Databases 3 Assignment 1

20410488

Ștefan-Felician Bãloiu

Contents

[Introduction 1](#_Toc130764804)

[Analysis and Design 1](#_Toc130764805)

[Creation 2](#_Toc130764806)

[Summary 3](#_Toc130764807)

[Queries 5](#_Toc130764808)

[Conclusion 15](#_Toc130764809)

[References 16](#_Toc130764810)

# Introduction

Neo4j is a graph database management system that is designed to store, manage, and query complex data that uses nodes and the Cypher query language to query and manipulate graph data. It is good at finding patterns in that data and is often used in large companies and social media.

In this assignment a dataset related to the English Premier League (EPL) has been provided since 2000. The objectives are to analyse the given dataset, design an appropriate graph model, create a graph database to store information in the dataset and to perform queries using Cypher Query Language.

The aim is to develop a functional graph database that can efficiently execute the given queries.

# Analysis and Design

The dataset provided for the English Premier League (EPL) contains the following information:

* Details regarding each match played in the EPL, including the season in which the match was played, the date and time of the match, and the two participating teams, with one team playing as the home team and the other as the away team.
* The number of goals scored by each team during the match, with FTHG and FTAG representing the number of goals scored by the home and away teams, respectively.
* The outcome of the match, categorized as either a home win (H), a draw (D), or an away win (A).
* The number of goals scored by each team during the first half of the match, with HTHG and HTAG representing the number of goals scored by the home and away teams, respectively.
* The half-time result of the match, noted as a win for either the home team (H), the away team (A), or a draw (D).

The database will have the following nodes and properties:

* Teams
  + name
* Matches
  + matchName
  + dateTime
  + full\_time\_home\_team\_goals
  + full\_time\_away\_team\_goals
  + full\_time\_result
  + half\_time\_home\_team\_goals
  + half\_time\_away\_team\_goals
  + half\_time\_result
  + home\_team\_shots
  + away\_team\_shots
  + home\_team\_shots\_on\_target
  + away\_team\_shots\_on\_target
  + home\_team\_corners
  + away\_team\_corners
  + home\_team\_fouls\_committed
  + away\_team\_fouls\_committed
  + home\_team\_yellow\_cards
  + away\_team\_yellow\_cards
  + home\_team\_red\_cards
  + away\_team\_red\_cards
* Seasons
  + season
* Referees
  + Referee

The relationships will be:

Home teams and away teams will be part of a match ( PLAYED\_HOME, PLAYED\_AWAY), a referee will be part of a match (REFEREED) and a match will be part of a season(PART\_OF). The schema is displayed in the following figure 4.

# Creation

LOAD CSV WITH HEADERS FROM "file:///EPL\_matches.csv" AS row

MERGE (h:Teams{name:row.HomeTeam})

MERGE (a:Teams{name:row.AwayTeam})

MERGE (s:Seasons{season:row.Season })

MERGE (m:Matches{matchName:"Match", dateTime:dateTime(row.DateTime), full\_time\_home\_team\_goals:toInteger(row.FTHG), full\_time\_away\_team\_goals:toInteger(row.FTAG), full\_time\_result:row.FTR, half\_time\_home\_team\_goals:toInteger(row.HTHG), half\_time\_away\_team\_goals:toInteger(row.HTAG), half\_time\_result:row.HTR, home\_team\_shots:toInteger(row.HS), away\_team\_shots:toInteger(row.AS), home\_team\_shots\_on\_target:toInteger(row.HST), away\_team\_shots\_on\_target:toInteger(row.AST), home\_team\_corners:toInteger(row.HC), away\_team\_corners:toInteger(row.AC), home\_team\_fouls\_committed:toInteger(row.HF), away\_team\_fouls\_committed:toInteger(row.AF), home\_team\_yellow\_cards:toInteger(row.HY), away\_team\_yellow\_cards:toInteger(row.AY), home\_team\_red\_cards:toInteger(row.HR), away\_team\_red\_cards:toInteger(row.AR)})

MERGE (r:Referees{referee:row.Referee})

CREATE (h)-[:PLAYED\_HOME]->(m)<-[:PLAYED\_AWAY]-(a)

CREATE (r)-[:REFEREED]->(m)

CREATE (m)-[:PART\_OF]->(s)

