

## **Introduction**

The 2014 FIFA World Cup, hosted by Brazil was an international association football competition within the senior men's division of national teams. The FIFA World Cup tournament has been hosted every four years and consisted of 32 teams in group stages with the top performing teams advancing to the knockout stages until there is one country named the winner.

The 2014 FIFA World Cup dataset, containing information on players' performances, positions, clubs, and countries, holds immense value. The dataset offers a comprehensive overview of the tournament's dynamics for sports analysts seeking to uncover trends and patterns for coaches aiming to refine team strategies. Media professionals can utilise the data to expand their coverage, providing fans/followers with deeper insights and context behind the on-field action. Even passionate fans can delve into the dataset to relive memorable moments, track their favourite players' performances, and gain a deeper understanding of the tournament's narrative. Leveraging a graph database for analysing this dataset offers numerous advantages, primarily due to its capability to efficiently manage and analyse interconnected data.

By modelling player relationships, team dynamics, and performance metrics within a graph database, stakeholders can gain deeper insights into player interactions, team strategies, and overall tournament dynamics. This approach facilitates more intuitive data modelling, efficient querying of relationships, and powerful analytical capabilities, ultimately enhancing the understanding of player and team dynamics during the 2014 FIFA World Cup.

# Graph database

We use property graphs as they provide an expansive and efficient way to represent relationships between our entities.

## Players Node

### Properties of Players:

- **player\_id**: Unique identifier for each player.
- **Player**: Name of the player.
- **Position**: Playing position of the player (e.g., Forward, Midfielder).
- **Number**: Jersey number of the player.
- **D.O.B**: Date of birth of the player.
- **Age**: Age of the player.
- **Height(cm)**: Height of the player in centimetres.
- **Caps**: Number of appearances for the player's national team.
- **Goals**: Number of goals scored by the player.
- **Home**: Boolean indicating if the player is playing for their home country's club.

## Clubs Node

### Properties of Clubs:

- **club\_id**: Unique identifier for each club.
- **club**: Name of the club.
- **club(country)**: The country where the club is located.

## Country Node

### Properties of Country:

- **country\_id**: Unique identifier for each country.
- **country**: Name of the country.

## BELONGS\_TO Relationship

- **From**: PLAYERS
- **To**: CLUBS
- **Purpose**: Indicates the affiliation between players and their respective clubs.

## LOCATED\_IN Relationship

- **From**: CLUBS
- **To**: COUNTRY
- **Purpose**: Helps identify the geographical location of each club within a specific country.

## HAS\_CLUB Relationship

- **From:** COUNTRY
- **To:** CLUBS
- **Purpose:** Provides a reverse mapping from countries to the clubs within them.

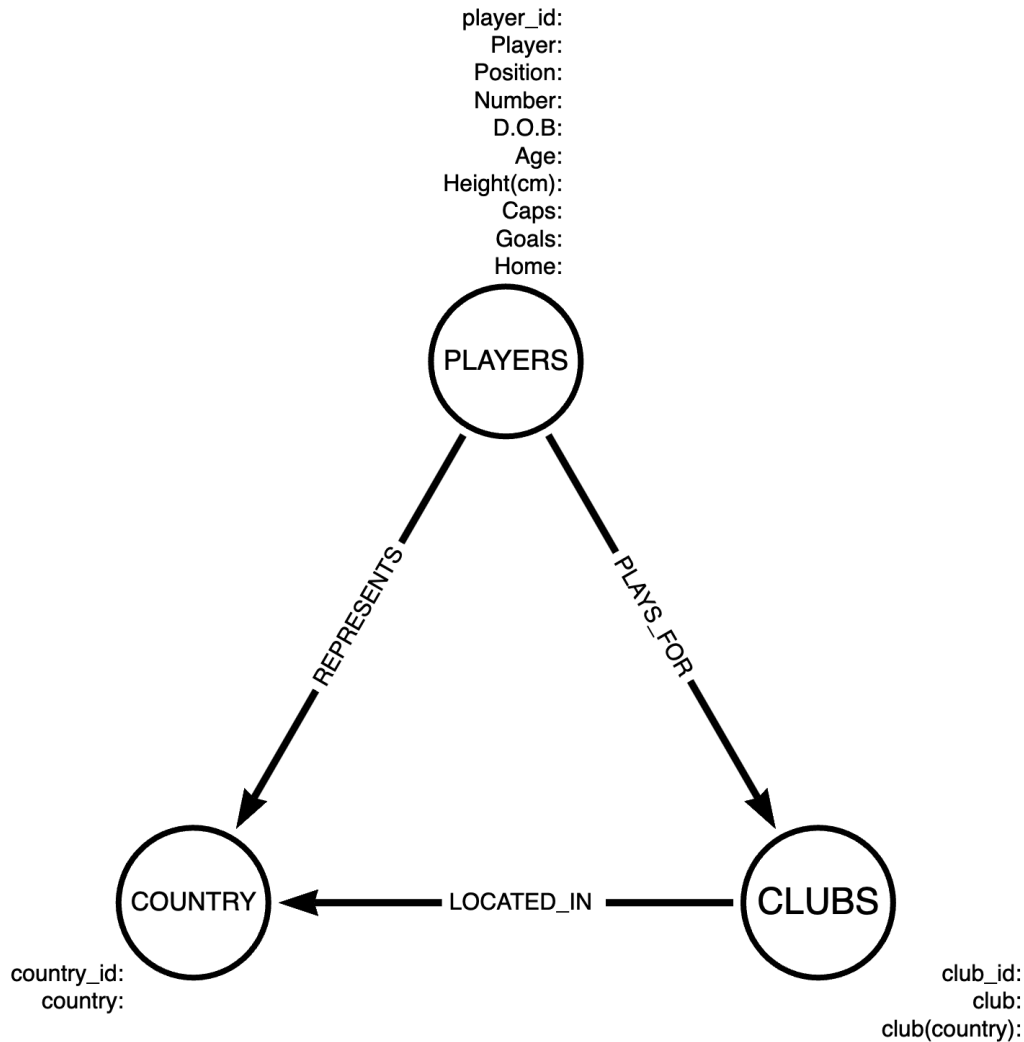
## Rationale

**Modularity:** The separation of entities into distinct nodes ensures a clear and modular data representation. By organising the data into separate nodes for players, clubs, and countries, we create a more organised and manageable structure, making it easier to understand and maintain the dataset.

