KATHMANDU UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Dhulikhel, Kavre

A Report on

NOSQL

[Code No: COMP 232]

Submitted by:

Aakriti Banjara(05)

Submitted to:

Mr. Santosh Khanal Sir

Department of Computer Science and Engineering

Submission Date: 27th May, 2023

Introduction

NoSQL (not only SQL or non relational) is a database system which provides a mechanism for storage and retrieval of data that is modeled in non tabular relations. NoSQL databases emerged as a response to the challenges posed by big data, real-time data processing, and scalability requirements of modern applications . These models are different from those in relational databases, making some operations faster in NoSQL. A common misconception is that NoSql doesn't store relationship data, but NoSQL can store relationship data; they just store it in a different way than relational databases do. However, they may sacrifice some features provided by traditional SQL databases, such as strong consistency guarantees and complex transaction support.

Some of the basic data models of NoSQL are:

- Key-value stores: These databases store data as a collection of key-value pairs. The keys are unique identifiers, and the values can be any type of data. Examples of key-value stores include Redis, Amazon DynamoDB, and Riak.

- Document databases: They store and retrieve data in the form of documents, usually using JSON or XML. Documents can be organized into collections, and each document can have a different structure. MongoDB and CouchDB are popular examples of document databases.

- Columnar databases: These databases store data in columns rather than rows, allowing for efficient data retrieval and analysis. They are suitable for analytics and big data workloads. Apache Cassandra and Apache HBase are widely used columnar databases.

- Graph databases: They are designed to represent and process data in the form of interconnected graphs, consisting of nodes (entities) and edges (relationships). Graph databases excel at managing highly connected data and performing complex graph-based queries. Examples include Neo4j and Amazon Neptune.

Importance of NoSQL database system:

- Flexibility: Provides flexible schemas for faster development.
- Scalability: Designed to scale out by using distributed clusters of hardware.
- High Performance: Optimized for specific data models and access patterns that enable higher performance.
- Big Data and Real-time processing : NoSQL databases excel in handling big data applications, where large volumes of data need to be processed and analyzed in real-time
- Agile Development: NoSQL databases embrace agile development practices by allowing developers to iterate rapidly and make schema changes on the fly.
- Handling Unstructured and Semi-Structured Data: NoSQL databases are well-suited for managing unstructured and semi-structured data, such as documents, JSON, XML, or graph structures.
- Cost-Effectiveness: NoSQL databases often leverage commodity hardware and distributed architectures, making them cost-effective alternatives to traditional relational databases.

Neo4j

Neo4j is a graph database management system, described as an ACID compliant transactional database with native graph storage and processing. It is widely used in diverse domains, including social networking, recommendation systems, fraud detection, network analysis, bioinformatics, and knowledge management. It is implemented in Java and accessible from software written in other languages using Cypher query language. In Neo4j, everything is stored in the form of an edge, node or attribute. Each node and edge can have any number of attributes. Both nodes and edges can be labeled.

Installation

Download the installation file from https://neo4j.com/download/. Copy the activation key provided from the website. Run the installation file and follow its steps. After installation paste the activation key. Now, we are ready to run the Neo4j desktop.

Lab Works

We were provided with the epl_Dataset.csv file for our lab work assignment. The file included information on the EPL of 2018/2019 season such as wins, loss, goals of the participating teams. We were to graph databases to visualize the dataset and also answer certain queries.