## Summary

Text

Description automatically generated

Figure 1 Labels, Nodes and Relationships added

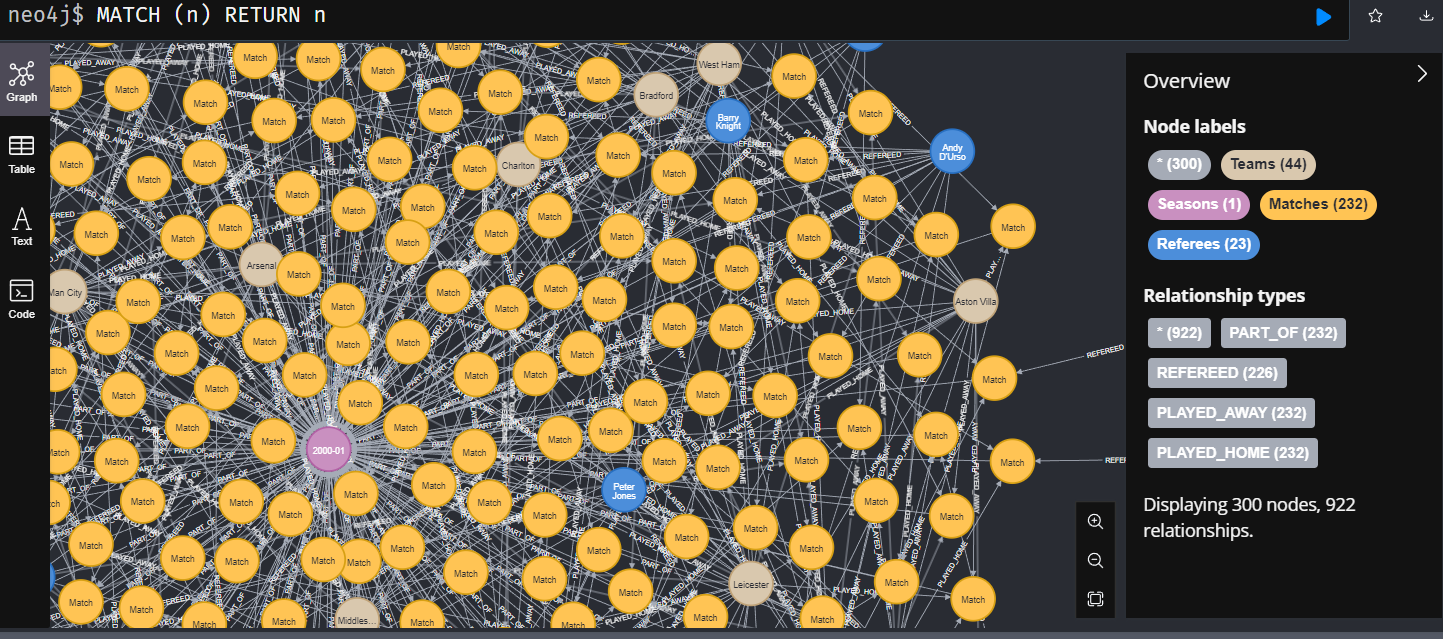


Figure 2 Viewing nodes and relationships

A screenshot of a computer

Description automatically generated with medium confidence

Figure 3 Viewing nodes and relationships

Graphical user interface, application

Description automatically generated

Figure 4 Schema

‘**LOAD CSV WITH HEADERS FROM "file:///EPL\_matches.csv" AS row**

Loads the CSV file with headers, located at the given file path, and assigns each row to the variable 'row'.

**MERGE (h:Teams{name:row.HomeTeam})**

Creates a new node with the label 'Teams' and the property 'name' set to the home team's name.

**MERGE (a:Teams{name:row.AwayTeam})**

Does the same to the away team's name.

**MERGE (s:Seasons{season:row.Season })**

Does the same with label 'Seasons' and the property 'season' set to the season value

**MERGE (m:Matches{...})**

Creates ‘Matches’ node and adds properties such as date/time, goals, results ect.

**MERGE (r:Referees{referee:row.Referee})**

Creates a new node 'Referees' and the property 'referee' set to the referee's name.

**CREATE (h)-[:PLAYED\_HOME]->(m)<-[:PLAYED\_AWAY]-(a)**

Creates two relationships, one between the home team node and the match node with the 'PLAYED\_HOME' relationship type, and another between the away team node and the match node with the ‘PLAYED\_AWAY’ relationship type.

**CREATE (r)-[:REFEREED]->(m)**

This line creates a relationship between the referee node and the match node with the 'REFEREED' relationship type.