**Explicit Relationships:** Defining relationships explicitly between nodes captures the associations between players, clubs, and countries. This aids in querying and understanding the data structure, as it provides clarity on how entities are connected. For example, the BELONGS\_TO relationship indicates which player belongs to which club.

**Data Integrity:** Unique identifiers, such as player\_id, club\_id, and country\_id, prevent data redundancy and maintain data integrity. These identifiers ensure that each entity is uniquely identifiable within the graph, enabling accurate retrieval and manipulation of data without ambiguity.

**Scalability:** The design is scalable and flexible, accommodating additional attributes or new relationship types as data requirements evolve. This scalability ensures that the graph can grow and adapt to future changes in the dataset, allowing for the incorporation of new data without requiring significant restructuring of the data model.



## ETL Process

Extract, Transform, and Load (ETL) processes for graph databases are designed to handle relationships and connections between entities.

The extraction process involves gathering data from the 'FIFA2014 - all players'.csv file, which contains various attributes related to each player participating in the 2014 FIFA World Cup. This dataset includes information such as player name, position, club, club country, country represented, date of birth, height, age, caps, international goals, and a boolean indicating whether the player plays in their home country.

In Python, the pandas library is used to read the CSV file into a DataFrame. Then, unique countries and clubs are extracted from the dataset, and unique identifiers (country\_id and club\_id) are assigned to each country and club, respectively.

The transformation process involves preparing the data for the graph database schema by normalising and restructuring it. This step includes creating separate CSV files for each entity (players, countries, clubs) and relationships (player-club, club-country, player-country).

For players, the relevant columns are selected and renamed appropriately. Similarly, for clubs, unique clubs and their respective countries are identified, and club\_id is assigned. Finally, country\_id is mapped to each player and club based on the country they represent or are located in.

```
import pandas as PD

# Load the FIFA2014 - all players.csv file

fifa_file_path = 'unfiltered/FIFA2014 - all players.csv'

fifa_data = pd.read_csv(fifa_file_path)

# Extract the unique countries and assign unique country_id starting from 101

unique_countries = fifa_data['Country'].unique()

country_dict = {country: idx + 101 for idx, country in enumerate(unique_countries)}

# Create country.csv

country_df = pd.DataFrame(list(country_dict.items()), columns=['country',
'country_id'])

country_output_path = 'filtered/country.csv'

country_df.to_csv(country_output_path, index=False)

# Extract the unique clubs and assign unique club_id starting from 101

unique_clubs = fifa_data[['Club', 'Club (country)']].drop_duplicates()

unique_clubs['club_id'] = range(101, 101 + len(unique_clubs))

# Create club.csv

club_df = unique_clubs.rename(columns={'Club': 'club', 'Club (country)':
'club_country'})
```

```

club_output_path = 'filtered/club.csv'

club_df.to_csv(club_output_path, index=False)

# Create players.csv with relevant columns

players_df = fifa_data[['Player id', 'Player', 'Position', 'Number', 'D.O.B', 'Age',
'Height (cm)', 'Caps', 'International goals', 'Plays in home country?']]

players_df.columns = ['player_id', 'Player', 'Position', 'Number', 'D.O.B', 'Age',
'Height (cm)', 'Caps', 'Goals', 'Home']

players_output_path = 'filtered/players.csv'

players_df.to_csv(players_output_path, index=False)

# Create playerscountry.csv by combining players_df and country_df

players_df['country'] = fifa_data['Country']

players_df['country_id'] = players_df['country'].map(country_dict)

playerscountry_df = players_df[['player_id', 'Player', 'Position', 'Number', 'D.O.B',
'Age', 'Height (cm)', 'Caps', 'Goals', 'Home', 'country', 'country_id']]

playerscountry_output_path = 'filtered/playerscountry.csv'

playerscountry_df.to_csv(playerscountry_output_path, index=False)

# Create clubplayer.csv by combining players_df and club_df

players_df['club'] = fifa_data['Club']

players_df['club_country'] = fifa_data['Club (country)']

players_df = pd.merge(players_df, club_df, left_on=['club', 'club_country'],
right_on=['club', 'club_country'], how='left')

clubplayer_df = players_df[['player_id', 'Player', 'Position', 'Number', 'D.O.B',
'Age', 'Height (cm)', 'Caps', 'Goals', 'Home', 'club', 'club_country', 'club_id']]

clubplayer_output_path = 'filtered/clubplayer.csv'

