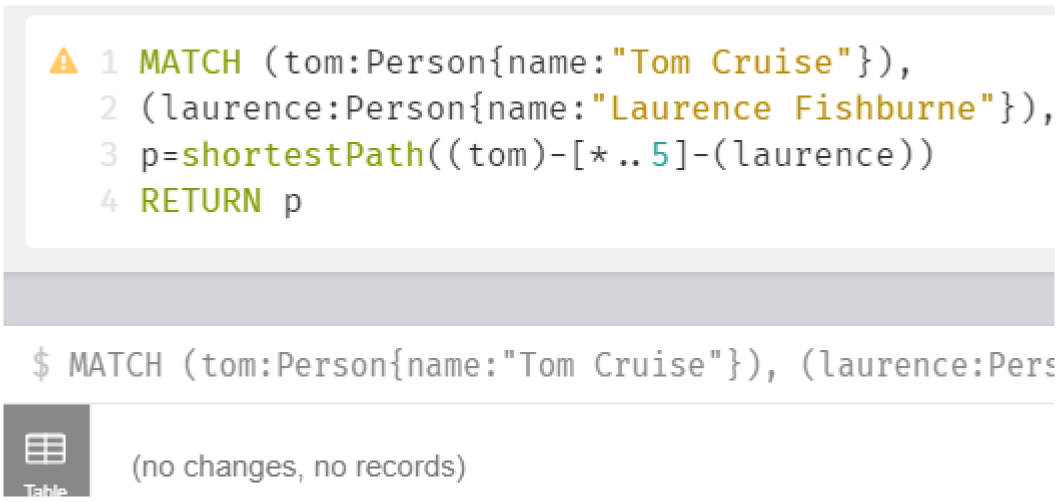
```
MATCH (helen:Person{name:"Helen Hunt"}),
(bonnie:Person{name:"Bonnie Hunt"}),
p=allShortestPaths((helen)-[*]-(bonnie))
RETURN p
```



• Make the following changes and run the codes again:

- Change the path lengths from "*..15" to "*..5" in the first "shortestPath" example

This displays no records because the path is longer than 5 and the restriction prevents searching for more than 5.



```
MATCH (tom:Person{name:"Tom Cruise"}),
(laurence:Person{name:"Laurence Fishburne"}),
p=shortestPath((tom)-[*..5]-(laurence))
RETURN p
```

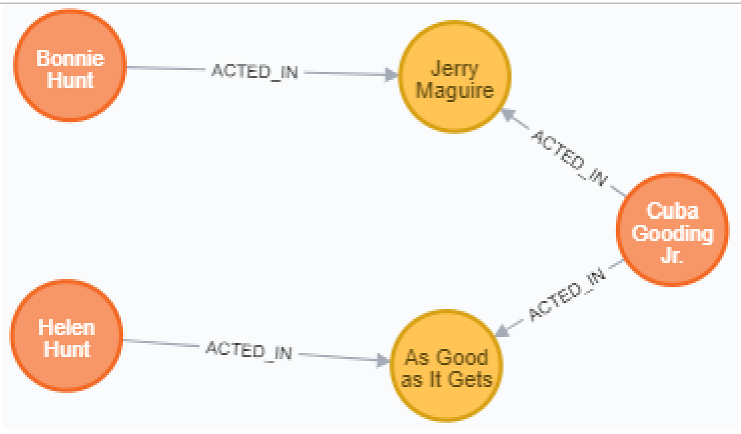
```
$ MATCH (tom:Person{name:"Tom Cruise"}), (laurence:Pers
```

Table

(no changes, no records)

- Remove the NONE command in the second "shortestPath()" example

```
MATCH (helen:Person{name:"Helen Hunt"}),
(bonnie:Person{name:"Bonnie Hunt"}),
p=shortestPath((helen)-[*]-(bonnie))
RETURN p
```



Finally, the OPTIONAL MATCH clause is discussed in this chapter. This clause is relatable to the OUTER JOIN in SQL wherein it returns NULL values if matches are not found.