**CREATE (m)-[:PART\_OF]->(s)**

Creates a relationship between the match node and the season node with the 'PART\_OF' relationship type, showing which season the match belongs to

# Queries

1. **List all teams that have ever played EPL matches since 2000.**

MATCH (t:Teams)-[:PLAYED\_HOME|PLAYED\_AWAY]->(m:Matches)-[:PART\_OF]->(s:Seasons)

WHERE s.season >= "2000"

RETURN DISTINCT t.name as Team

ORDER BY Team

The query works by teams with their matches and seasons.

**MATCH (t:Teams)-[:PLAYED\_HOME|PLAYED\_AWAY]->(m:Matches)-[:PART\_OF]->(s:Seasons)**

Filters the results to include only seasons from 2000 or later, and returns the distinct names of the teams in alphabetical order

**WHERE s.season >= "2000"**

**RETURN DISTINCT t.name as Team**

**ORDER BY Team**

Graphical user interface, application

Description automatically generated

Figure 5 Query 1

1. **Display all matches “Liverpool” won against “Man United” since 2010.**

MATCH (l:Teams{name:"Liverpool"})-[:PLAYED\_HOME]->(m:Matches)<-[:PLAYED\_AWAY]-(u:Teams{name:"Man United"})

WHERE m.dateTime >= datetime("2010-01-01") AND m.full\_time\_result = "H"

RETURN m.dateTime AS MatchDate, l.name AS HomeTeam, m.full\_time\_home\_team\_goals AS HomeGoals, u.name AS AwayTeam, m.full\_time\_away\_team\_goals AS AwayGoals

This query matches team Liverpool with their matches and filters the results to only include only matches with Man united.

**MATCH (l:Teams{name:"Liverpool"})-[:PLAYED\_HOME]->(m:Matches)<-[:PLAYED\_AWAY]-(u:Teams{name:"Man United"})**

**WHERE m.dateTime >= datetime("2010-01-01") AND m.full\_time\_result = "H"**

It returns information about the match, including the date, the names of the teams, and the number of goals each team scored.

**RETURN m.dateTime AS…**

A screenshot of a computer

Description automatically generated with medium confidence

Figure 6 Query 2

1. **Display top five referees and the number of matches they refereed since 2000.**

MATCH (r:Referees)-[:REFEREED]->(m:Matches)

WHERE m.dateTime >= datetime("2000-01-01")

WITH r, count(\*) AS matches\_count

ORDER BY matches\_count DESC

LIMIT 5

A screenshot of a computer

Description automatically generated with medium confidenceRETURN r.referee AS Referee, matches\_count AS MatchesRefereed

Figure 7 Query 3

Finds referees connected to matches through the relationship from 2000.

**MATCH (r:Referees)-[:REFEREED]->(m:Matches)**

**WHERE m.dateTime >= datetime("2000-01-01")**

It then counts the number of matches each referee participated in.

**WITH r, count(\*) AS matches\_count**

Orders the referees by the number of matches in descending order, limiting the results to the top 5 referees.

**ORDER BY matches\_count DESC**

**LIMIT 5**

Returns the referee's name and the number of matches they officiated.

**RETURN…**

1. **Display all teams and the total numbers of goals they scored and conceded in the 2020/21 season;**

MATCH (t:Teams)-[:PLAYED\_HOME|PLAYED\_AWAY]->(m:Matches)-[:PART\_OF]->(s:Seasons{season:"2020-21"})

WITH t, SUM(CASE WHEN t.name = m.HomeTeam THEN m.full\_time\_home\_team\_goals ELSE m.full\_time\_away\_team\_goals END) AS goals\_scored,

SUM(CASE WHEN t.name = m.HomeTeam THEN m.full\_time\_away\_team\_goals ELSE m.full\_time\_home\_team\_goals END) AS goals\_conceded

RETURN t.name AS Team, goals\_scored AS GoalsScored, goals\_conceded AS GoalsConceded

ORDER BY GoalsScored DESC;

Matches teams to matches in the 2020-21 season as either home or away teamsA screenshot of a computer screen

Description automatically generated with medium confidence.

Figure 8 Query 4

**MATCH (t:Teams)-[:PLAYED\_HOME|PLAYED\_AWAY]->(m:Matches)-[:PART\_OF]->(s:Seasons{season:"2020-21"})**

Calculates the total number of goals scored by each team. Using a conditional statement it finds if the team played at home, it adds the home team's goals to the total, and if the team played away, it adds the away team's goals

**WITH t, SUM(CASE WHEN t.name = m.HomeTeam THEN m.full\_time\_home\_team\_goals ELSE m.full\_time\_away\_team\_goals END) AS goals\_scored,**

**SUM(CASE WHEN t.name = m.HomeTeam THEN m.full\_time\_away\_team\_goals ELSE m.full\_time\_home\_team\_goals END) AS goals\_conceded**

Then returns the team name, total goals scored, and total goals conceded and orders by descending.