```

```

clubplayer_df.to_csv(clubplayer_output_path, index=False)

# Create clubcountry.csv by linking clubs to their respective countries

# Ensure clubs are matched with their correct country from the original data

club_country_map = fifa_data[['Club', 'Club (country)']].drop_duplicates()

club_country_map = club_country_map.rename(columns={'Club': 'club', 'Club (country)':
'club_country'})

# Merge to get club_id

club_country_map = pd.merge(club_country_map, club_df[['club', 'club_country',
'club_id']], on=['club', 'club_country'], how='left')

# Add the country based on club_country

club_country_map['country'] = club_country_map['club_country']

# Ensure all unique countries from the club_country are in the country_dict

club_country_unique = club_country_map['country'].unique()

for country in club_country_unique:

    if country not in country_dict:

        country_dict[country] = len(country_dict) + 101

# Update country_df with any new countries added

country_df = pd.DataFrame(list(country_dict.items()), columns=['country',
'country_id'])

country_df.to_csv(country_output_path, index=False)

# Map country_id

club_country_map['country_id'] = club_country_map['country'].map(country_dict)

# Final clubcountry DataFrame

```

```
clubcountry_df = club_country_map[['club', 'club_country', 'club_id', 'country',  
'country_id']]  
  
clubcountry_output_path = 'filtered/clubcountry.csv'  
  
clubcountry_df.to_csv(clubcountry_output_path, index=False)
```

Documenting the code (code located in 'Python,ipynb' run in Python environment), the ETL Process follows:

**Importing Libraries:** The code starts by importing the pandas library, which is commonly used for data manipulation and analysis in Python.

**Loading the Dataset:** The code reads the FIFA2014 - all players.csv file from the 'unfiltered' folder into a pandas DataFrame named fifa\_data. This file contains information about players participating in the 2014 FIFA World Cup, including attributes such as player name, club, country, position, etc.

**Extracting Unique Countries:** The code extracts unique country names from the 'Country' column of the dataset and creates a dictionary (country\_dict) to assign a unique country\_id to each country. The enumerate() function is used to generate unique identifiers starting from 101.

**Creating country.csv:** The unique country names and their corresponding country\_id values are stored in a pandas DataFrame named country\_df, which is then saved to a CSV file named filtered/country.csv'.

**Extracting Unique Clubs:** Similar to extracting countries, the code extracts unique club names and their corresponding countries from the dataset. The unique club names are assigned unique club\_id values starting from 101.

**Creating club.csv:** The unique club names, their corresponding countries, and club\_id values are stored in a pandas DataFrame named club\_df, which is then saved to a CSV file named filtered/club.csv'.

**Creating players.csv:** The code selects relevant columns from the dataset to create a DataFrame (players\_df) containing player information such as player\_id, name, position, etc. The column names are renamed for consistency, and the DataFrame is saved to a CSV file named filtered/players.csv'.



Creating playerscountry.csv: The code combines player information with country information to create a DataFrame (playerscountry\_df) containing player information along with their respective countries and country\_ids. This DataFrame is then saved to a CSV file named filtered/playerscountry.csv'.

Creating clubplayer.csv: The code combines player information with club information to create a DataFrame (clubplayer\_df) containing player information along with the clubs they belong to and their respective club\_ids. This DataFrame is then saved to a CSV file named filtered/clubplayer.csv'.

Creating clubcountry.csv: The code creates a DataFrame (club\_country\_map) linking clubs to their respective countries. It ensures that all unique countries are mapped to their respective country\_id values. The final DataFrame (clubcountry\_df) containing club-country mappings is saved to a CSV file named 'data/clubcountry.csv'.

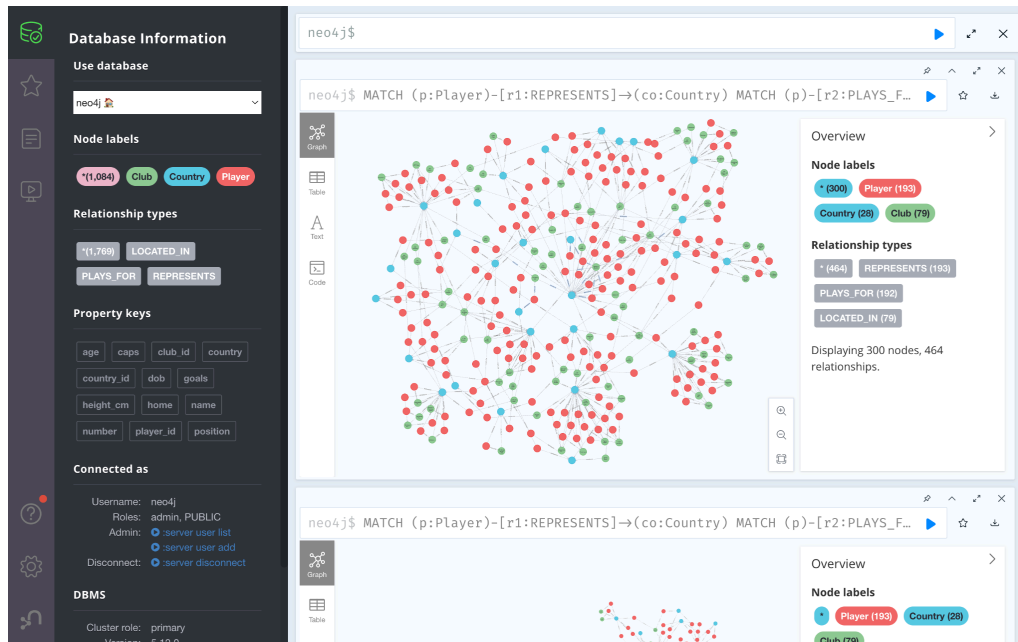
Using Neo4j, all created CSV files are copied into the imports folder for the database. Once the active DBMS is opened, each entity (players, countries, clubs) is represented as a node, and relationships between entities are represented as edges.

