

CSY3024 Database 3

AS1: Neo4j Database on EPL matches



January 17, 2024

university of northampton

Elizabeth Aning-Amponsah 18412818

Table of Contents

[Graph Database 2](#_Toc161535207)

[Schema 2](#_Toc161535208)

[Creation of graph database 2](#_Toc161535209)

[Queries 5](#_Toc161535210)

[References 16](#_Toc161535211)

# Graph Database

## Schema

A diagram of a football match

Description automatically generated

This schema has been carefully curated for querying to be much easier and straightforward rather than too complex. There are a total of 3 labelled nodes, one for team 1, one for team 2 and another for matches. As well as this, for each team node, has 3 types of relationships coming out of them being WON, LOST and DRAW. This allows for the easy identification of the result of the matches without having seen the scores. Additionally, the date from the csv file was originally stored in the format of ‘(Sat) 12 Aug 2017 (32)’ etc. but as shown in the schema, the plan is to split the date into integer values for easier querying, especially for queries such as most wins in January etc.

## Creation of graph database

The creation of the graph database is as follows:

Firstly, the creation of the team nodes by loading the data from the csv file as a row and merging the creation of these team nodes provided they do not already exist

LOAD CSV WITH HEADERS FROM "file:///epl\_mataches.csv" AS row

MERGE (team1:Team {name: row.Team1})

MERGE (team2:Team {name: row.Team2})

A screenshot of a computer

Description automatically generated

secondly the creation of the matches node as row, with round being converted into an integer value since the csv file stores it as an integer. The date is then converted and broken down into integer values using the split and case clauses and replacing the original values with digits rather. E.g. Jan is now stored as 1.

LOAD CSV WITH HEADERS FROM "file:///epl\_mataches.csv" AS row

CREATE (m:Matches {

round: toInteger(row.Round),

day: toInteger(split(split(row.Date, " ")[0], "(")[1]),

month: CASE split(split(row.Date, " ")[2], " ")[0]

WHEN 'Jan' THEN 1

WHEN 'Feb' THEN 2

WHEN 'Mar' THEN 3

WHEN 'Apr' THEN 4

WHEN 'May' THEN 5

WHEN 'Jun' THEN 6

WHEN 'Jul' THEN 7

WHEN 'Aug' THEN 8

WHEN 'Sep' THEN 9

WHEN 'Oct' THEN 10

WHEN 'Nov' THEN 11

WHEN 'Dec' THEN 12

END,

year: toInteger(split(row.Date, " ")[3]),

team1: row.Team1,

FT\_score: row.FT,

HT\_score: row.HT,

team2: row.Team2

})

A screenshot of a computer program

Description automatically generated

## Queries

1. Display the total number of matches played:

MATCH (m:Matches)

RETURN count(m) AS numberOfMatches

Match to find the node labelled Matches, whilst the return clause along with Count counts the number of rows in the Matches node and returns within the alias number of matches.

A screenshot of a computer

Description automatically generated

1. Display details of all matches involved “Manchester united FC”.

LOAD CSV WITH HEADERS FROM "file:///epl\_mataches.csv" AS row

WITH row WHERE row.Team1 = "Manchester United FC" OR row.Team2 = "Manchester United FC"

RETURN row;

Firstly, loading the csv file with headers and as row as the csv file contains headers. The With row WHERE row.team1 = Manchester or row.team2 line filters rows loaded from the csv file, only retaining those where the value of the column team1 and team2 equal to “Manchester united FC”. The with clause passes these filtered rows to the next query which is the return clause.

A screenshot of a computer

Description automatically generated

1. Display all teams that played the epl matches in the season.

MATCH (t:Team)

RETURN t.name

This query matches all the nodes in the graph labelled team and return them as its name property of each team.