**RETURN t.name AS Team, goals\_scored AS GoalsScored, goals\_conceded AS GoalsConceded**

**ORDER BY GoalsScored DESC;**

1. **Which team had the best home winning record since 2000.**

MATCH (t:Teams)-[:PLAYED\_HOME]->(m:Matches)

WHERE m.dateTime >= datetime("2000-01-01") AND m.full\_time\_result = "H"

WITH t, count(\*) AS home\_wins

RETURN t.name AS Team, home\_wins AS `Home Wins`

A screenshot of a computer

Description automatically generated with medium confidenceORDER BY `Home Wins` DESC LIMIT 1

Figure 9 Query 5

Matches all the teams that have played at home and filter to after 2000 with a full time win

**MATCH (t:Teams)-[:PLAYED\_HOME]->(m:Matches)**

**WHERE m.dateTime >= datetime("2000-01-01") AND m.full\_time\_result = "H"**

Group the teams and counts the number of home wins as home\_wins adding ‘ Home Wins’ to column.

**WITH t, count(\*) AS home\_wins**

Returns names and home wins amount, ordering descending and limits to 1 to show the most

**RETURN t.name AS Team, home\_wins AS `Home Wins`**

**ORDER BY 'Home Wins' DESC**

**LIMIT 1**

1. **Which team lost the most matches in 2020/21 season?**

MATCH (t:Teams)-[:PLAYED\_HOME|PLAYED\_AWAY]->(m:Matches)-[:PART\_OF]->(s:Seasons{season:"2020-21"})

WHERE m.full\_time\_result = "A"

WITH t, count(\*) AS losses

RETURN t.name AS `Team Name`, losses AS `Losses`

ORDER BY `Losses` DESC LIMIT 1

Graphical user interface, text, application

Description automatically generated

Figure 10 Query 6

Finds teams linked to matches in the 2020-21 season then filters the matches to include only those where the away team won.

**MATCH (t:Teams)-[:PLAYED\_HOME|PLAYED\_AWAY]->(m:Matches)-[:PART\_OF]->(s:Seasons{season:"2020-21"})**

**WHERE m.full\_time\_result = "A"**

Counts the number of losses for each team and store it as losses.

**WITH t, count(\*) AS losses**

Returned is the team name and the number of losses, ordered by descending and limited to 1.

**RETURN t.name AS `Team Name`, losses AS `Losses`**

**ORDER BY `Losses` DESC LIMIT 1**

1. **Which teams lost the 1st half but won the match in 2020/21 season.**

MATCH (h:Teams)-[:PLAYED\_HOME|PLAYED\_AWAY]->(m:Matches)-[:PART\_OF]->(s:Seasons{season:"2020-21"})

WHERE m.half\_time\_result = "A" AND m.full\_time\_result = "H"

RETURN h.name AS `Team Name`, m.dateTime AS `Match Date`

A screenshot of a computer

Description automatically generatedORDER BY `Match Date`

Figure 11 Query 7

Finds all relationships between the Teams and Matches, the match is part of the 2020-21 season.

**MATCH (h:Teams)-[:PLAYED\_HOME|PLAYED\_AWAY]->(m:Matches)-[:PART\_OF]->(s:Seasons{season:"2020-21"})**

Where the team was losing’A’ at half time but them won ‘H’ at full time

**WHERE m.half\_time\_result = "A" AND m.full\_time\_result = "H"**

Returns the name, match date and orders by date

**RETURN h.name AS `Team Name`, m.dateTime AS `Match Date**`

1. **Which team earned the highest ever points in all seasons since 2000**

MATCH (t:Teams)-[:PLAYED\_HOME|PLAYED\_AWAY]->(m:Matches)-[PLAYED\_IN]->(s:Seasons)

WHERE toInteger(split(s.season, '-')[0]) >= 2000

WITH t,

sum(CASE WHEN t.name = m.HomeTeam THEN

CASE WHEN m.full\_time\_result = 'H' THEN 3

WHEN m.full\_time\_result = 'D' THEN 1

ELSE 0 END

ELSE

CASE WHEN m.full\_time\_result = 'A' THEN 3

WHEN m.full\_time\_result = 'D' THEN 1

ELSE 0 END

END) AS points

RETURN t.name AS `Team Name`, sum(points) AS `Total Points`

ORDER BY `Total Points` DESC

Graphical user interface, text

Description automatically generatedLIMIT 1

Figure 12 Query 8

Finds teams that played either as a home team or away team in matches of seasons from the year 2000 and onwards

**MATCH (t:Teams)-[:PLAYED\_HOME|PLAYED\_AWAY]->(m:Matches)-[PLAYED\_IN]->(s:Seasons)**

**WHERE toInteger(split(s.season, '-')[0]) >= 2000**

Calculates the total number of points each team earned in those matches based on the match results: 3 points for a win, 1 point for a draw, and 0 points for a loss.