For each node type (Player, Club, Country), a Cypher query is executed to create nodes with properties extracted from the CSV files. Similarly, for each relationship type (PLAYS\_FOR, LOCATED\_IN, REPRESENTS), a Cypher query is executed to create relationships between nodes based on the corresponding CSV files.

Player Node	<pre>LOAD CSV WITH HEADERS FROM 'file:///players.csv' AS row CREATE (p: Player {     player_id: row.player_id,     name: row.Player,     position: row.Position,     number: row.Number,     dob: row.`D.O.B`,     age: toInteger(row.Age),     height_cm: toInteger(row.`Height (cm)`),     caps: toInteger(row.Caps),     goals: toInteger(row.Goals),     home: row.Home });</pre>
-------------	---

Club Node	LOAD CSV WITH HEADERS FROM 'file:///club.csv' AS row CREATE (c: Club { club_id: toInteger(row.club_id), name: row.club, country: row.club_country });
Country Node	LOAD CSV WITH HEADERS FROM 'file:///country.csv' AS row CREATE (co: Country { country_id: toInteger(row.country_id), name: row.country });
Player-Club (PLAYS_FOR)	LOAD CSV WITH HEADERS FROM 'file:///clubplayer.csv' AS row MATCH (p:Player {player_id: row.player_id}) MATCH (c:Club {club_id: toInteger(row.club_id)}) CREATE (p)-[:PLAYS_FOR]->(c);
Club-Country(LOCATED_IN)	LOAD CSV WITH HEADERS FROM 'file:///clubcountry.csv' AS row MATCH (c:Club {club_id: toInteger(row.club_id)}) MATCH (co:Country {country_id: toInteger(row.country_id)}) CREATE (c)-[:LOCATED_IN]->(co);
Player-Country(REPRESENTS)	LOAD CSV WITH HEADERS FROM 'file:///playerscountry.csv' AS row MATCH (p:Player {player_id: row.player_id}) MATCH (co:Country {country_id: toInteger(row.country_id)}) CREATE (p)-[:REPRESENTS]->(co);

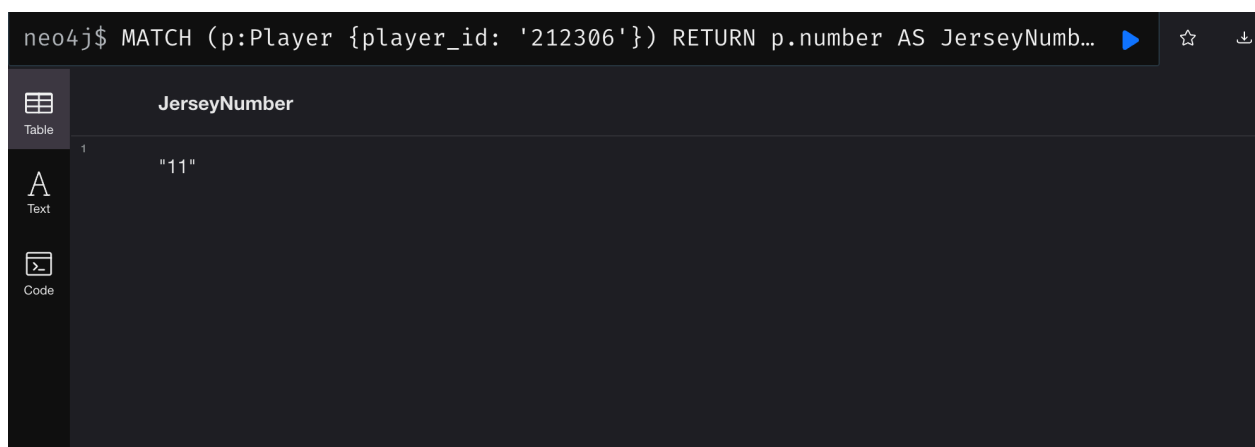
After successfully loading the entities, the verification step ensures the ETL process is successful. We do this simply by visualising entities:



# Answering Cypher Queries

1. What is the jersey number of the player with player\_id <a specific player id>?

`MATCH (p:Player {player_id: '<a specific player id>'}) RETURN p.number AS JerseyNumber;`



## 2. Which clubs are based in <a specific country>?

MATCH (c:Club)-[:LOCATED\_IN]->(co:Country {name: '<a specific country>'}) RETURN c.name AS ClubName;



The image shows the Neo4j query interface. The query entered is: `neo4j$ MATCH (c:Club)-[:LOCATED_IN]->(co:Country {name: 'Australia'}) RETURN c.name AS ClubName;`. The results are displayed in a table view with the following data:

	ClubName
1	"Newcastle United Jets FC"
2	"Western Sydney Wanderers FC"
3	"Brisbane Roar FC"
4	"Adelaide United FC"
5	"Melbourne Victory FC"

## 3. Which club does <a specific player> play for?

MATCH (p:Player {name: '<a specific player>'})-[:PLAYS\_FOR]->(c:Club) RETURN c.name AS ClubName;