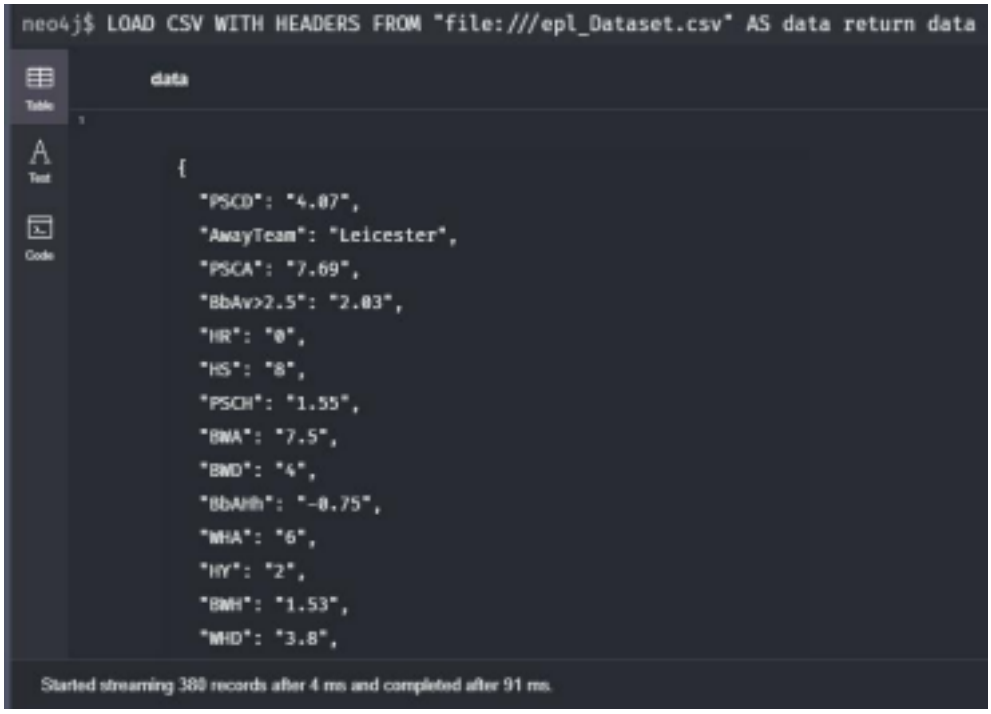
Importing data

To import the data, first download the provided epl_Dataset.csv. Then, in the Neo4j application, create a database with a name such as 'EPL Database'. Now, store the file in the location:

C:\Users\Admin\.Neo4jDesktop\relate-data\dbmss\dbms-fc2845f6-6b0e-4fb1-9ed4-225ef0a896f4\import

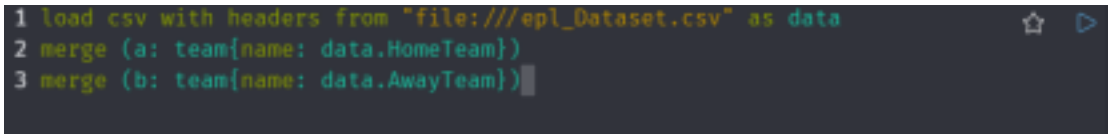
Now start the database and use the following command to check if import was correct.

```
load csv with headers from "file:///epl_Dataset.csv" as
data return data
```



Creating Nodes

Nodes for team:



Nodes for all games:

```
load csv with headers from "file:///epl_Dataset.csv" AS data
MERGE (m:Games{
div: data.Div,
date:data.Date,
HomeTeam: data.HomeTeam,
AwayTeam: data.AwayTeam,
FTHG: data.FTHG,
FTAG: data.FTAG,
HTHG: data.HTHG,
HTAG: data.HTAG
})
```

In the creation of nodes called Games, where the game information has been stored. The node is created with data from csv file which uses acronym whose meaning are as follows:

Div = League Division
Date = Match Date (dd/mm/yy)
FTHG = Full Time Home Team Goals
FTAG = Full Time Away Team Goals
FTR = Full Time Result (H= Win for Home, D=Draw, A=Win for Away)
HTHG = Half Time Home Team Goals
HTAG = Half Time Away Team Goals

HTR = Half Time Result (H=Home Win, D=Draw, A=Away Win)

Nodes for Home team wins:

```
match (n:Games) where n.FTHG > n.FTAG
match (a:team) where a.name = n.HomeTeam
match (b:team) where b.name = n.AwayTeam
merge (a) - [r1:won{FTgoals: n.FTHG, HTgoals : n.HTHG}]→(n)←[r2:lost{FTgoals:n.FTAG,HTgoals:n.HTAG}] - (b)
```

Nodes for Away team wins:

```
match (n:Games) where n.FTHG < n.FTAG
match (a:team) where a.name = n.HomeTeam
match (b:team) where b.name = n.AwayTeam
merge (a) - [r1:lost{FTgoals: n.FTHG, HTgoals : n.HTHG}]→(n)←[r2:won{FTgoals:n.FTAG,HTgoals:n.HTAG}] - (b)
```

Nodes for Draw games:

```
match (n:Games) where n.FTHG = n.FTAG
match (a:team) where a.name = n.HomeTeam
match (b:team) where b.name = n.AwayTeam
merge (a) - [r1:draw{FTgoals: n.FTHG, HTgoals : n.HTHG}]→(n)←[r2:draw{FTgoals:n.FTAG,HTgoals:n.HTAG}] - (b)
```

Query and its outputs(screenshots)

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

   Query: `match (a:team) return distinct (a)`

   Output:

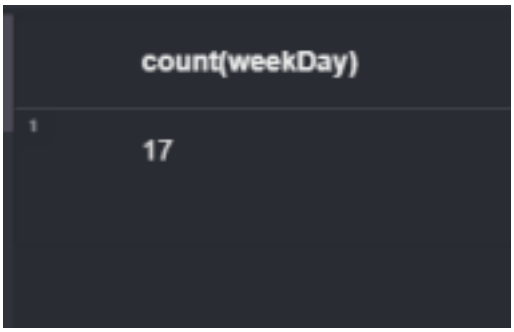
   In the section of creating nodes, we had created the node called team with merge command, and here we use the match command to display all the nodes called team.