**sum(CASE WHEN t.name = m.HomeTeam THEN**

**CASE WHEN m.full\_time\_result = 'H' THEN 3**

**WHEN m.full\_time\_result = 'D' THEN 1**

**ELSE 0 END**

**ELSE**

**CASE WHEN m.full\_time\_result = 'A' THEN 3**

**WHEN m.full\_time\_result = 'D' THEN 1**

**ELSE 0 END**

**END) AS points**

Returns the team name and its total points, sorted in descending order by points, and the team with the most points is returned

**RETURN…**

1. **Display the final league table ranking of all teams in the 2020/21 season (based on the total points).**

MATCH (t:Teams)-[:PLAYED\_HOME|PLAYED\_AWAY]->(m:Matches)-[:PART\_OF]->(s:Seasons{season:"2020-21"})

WITH t, SUM(CASE WHEN t.name = m.HomeTeam THEN

CASE WHEN m.full\_time\_result = 'H' THEN 3

WHEN m.full\_time\_result = 'D' THEN 1

ELSE 0 END

ELSE

CASE WHEN m.full\_time\_result = 'A' THEN 3

WHEN m.full\_time\_result = 'D' THEN 1

ELSE 0 END

END) AS points

RETURN t.name AS `Team Name`, sum(points) AS `Total Points`

A screenshot of a computer

Description automatically generated with medium confidenceORDER BY `Total Points` DESC

Figure 13 Query 9

Finds all the teams that played home or away in the 2020-21 season.

The SUM function is used to total the points earned in each match, and the CASE statement is used to calculate the points based on the full-time result of the match.

Returns team name and sorts the results by total points in descending order.

1. **Which team had the longest unbeaten record;**

MATCH (t:Teams)-[r:PLAYED\_HOME|PLAYED\_AWAY]->(m:Matches)-[p:PART\_OF]->(s:Seasons)

WHERE m.dateTime >= datetime("2000-01-01") AND m.full\_time\_result IN ['H', 'D']

WITH t, COLLECT(DISTINCT m.dateTime) AS matchDates

WITH t, REDUCE(acc = {prev: null, max: 0}, d IN matchDates |

CASE WHEN acc.prev IS NULL OR d = acc.prev + duration({days: 1}) THEN

{prev: d, max: acc.max + 1}

ELSE

{prev: d, max: 1}

END

) AS longestUnbeatenRecord

RETURN t.name AS `Team Name`

ORDER BY longestUnbeatenRecord.max DESC LIMIT 1

Graphical user interface, text

Description automatically generated

Figure 14 Query 10

Matches teams that played in matches in a season.

filters the matches based on the date and result and collects the distinct match dates for each team.

Uses reduce to iterate through the match dates and calculate the length of the longest unbeaten record.

Returns the name of team.

# Conclusion

Overall, the aims and objectives have been met. The dataset was analysed, and the graph model was made with what is believed to be the current information and relationships. Using Cypher query language all ten queries were answers and they are believed to be correct. The graph database could efficiently complete the queries.

The assignment was challenging and time consuming but overall, it was interesting, the nodes and relationships are more user friendly however took some time to get used to. However it was efficient and mostly easy to query the information needed.

# References

Neo4j (2021). Introduction to Neo4j - a hands-on crash course. YouTube. Available at: https://www.youtube.com/watch?v=ou2st6FYxR8 [Accessed 23 March. 2023].

Neo4j Graph Data Platform. (n.d.). Neo4j Learning Resources - Developer Guides. [online] Available at: https://neo4j.com/developer/resources/ [Accessed 26 Mar. 2023].

Neo4j Graph Database Platform. (n.d.). Cypher Query Language. [online] Available at: <https://neo4j.com/developer/cypher/>

Memgraph.com. (2021). The Complete Cypher Cheat Sheet. [online] Available at: https://memgraph.com/blog/cypher-cheat-sheet.