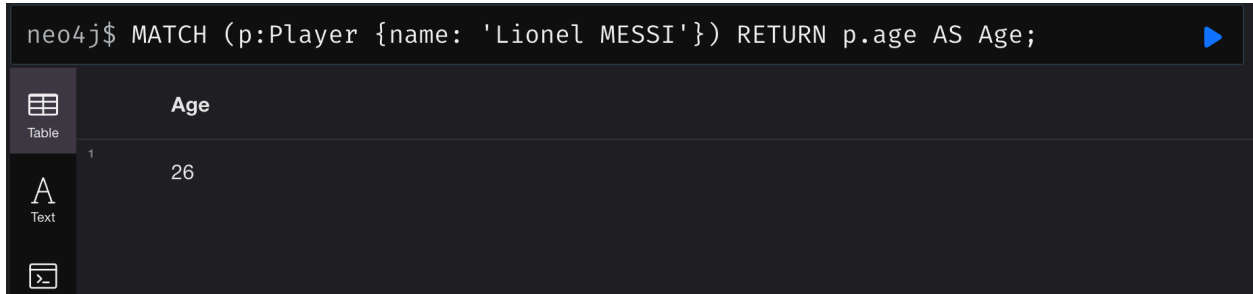


The image shows the Neo4j query interface. The query entered is: `1 MATCH (p:Player {name: 'Lionel MESSI'})-[:PLAYS_FOR]->(c:Club) RETURN c.name AS ClubName;`. The results are displayed in a table view with the following data:

	ClubName
1	"FC Barcelona"

#### 4. How old is <a specific player>?

```
MATCH (p:Player {name: '<a specific player>'}) RETURN p.age AS Age;
```

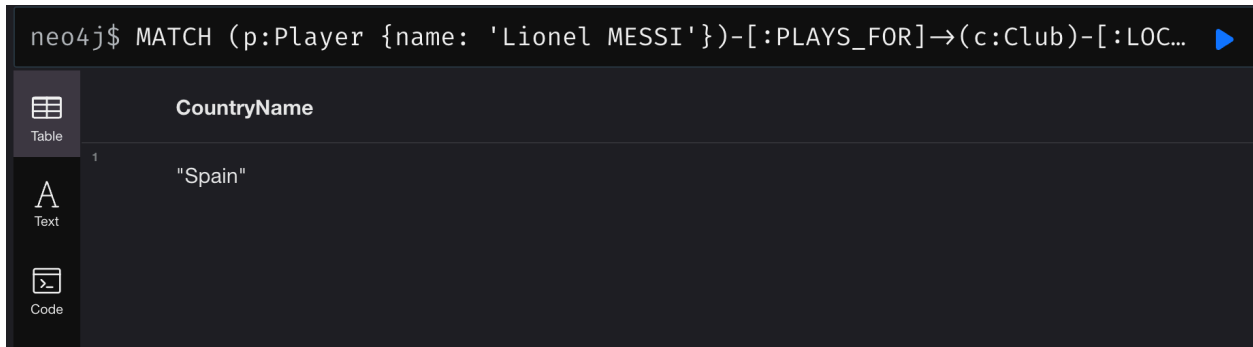


The image shows a Neo4j query interface. At the top, a query is entered: `neo4j$ MATCH (p:Player {name: 'Lionel MESSI'}) RETURN p.age AS Age;`. Below the query, there is a table with one column labeled 'Age'. The table has one row with the value '26'. On the left side of the table, there are three icons: a table icon labeled 'Table', a text icon labeled 'Text', and a code icon labeled 'Code'.

Age
26

#### 5. In which country is the club that <a specific player> plays for?

```
MATCH (p:Player {name: '<a specific player>'})-[:PLAYS_FOR]->(c:Club)-[:LOCATED_IN]->(co:Country)
RETURN co.name AS CountryName;
```



The image shows a Neo4j query interface. At the top, a query is entered: `neo4j$ MATCH (p:Player {name: 'Lionel MESSI'})-[:PLAYS_FOR]->(c:Club)-[:LOCATED_IN]->(co:Country) RETURN co.name AS CountryName;`. Below the query, there is a table with one column labeled 'CountryName'. The table has one row with the value '"Spain"'. On the left side of the table, there are three icons: a table icon labeled 'Table', a text icon labeled 'Text', and a code icon labeled 'Code'.

CountryName
"Spain"

#### 6. Find a club that has players from <a specific country>.

```
MATCH (p:Player)-[:REPRESENTS]->(co:Country {name: '<a specific country>'}) MATCH
(p)-[:PLAYS_FOR]->(c:Club) RETURN DISTINCT c.name AS ClubName;
```

```
neo4j$ MATCH (p:Player)-[:REPRESENTS]->(co:Country {name: 'Spain'}) MATCH (p)-[:PLAYS_FOR]->(c:Club) RETURN DISTINCT c.name AS ClubName;
```

	ClubName
1	"FC Bayern Muenchen"
2	"Real Madrid CF"
3	"SSC Napoli"
4	"FC Barcelona"
5	"Chelsea FC"
6	"Atletico Madrid"
7	

Started streaming 9 records after 15 ms and completed after 27 ms.

**7. Find all players who play at <a specific club>, returning in ascending order of age.**

```
MATCH (p: Player)-[: PLAYS_FOR]->(c: Club {name: '<a specific club>'}) RETURN p.name AS PlayerName, p.age AS Age ORDER BY p.age ASC;
```

```
neo4j$ MATCH (p:Player)-[:PLAYS_FOR]->(c:Club {name: 'FC Barcelona'}) RETURN p.name AS PlayerName, p.age AS Age ORDER BY p.age ASC;
```

	PlayerName	Age
1	"NEYMAR"	22
2	"Jordi ALBA"	25
3	"Sergio BUSQUETS"	25
4	"Alexis SANCHEZ"	25
5	"Pedro RODRIGUEZ"	26
6	"Alexandre SONG"	26
7	"Lionel MESSI"	26

Started streaming 13 records after 9 ms and completed after 11 ms.