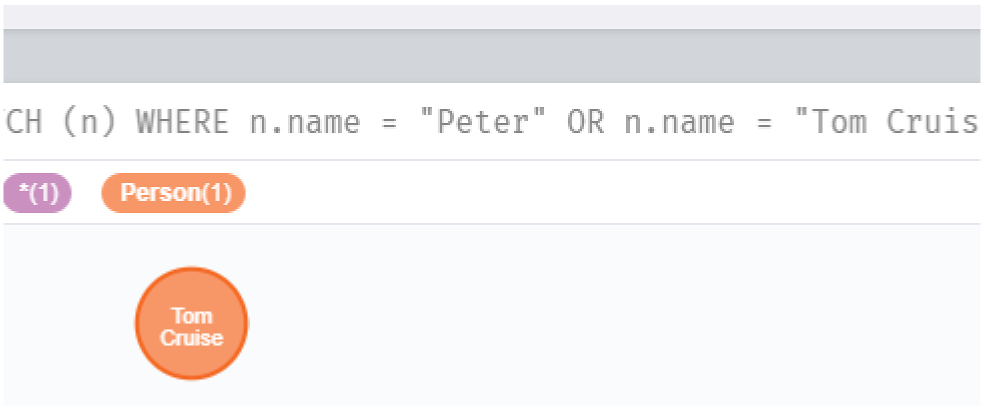
Without using OPTIONAL MATCH	
`MATCH (p{name:"Taylor Hackford"})` `MATCH (p)-[r:ACTED_IN]→()` `RETURN r`	(no changes, no records)
Using OPTIONAL MATCH	
`MATCH (p{name:"Taylor Hackford"})` `OPTIONAL MATCH` `(p)-[r:ACTED_IN]→()` `RETURN r`	r null

WEEK 10

As mentioned, Week 10 contains more details about using the WHERE clause. Many of these have already been implemented in the assignment and are familiar concepts. These are also similar to what used to be done in SQL for Database I and Database II.

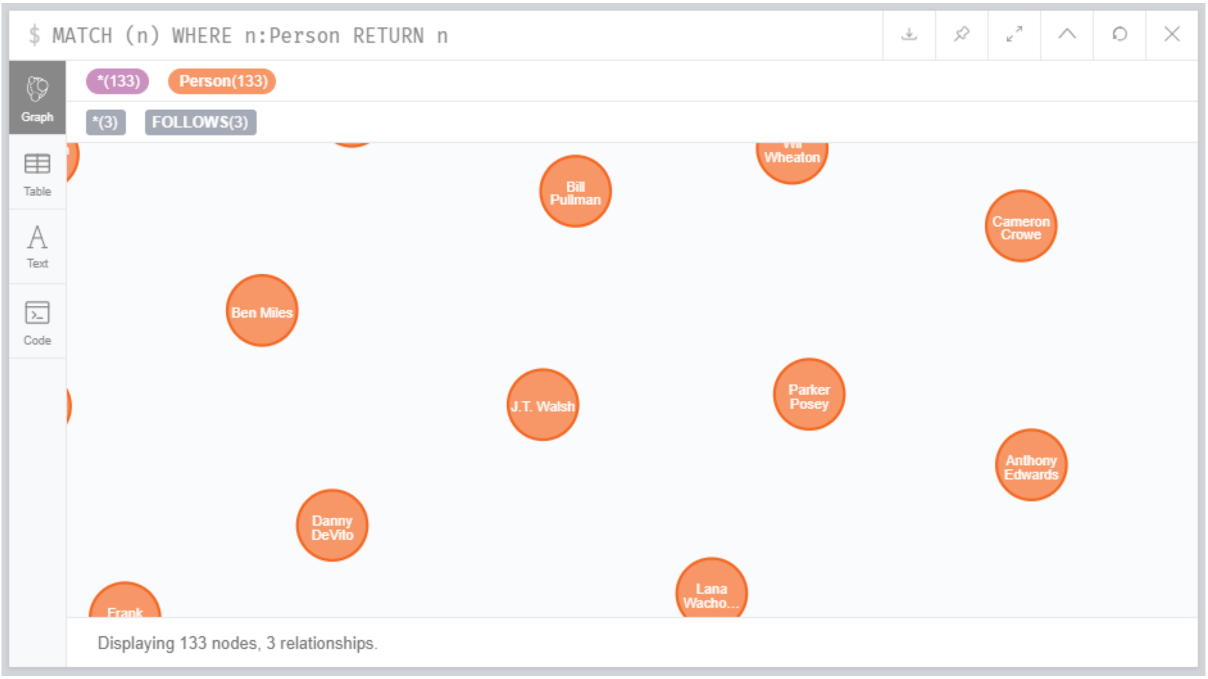
We can use AND, OR, NOT or other Boolean operators in the where clause like this.

```
MATCH (n)
WHERE n.name = "Peter" OR n.name = "Tom Cruise"
RETURN n
```



```
CH (n) WHERE n.name = "Peter" OR n.name = "Tom Cruis
```