2. **How many matches were played on Mondays?**

   Query:

```
load csv with headers from "file:///epl_Dataset.csv" as data
with [item in split(data['Date'], "/") | toInteger(item)] AS component
with date({day: component[0], month: component[1], year:
 component[2]}).dayOfWeek as weekDay
where weekDay = 1
return count(weekDay)
```

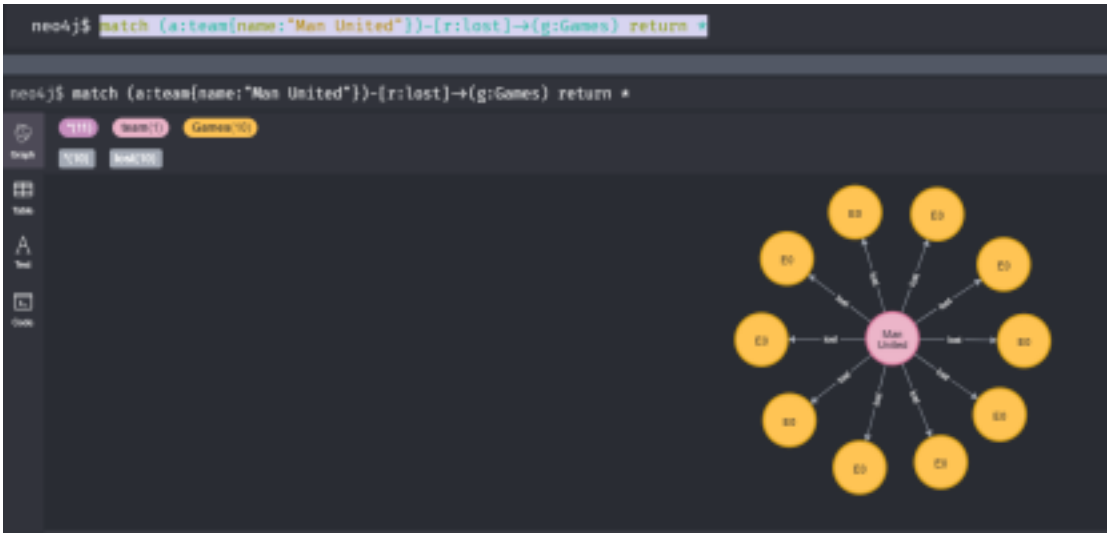
   Output:



The Neo4j date type enables us to figure out the day of week using date. We converted the data from the date column of our csv file and compared all the dates with weekday 1 as Monday has an index of 1.

3. Display all the matches that "Man United" lost.

Query and output:



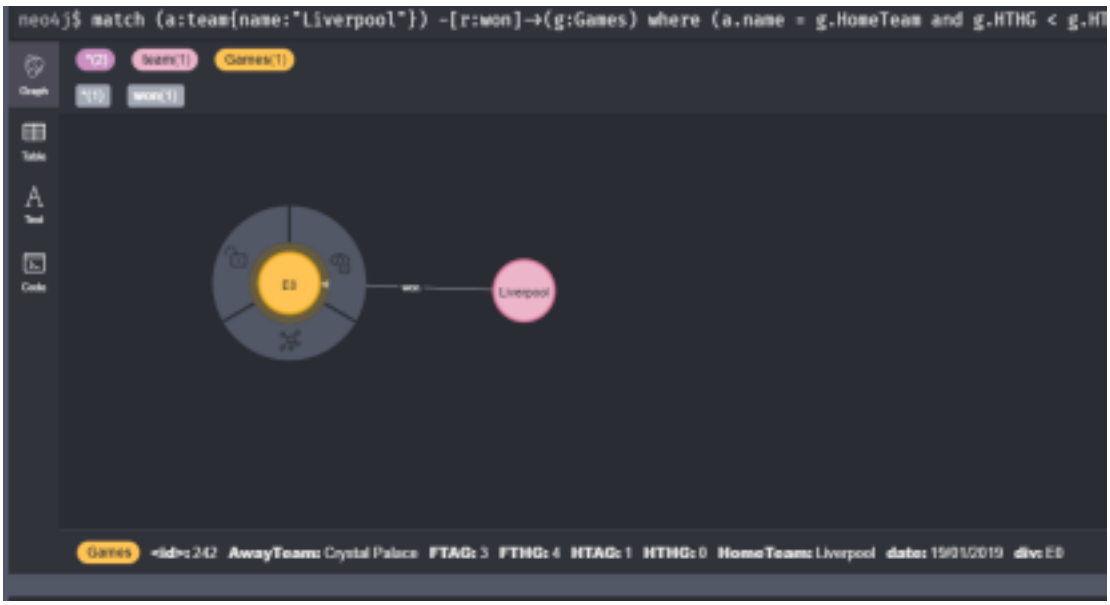
We have created the nodes for the games that a team, as both HomeTeam and as AwayTeam, has lost. So now, we use the Match clause to find the games that the team with the name "Man United" has lost and return it. The E0 are the nodes for games that we have created. On clicking the E0 node, it displays the match information such as, the winning teams, goals, etc.