**8. Find all <a specific position> players in the national team of <a specific country>, returning in descending order of caps.**

```
MATCH (p: Player {position: '<a specific position>'})-[: REPRESENTS]->(co: Country {name: '<a specific country>'}) RETURN p.name AS PlayerName, p.caps AS Caps ORDER BY p.caps DESC;
```

```
neo4j$ MATCH (p:Player {position: 'Midfielder'})-[:REPRESENTS]-(co:Country {name: 'Spain'}) RETURN p.name AS PlayerName, p.caps AS Caps
```

	PlayerName	Caps
1	"Xavi HERNANDEZ"	131
2	"Xabi ALONSO"	109
3	"Andres INIESTA"	96
4	"Cesc FABREGAS"	88
5	"David SILVA"	79
6	"Sergio BUSQUETS"	65
7		

Started streaming 10 records after 11 ms and completed after 13 ms.

**9. Find all players born in <a specific year> and in the national team of <a specific country>, returning in descending order of caps.**

```
MATCH (p: Player)-[: REPRESENTS]->(co: Country {name: '<a specific country>'}) WHERE p.dob ENDS WITH '<a specific year>' RETURN p.name AS PlayerName, p.caps AS Caps ORDER BY p.caps DESC;
```

```
neo4j$ MATCH (p:Player)-[:REPRESENTS]-(co:Country {name: 'Spain'}) WHERE p.dob ENDS WITH '1984' RETURN p.name AS PlayerName, p.caps...
```

	PlayerName	Caps
1	"Fernando TORRES"	107
2	"Andres INIESTA"	96
3	"Santi CAZORLA"	63

Started streaming 3 records after 7 ms and completed after 9 ms.

**10. Find the players that belong to the same club in the national team of <a specific country>, returning in descending order of international goals.**

```
MATCH (p: Player)-[: REPRESENTS]->(co: Country {name: '<a specific country>'}) MATCH (p)-[: PLAYS_FOR]->(c: Club) WITH c, p ORDER BY p.goals DESC RETURN c.name AS ClubName, COLLECT(p.name) AS Players, COLLECT(p.goals) AS Goals;
```

```
neo4j$ MATCH (p:Player)-[:REPRESENTS]→(co:Country {name: 'Spain'}) MATCH (p)-[:PLAYS_FOR]→(c:Club) WITH c, p ORDER BY p.goals DESC...
```

	ClubName	Players	Goals
1	"Atletico Madrid"	["David VILLA", "JUANFRAN", "KOKE", "Diego COSTA"]	[56, 1, 0, 0]
2	"Chelsea FC"	["Fernando TORRES", "Cesar AZPILICUETA"]	[36, 0]
3	"Manchester City FC"	["David SILVA"]	[20]
4	"Real Madrid CF"	["Xabi ALONSO", "Sergio RAMOS", "Iker CASILLAS"]	[15, 9, 0]
5	"FC Barcelona"	["Pedro RODRIGUEZ", "Cesc FABREGAS", "Xavi HERNANDEZ", "Andres INIESTA", "Jordi ALBA", "Gerard PIQUE", "Sergio BUSQUETS"]	[14, 13, 12, 12, 5, 4, 0]
6	"Arsenal FC"	["Santi CAZORLA"]	[11]
7			

Started streaming 9 records after 12 ms and completed after 14 ms.

## 11. How many players are born in <a specific year>?

MATCH (p: Player) WHERE p.dob ENDS WITH '<a specific year>' RETURN COUNT(p) AS  
NumberOfPlayersBornInSpecificYear;

```
neo4j$ MATCH (p:Player) WHERE p.dob ENDS WITH '1985' RETURN COUNT(p) AS NumberOfPlayersBornInSpecificYear;
```

	NumberOfPlayersBornInSpecificYear
1	66

## 12. Which age has the highest participation in the 2014 FIFA World Cup?

MATCH (p:Player) RETURN p.age AS Age, COUNT(p) AS NumberOfPlayers ORDER BY  
NumberOfPlayers DESC LIMIT 1;

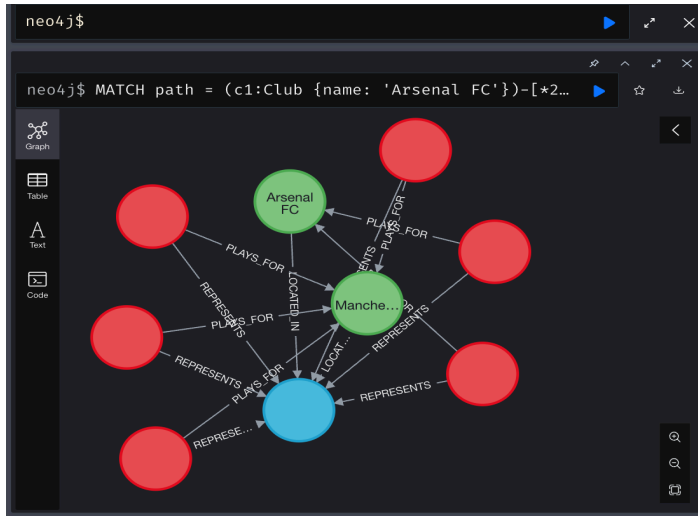
```
neo4j$ MATCH (p:Player) RETURN p.age AS Age, COUNT(p) AS NumberOfPlayers ORDER BY NumberOfPlayers DESC LIMIT 1;
```

	Age	NumberOfPlayers
1	27	77

## 13. Find the path with a length of 2 or 3 between <two specific clubs>.

MATCH path = (c1:Club {name: '<Club A>'})-[\*2..3]-(c2:Club {name: '<Club B>'}) RETURN path;





```
neo4j$ MATCH path = (c1:Club {name: 'Arsenal FC'})-[*2..3]-(c2:Club {name: 'Manchester United FC'}) RETURN path;
```

path
<pre>{   "length": 2.0 }</pre>

```
{
  "start": {
    "identity": 773,
    "labels": [
      "Club"
    ],
    "properties": {
      "country": "England",
      "club_id": 138,
      "name": "Arsenal FC"
    },
    "elementId": "4:08782b7f-e9c9-47dc-ad5d-12d13ff4adca:773"
  },
  ...
}
```

Started streaming 7 records after 1 ms and completed after 6 ms.