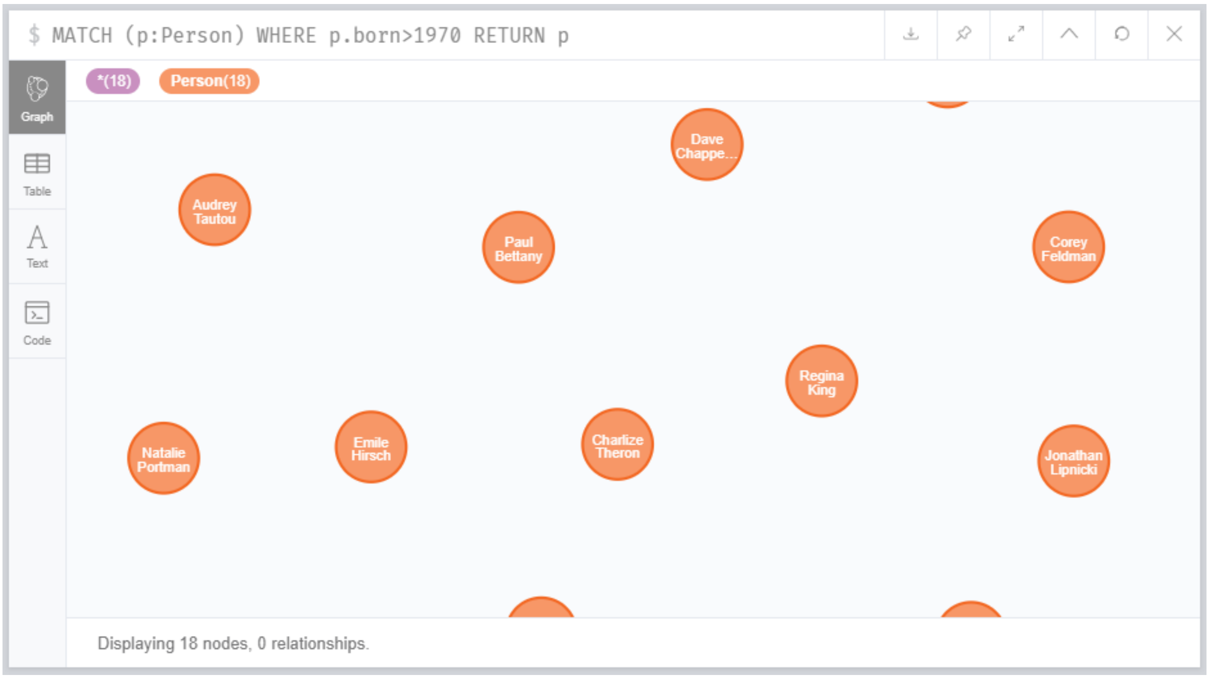
We can also filter the nodes from the where clause using Label.



```
$ MATCH (n) WHERE n:Person RETURN n
```

Displaying 133 nodes, 3 relationships.