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

Query:

```
1 match (a:team{name:"Liverpool"}) -[r:won]→(g:Games)
2 where (a.name = g.HomeTeam and g.HTHG < g.HTAG) or (a.name = g.AwayTeam and g.HTAG < g.HTHG)  return *
```

Output:



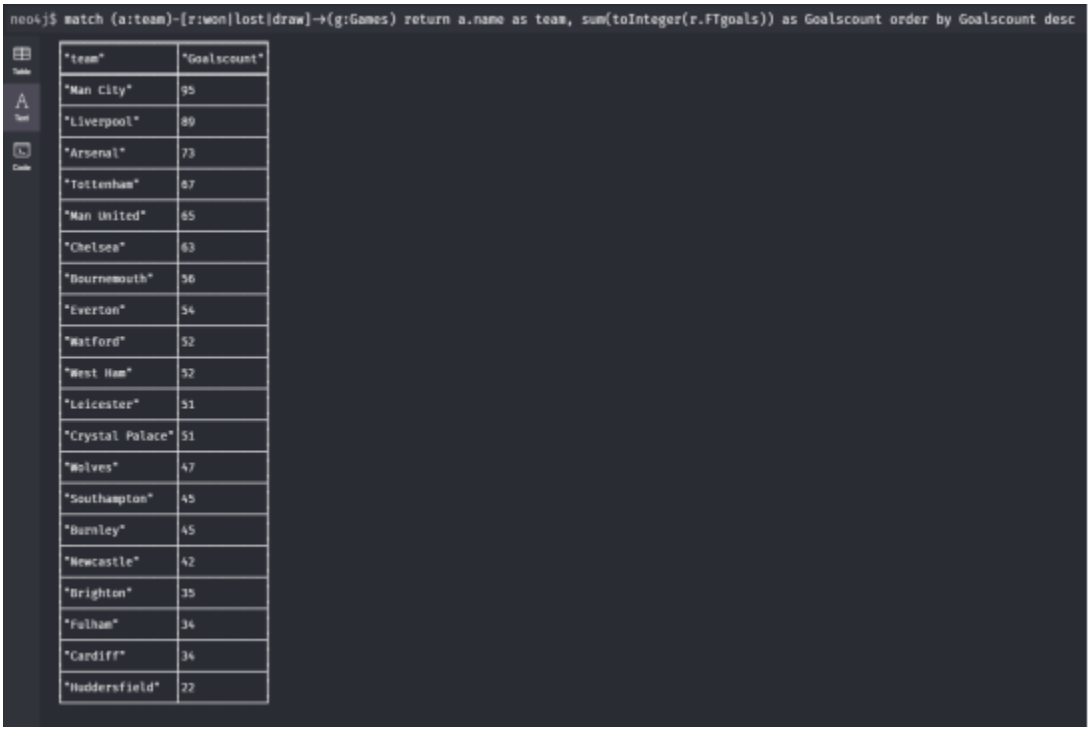
Here, we display the game where the team 'Liverpool' won the game in which they were down in the first half. We accomplish this, with the condition g.HTHG < g.HTAG for 'Liverpool's score' or g.HTAG < g.HTHG for away team score. Basically, we are comparing the half time goals, and displaying the games in which Liverpool won despite the goal before half time being lower than the opponent.

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

   Query:

```
match (a:team)-[r:won|lost|draw]→(g:Games) return a.name as team,
sum(toInteger(r.FTgoals)) as Goalscount order by Goalscount desc
```

Output:



```
neo4j$ match (a:team)-[r:won|lost|draw]→(g:Games) return a.name as team, sum(toInteger(r.FTgoals)) as Goalscount order by Goalscount desc
```

"team"	"Goalscount"
"Man City"	95
"Liverpool"	89
"Arsenal"	73
"Tottenham"	67
"Man United"	65
"Chelsea"	63
"Bournemouth"	56
"Everton"	54
"Watford"	52
"West Ham"	52
"Leicester"	51
"Crystal Palace"	51
"Wolves"	47
"Southampton"	45
"Burnley"	45
"Newcastle"	42
"Brighton"	35
"Fulham"	34
"Cardiff"	34
"Huddersfield"	22

To determine the final rankings, we used the total number of goals by a team as their total points. And then we displayed the data in descending order to display the final ranking.

Conclusion

By the end of all our lab sessions, we became confident in the use of the Neo4j desktop application. We became able to manage a graph database. We learned to create node relationships and answer basic queries in cypher query language. It was a great learning experience which will be helpful in our future career.