#### 14. Which are the top 5 countries with players who have the highest average number of international goals?

```
MATCH (p:Player)-[:REPRESENTS]->(co:Country) RETURN co.name AS CountryName, AVG(p.goals) AS AverageGoals ORDER BY AverageGoals DESC LIMIT 5;
```

```
neo4j$ MATCH (p:Player)-[:REPRESENTS]→(co:Country) RETURN co.name AS CountryName, AVG(p.goals) AS AverageGoals ORDER BY AverageGoals
```

	CountryName	AverageGoals
1	"Germany"	9.521739130434781
2	"Spain"	9.434782608695654
3	"Netherlands"	7.0
4	"Uruguay"	6.217391304347826
5	"Ivory Coast"	6.130434782608697
6	"Portugal"	6.130434782608695
7		

Started streaming 32 records after 6 ms and completed after 8 ms.

## Self-Designed Cypher Queries

### 1. Identifying the Top 3 Clubs with the Most Players in the 2014 FIFA World Cup

```
MATCH (p:Player)-[:PLAYS_FOR]→(c:Club)
RETURN c.name AS ClubName, COUNT(p) AS NumberOfPlayers
ORDER BY NumberOfPlayers DESC
LIMIT 3;
```

```
neo4j$ MATCH (p:Player)-[:PLAYS_FOR]→(c:Club) RETURN c.name AS ClubName, COUNT(p) AS NumberOfPlayers ORDER BY NumberOfPlayers DESC ...
```

	ClubName	NumberOfPlayers
1	"FC Bayern Muenchen"	15
2	"Manchester United FC"	14
3	"FC Barcelona"	13

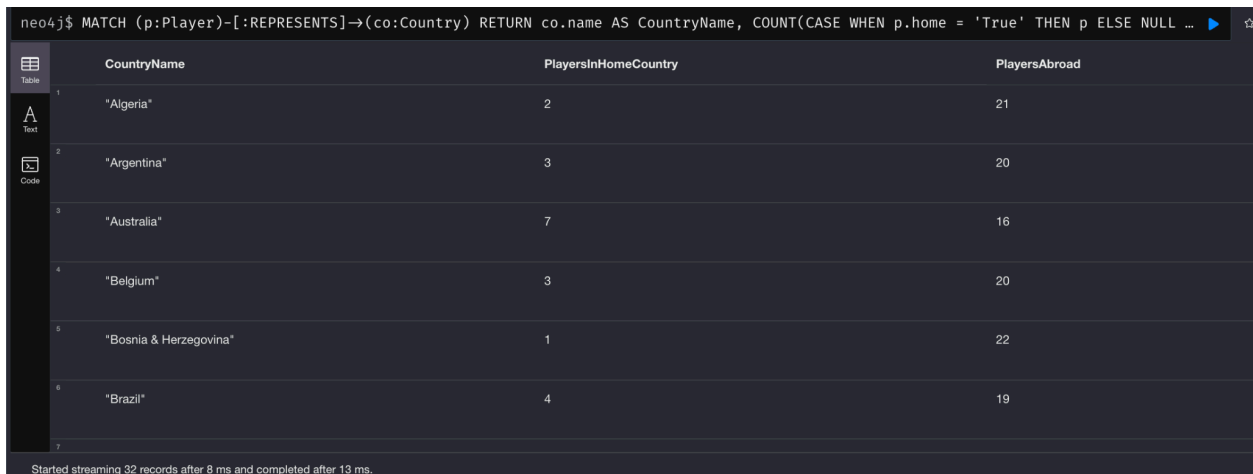
Understanding which clubs had the most players participating in the 2014 FIFA World Cup provides valuable insights into several aspects of global football dynamics. Clubs that contribute a large number of players to the World Cup are often seen as strong and reputable institutions in football. This indicates their success in attracting and developing top-tier talent.

Analysing which clubs have the most players in the World Cup can highlight those with excellent player development programs. These clubs likely have robust youth academies and training facilities that nurture talent effectively.

Clubs with a significant presence in the World Cup tend to have a wider market influence and global reach. This enhances their brand value and can lead to increased fanbase and commercial opportunities. The presence of many players in the World Cup can affect a club's performance in domestic leagues due to player fatigue and injuries. Understanding this relationship can help clubs manage their squads better during international tournaments.

## 1. Players from Each Country: Home Country vs. Abroad

```
MATCH (p:Player)-[:REPRESENTS]->(co:Country)
RETURN co.name AS CountryName,
       COUNT(CASE WHEN p.home = 'True' THEN p ELSE NULL END) AS PlayersInHomeCountry,
       COUNT(CASE WHEN p.home = 'False' THEN p ELSE NULL END) AS PlayersAbroad
ORDER BY CountryName ASC;
```



The screenshot shows a Neo4j query interface with the following query: `neo4j$ MATCH (p:Player)-[:REPRESENTS]->(co:Country) RETURN co.name AS CountryName, COUNT(CASE WHEN p.home = 'True' THEN p ELSE NULL ...`. The result is displayed as a table with three columns: CountryName, PlayersInHomeCountry, and PlayersAbroad. The table contains 7 rows of data, ordered by CountryName ASC. The status bar at the bottom indicates: "Started streaming 32 records after 8 ms and completed after 13 ms."