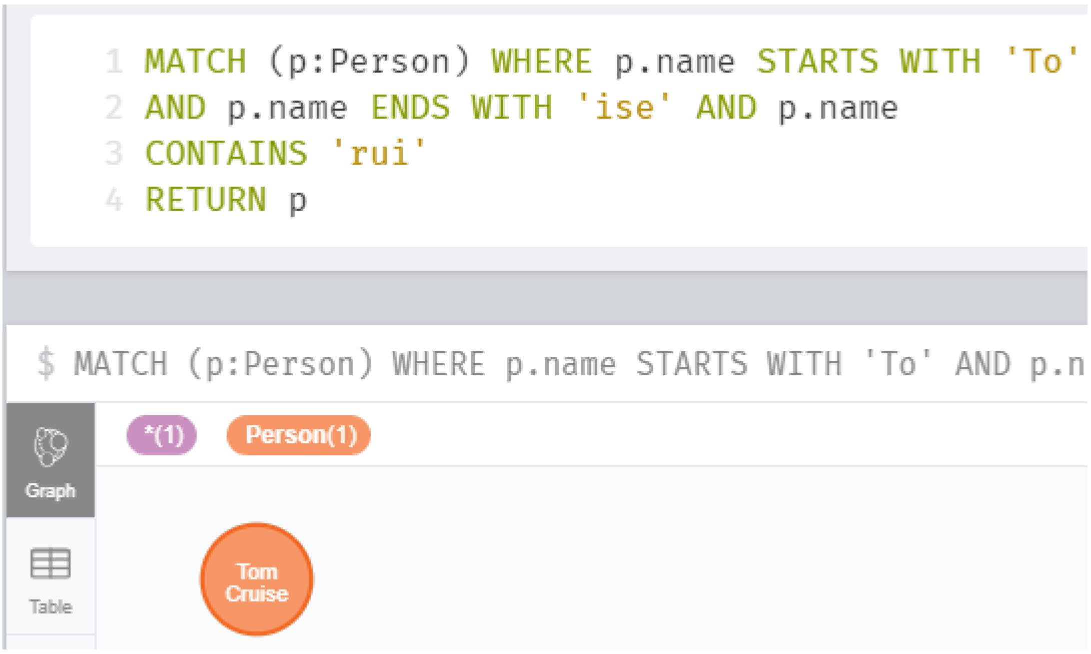
We can filter specific property values using the equals to, less than, greater than operators. The example below displays only the people in the movie database who were born later than 1970.



```
$ MATCH (p:Person) WHERE p.born>1970 RETURN p
```

Displaying 18 nodes, 0 relationships.

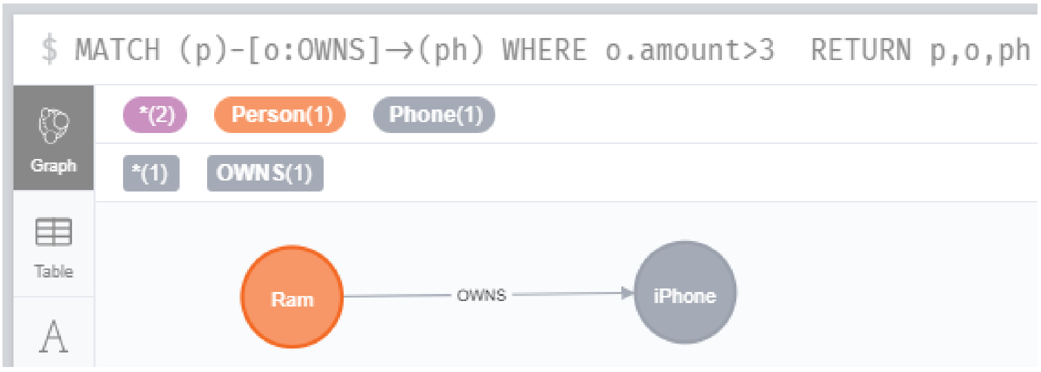
We can also match the start, end or characters contained in a string to get the desired output. Let’s try using a string that starts with To, ends with ise, and contains rui. The result should be Tom Cruise. Regular expressions can also be matched in strings using =~.



```
MATCH (p:Person) WHERE p.name STARTS WITH 'To'
AND p.name ENDS WITH 'ise' AND p.name
CONTAINS 'rui'
RETURN p
```

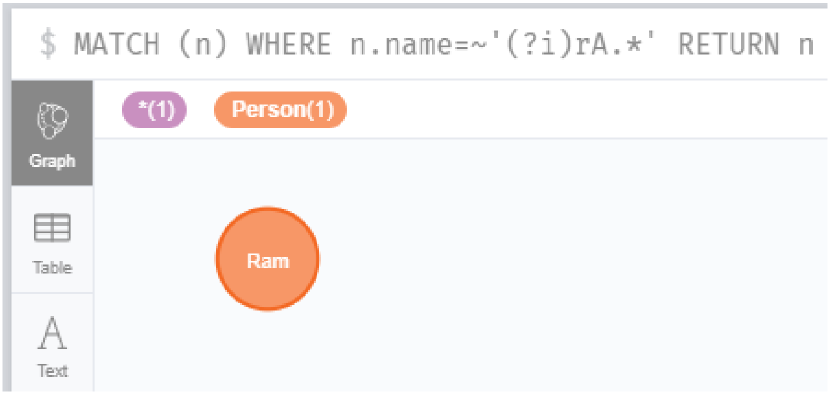
```
$ MATCH (p:Person) WHERE p.name STARTS WITH 'To' AND p.n
```

We can always use the NOT keyword to get opposite results of anything we query. We can also use the filtering on properties of any relationships between nodes.