A screenshot of a computer

Description automatically generated

1. Display the team with the most wins in January:

Firstly, create relationships of won, lost and draw – separately run the creation of each relationship

MATCH (m:Matches)

WHERE split(m.FT\_score, '-')[0] > split(m.FT\_score, '-')[1]

MATCH (team1:Team {name: m.team1}), (team2:Team {name: m.team2})

WITH m, team1, team2

MERGE (team1)-[w:WON]->(m)

MERGE (team2)-[l:LOST]->(m)

SET w.scoreAtHalfTime = toInteger(split(m.HT\_score, '-')[0]),

w.scoreAtFullTime = toInteger(split(m.FT\_score, '-')[0]),

l.scoreAtHalfTime = toInteger(split(m.HT\_score, '-')[1]),

l.scoreAtFullTime = toInteger(split(m.FT\_score, '-')[1])

MATCH (m:Matches)

WHERE split(m.FT\_score, '-')[0] < split(m.FT\_score, '-')[1]

MATCH (team1:Team {name: m.team1}), (team2:Team {name: m.team2})

WITH m, team1, team2

MERGE (team1)-[l:LOST]->(m)

MERGE (team2)-[w:WON]->(m)

SET l.scoreAtHalfTime = toInteger(split(m.HT\_score, '-')[0]),

l.scoreAtFullTime = toInteger(split(m.FT\_score, '-')[0]),

w.scoreAtHalfTime = toInteger(split(m.HT\_score, '-')[1]),

w.scoreAtFullTime = toInteger(split(m.FT\_score, '-')[1])

MATCH (m:Matches)

WHERE split(m.FT\_score, '-')[0] = split(m.FT\_score, '-')[1]

MATCH (team1:Team {name: m.team1}), (team2:Team {name: m.team2})

WITH m, team1, team2

MERGE (team1)-[d:DRAW]->(m)

MERGE (team2)-[d2:DRAW]->(m)

SET d.scoreAtHalfTime = toInteger(split(m.HT\_score, '-')[0]),

d.scoreAtFullTime = toInteger(split(m.FT\_score, '-')[0]),

d2.scoreAtHalfTime = toInteger(split(m.HT\_score, '-')[1]),

d2.scoreAtFullTime = toInteger(split(m.FT\_score, '-')[1])

Now to display the most the team with the most wins in Jan:

MATCH (m:Matches)

WHERE m.month = 1 AND split(m.FT\_score, '-')[0] > split(m.FT\_score, '-')[1]

MATCH (team1:Team {name: m.team1})

RETURN team1.name AS Team, count(team1) AS Wins

ORDER BY Wins DESC

LIMIT 1

A screenshot of a computer

Description automatically generated

The first part of the query creates relationships between teams and matches, representing the outcome (win, lose, draw) of each match. It also sets properties for these relationships based on the score at halftime and fulltime using the split function given that it’d be allowing for easier querying since the scores as stored with a hyphen in between each score in the csv file e.g. 0-3.

Whilst the second part of the query calculates the number of wins for each team in January by matching matches where the first teams score is greater than the second teams score, indicating a win for the first team and vice versa. it then counts it using count as wins and then the result is ordering the rows in descending order and limits it to only show one record.

1. Display the top five teams with the best scoring power

MATCH (team:Team)-[r]-(m:Matches)

RETURN team.name AS Team, SUM(r.scoreAtFullTime) AS TotalGoals

ORDER BY TotalGoals DESC

LIMIT 5

This query matches patterns where there is a relationship ‘r’ between a ‘team node and a match node. The direction is not specified so it takes into consideration all relationships whether it is incoming or outgoing.

The return team.name As sum(r.scoreAtFullTime) AS TotalGoals, specifies which data to return. It returns the name property of the team node as team and calculates the total number of goals scored in all matches and stores in the alias TotalGoals. Again, it is ordered by descending order but this time limits to five to only display first 5 records.

A screenshot of a computer

Description automatically generated

1. Display the top five teams that have the best winning records.

MATCH (team:Team)-[result:WON]->(:Matches)

WITH team, COUNT(result) AS Wins

RETURN team.name AS Team, Wins

ORDER BY Wins DESC

LIMIT 5

This query matches all teams that have won matches and counts the teams wins and stores it in the alias wins, it then returns the teams name property as team and also the count of the teams wins. It is then ordered in descending order and limited to five to show the top 5 records, meaning the teams with the highest number of wins will appear first.