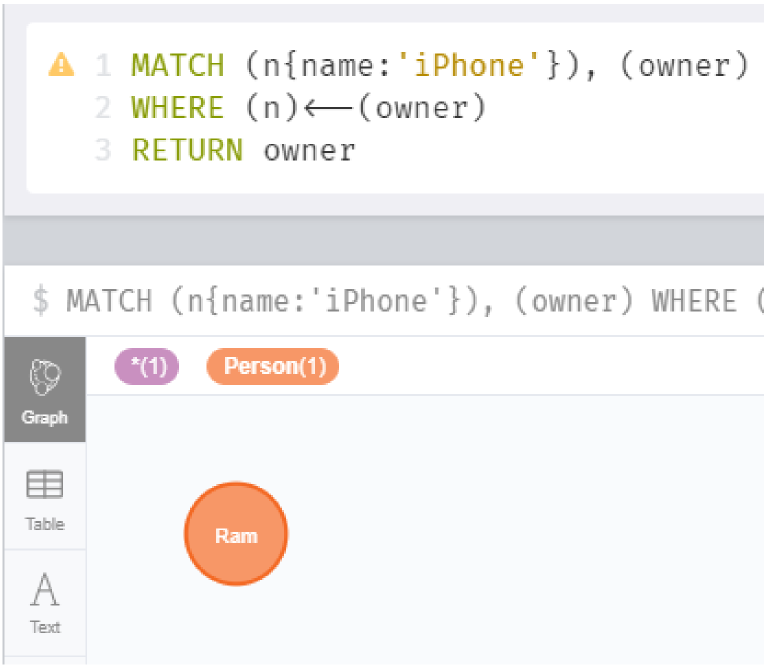
```
$ MATCH (p)-[o:OWNS]→(ph) WHERE o.amount>3  RETURN p,o,ph
```

Regular expressions can be used by =~. This allows case insensitive pattern matching in the provided string. Need to use (?i) in the regular expression.



```
$ MATCH (n) WHERE n.name=~'(?i)rA.*'  RETURN n
```

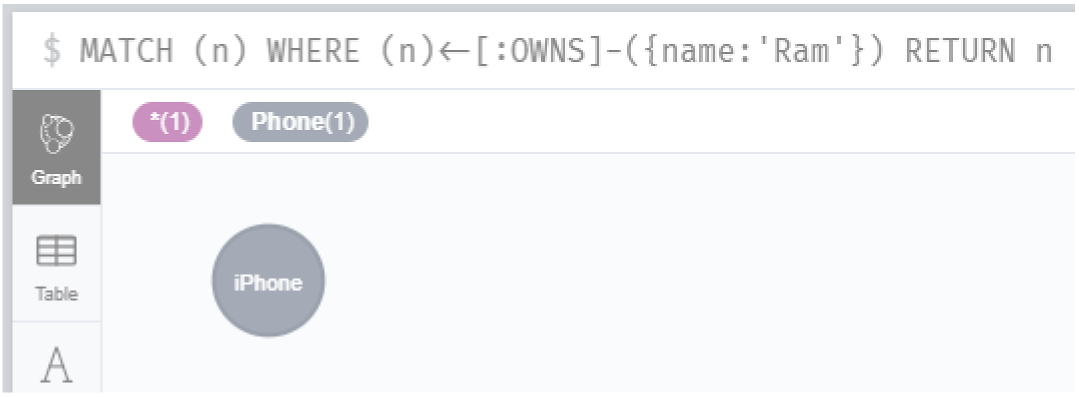
Using Path Patterns, we can return the owner of the iPhone.



```
MATCH (n{name:'iPhone'}), (owner)
WHERE (n)<--(owner)
RETURN owner
```

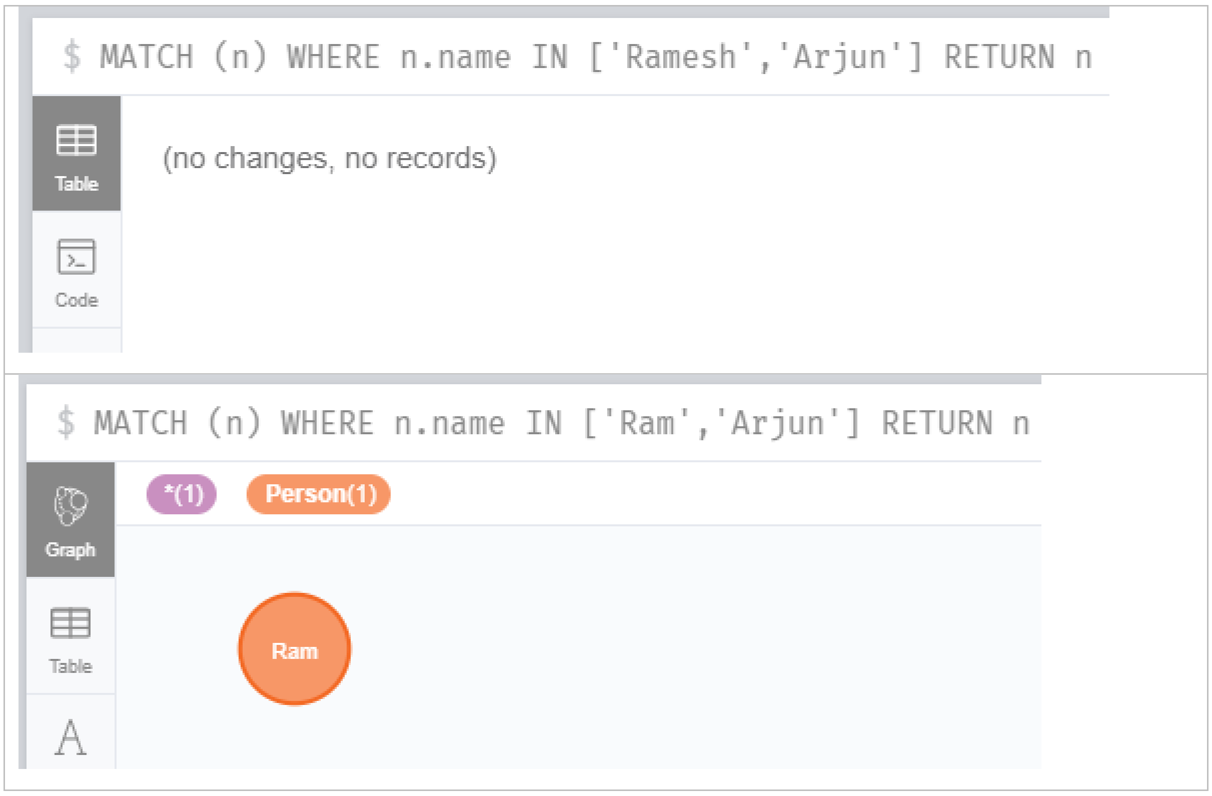
```
$ MATCH (n{name:'iPhone'}), (owner) WHERE (
```

Filtering relationship properties to return what Ram owns.



```
$ MATCH (n) WHERE (n)←[:OWNS]-({name:'Ram'}) RETURN n
```

We can use IN operator to check whether an element exists from the given list.

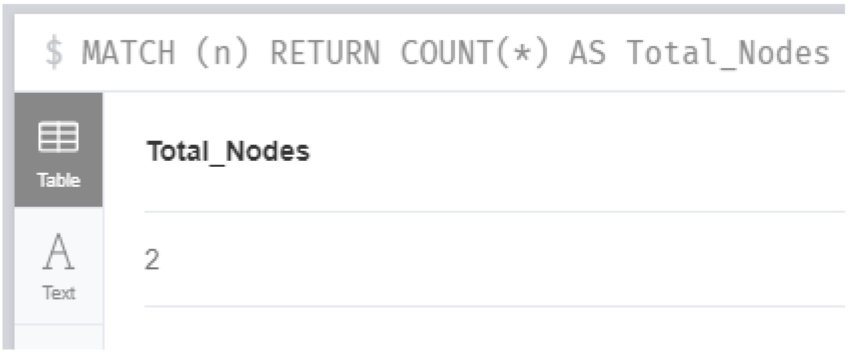


```
$ MATCH (n) WHERE n.name IN ['Ramesh','Arjun'] RETURN n
```

(no changes, no records)

```
$ MATCH (n) WHERE n.name IN ['Ram','Arjun'] RETURN n
```

Aggregate functions such as COUNT can be used to count the number of nodes.



```
$ MATCH (n) RETURN COUNT(*) AS Total_Nodes
```

Total_Nodes
2