A screenshot of a computer

Description automatically generated

1. Display the top 5 teams with the worst defending .

MATCH (team:Team)-[r1]->(:Matches)<-[r2]-(team2:Team)

RETURN team.name, sum(r2.scoreAtFullTime) AS goalsByOppTeam

ORDER BY goalsByOppTeam DESC

LIMIT 5

This query matches all teams that have a relationship to matches, it then returns the team by its name property whilst calculating the sum of the scoreAtFullTime property of all incoming relationships ‘r2’ for each team. This sum represents the total goals scored against each team, indicating their defensive performance. Whilst the orderby orders the rows in descending order and is limited to 5 meaning the teams with highest number of goals scored against them will appear first.

A screenshot of a computer

Description automatically generated

1. Display the top five teams with the best half-time result

MATCH (team:Team)-[result:WON|LOST|DRAW]->(m:Matches)

WHERE result.scoreAtHalfTime IS NOT NULL

WITH team, SUM(result.scoreAtHalfTime) AS GoalsScored

RETURN team.name AS Team, GoalsScored

ORDER BY GoalsScored DESC

LIMIT 5

This query matches all patterns where a team node is connected to a matches node via. A relationship type labelled either WON, LOST or DRAW. Once captured the WHERE line filters the matches to include only those where the scoreAtHalfTime property is recorded are considered. The WITH team line aggregates the halftime scores for each team by summing up the scoreAtHalfTime property of the result relationship and then returned under the alias goalsScored. Order by and DESC to arrange the rows in descending order, meaning teams with the highest total half-time scores will appear first. This is finally limited to five to only show the top 5 results.

A screenshot of a computer

Description automatically generated

1. Display the teams with the most loss.

MATCH (team:Team)-[result:LOST]->(m:Matches)

RETURN team.name AS Team, COUNT(result) AS Losses

ORDER BY Losses DESC

LIMIT 5

This query first matches patterns where a team node is connected to a matches node via a relationship labelled LOST. This is captured and then returned as the name property of the team node as team along with the count of lost relationships (result) connected to each team, which represents to number of losses. Order by and DESC orders the rows in descending order and then it is limited to 5 to display only the top 5 teams with the most losses.

A screenshot of a computer

Description automatically generated

1. Display the team with the most consecutive wins.

MATCH (team:Team)-[:WON]->(m1:Matches)

WHERE NOT (team)-[:LOST]->(:Matches {round: m1.round + 1}) AND NOT (team)-[:DRAW]->(:Matches {round: m1.round + 1})

WITH team, count(m1) AS consecutiveWins

RETURN team.name AS Team, max(consecutiveWins) AS MaxConsecutiveWins

ORDER BY MaxConsecutiveWins DESC

LIMIT 1

This final query, it matches patterns where a team node Is connected to a matches node labelled WON. The second line in the query filters the matches to ensure that there are no matches where the team lost or drew, ensuring that the consecutive win streak is uninterrupted. The WITH team line follows the filtering and counts the number of consecutive wins (m1) for each team and assigns it to the variable consecutiveWins. The return line then returns team name and the maximum count of consecutive wins among all teams. The order by and desc function orders the result by the maximum consecutive wins in descending order, meaning the team with the longest consecutive win streak will appear first. This is then limited to only show one row out of all the records provided.

A screenshot of a computer

Description automatically generated

# References

1. www.graphable.ai. (2022). *What is Neo4j (Graph Dabatase)? Complete Overview of Neo4j*. [online] Available at: <https://www.graphable.ai/software/what-is-neo4j-graph-database/>.
2. Shukla, D. (2024). *Top 9 Open Source Graph Databases*. [online] Analytics Vidhya. Available at: https://www.analyticsvidhya.com/blog/2024/01/top-9-open-source-graph-databases/#:~:text=Neo4j%3A%20Neo4j%20is%20one%20of [Accessed 17 Mar. 2024].
3. Google Cloud. (n.d.). *Run JanusGraph on GKE with Bigtable*. [online] Available at: https://cloud.google.com/bigtable/docs/running-janusgraph-with-bigtable [Accessed 17 Mar. 2024].
4. janusgraph.org. (n.d.). *JanusGraph*. [online] Available at: https://janusgraph.org/.
5. Nearshore Software Development Company - IT Outsourcing Services. (2019). *Neo4j – an invitation to graph databases.* [online] Available at: https://www.nearshore-it.eu/articles/technologies/neo4j-invitation-to-graph-databases/ [Accessed 17 Mar. 2024].

‌