	CountryName	PlayersInHomeCountry	PlayersAbroad
1	"Algeria"	2	21
2	"Argentina"	3	20
3	"Australia"	7	16
4	"Belgium"	3	20
5	"Bosnia & Herzegovina"	1	22
6	"Brazil"	4	19
7			

Analysing the number of players from each country who play in their home country versus those who play abroad is crucial for several reasons. Countries with a high number of players staying in their home leagues may have strong domestic competitions that provide sufficient opportunities and quality for top players to develop and succeed locally.

Countries that export many players to foreign leagues can benefit from these players gaining international experience and exposure. This can enhance their skills and performance when they represent their national teams. The number of players playing abroad can reflect the economic and infrastructural conditions of the home country's football environment. Countries with less developed leagues might see more talent migrating abroad for better opportunities.

Players staying in their home country can contribute to the local culture and economy, fostering a stronger connection with local fans. Conversely, players abroad can become ambassadors for their home country, promoting its football culture globally.

Understanding these patterns helps in assessing the effectiveness of different player development pathways. It provides insights into whether domestic leagues or international experiences are more beneficial for player growth.

## **Graph Database VS Relational Database**

Graph database stores entities as nodes and relationships as edges as a 'relationship first' type data model. Relational databases store data in tables and use 'JOIN' for fast querying- using tables to represent entities and foreign keys to represent relationships. In relational databases, many to many relationships require join tables. This causes complexity and inefficiency as the number of relationships grows so graph databases offer fewer challenges as they are designed to represent and query complex relationships [1].

Graph Databases are optimised for traversing relationships. Queries that involve multiple hops (e.g., finding the shortest path between two nodes) are performed efficiently because relationships are first-class citizens. Relational Databases which multi-hop queries often require multiple joins, which can be computationally expensive and slow, especially with large datasets.

Graph Databases have schema-optional features that allow for flexible data structures without the requirement of a predefined schema. This is useful for scenarios where data models are not fixed and can evolve. Relational Databases are schema-based, meaning they require a predefined schema that can be considered inflexible and challenging to modify as in this current day and age, it cannot be guaranteed that the data structure will remain fixed and static across time.

Graph Databases can represent data in a way that aligns closely with real-world scenarios, making it easier for developers to conceptualise and work with the data. Relational Databases represent data through tables, which can be less intuitive for modelling real-world relationships and hierarchies. Overall, the client/user can comprehend the data more due to the network structure that graph databases offer.

Graph Databases are more scalable [2] by distributing both data and query loads across multiple machines. They handle large-scale graph traversals efficiently. Relational Databases are limited by the hardware of a single machine so it is complex and less efficient for highly relational queries.

## **Graph Data Science**

Graph data science (GDS) allows data science techniques to be applied to graph data structures to identify behavioural characteristics that can be used to build predictive and prescriptive models.

Graph data science and machine learning algorithms compute distances and paths, similarity, and communities; simulate the effects of changes in the graph; and allow predictions to be made for inferring new nodes or edges or classifying whole graph structures. [3]

### **Friendship Paradox**

The friendship paradox is a phenomenon observed in social networks where most people have fewer friends than their friends do, on average[4]. This occurs because individuals with more friends are more likely to be friends with many people, skewing the average.

If we consider a network where nodes represent people and edges represent friendships, the average number of friends of friends tends to be higher than the average number of friends an individual has. This paradox can be used to identify influential individuals in a network, as those with more connections (friends) have a broader reach and impact.

Graph traversal algorithms are fundamental in exploring and analysing the structure of social networks. Two primary methods are Breadth-First Search (BFS) and Depth-First Search (DFS).

**Breadth-First Search (BFS):** BFS explores a graph layer by layer, starting from a given node and exploring all its neighbours before moving to the next layer. BFS is used for finding the shortest path in unweighted graphs, identifying all nodes within a certain distance from a source node, and detecting connected components.

**Depth-First Search (DFS):** DFS explores a graph by going as deep as possible down one branch before backtracking and exploring other branches. DFS is useful for topological sorting, detecting cycles in graphs, and exploring all possible paths in a network.

We're able to use the Friendship paradox in a social network analysis using Graph Data Science. Understanding that this phenomenon results from a sampling bias that causes nodes to be counted in proportion to their degree, and it extends beyond popularity to other individual traits such as the amount of content in social networks [5], allows several practical applications in a social network analysis. If a marketing company were to push certain products on social media, identifying which influencers to campaign with will be a vital element in ensuring a successive promotion. Identifying influencers with the right level of power/influence to attract the right crowd will be key and that can be done with the following applications of Graph Data Science:

Influencer Identification: Applying centrality measures and leveraging the friendship paradox, we can identify key influencers in a social network. These influencers typically have a high degree of centrality- they have many connections and will have a significant impact on the network.

Information Diffusion: Graph traversal algorithms, especially BFS can be used to analyse how info spreads across the network. Understanding the pattern of these paths allows influential marketing strategies to be identified and implemented.

Community Identification: Analysing communities in a social network shows the social structure. Identifying these dynamics and interactions will allow the marketing company to single out the key person/node who will be optimal for product promotion

## References

- [1] Memgraph. (08/05/2023). Graph database vs. relational database. Memgraph. Retrieved from <https://memgraph.com/blog/graph-database-vs-relational-database>
- [2] Nebula Graph. (12/10/2022). Graph database vs. relational database. Nebula Graph. Retrieved from <https://www.nebula-graph.io/posts/graph-database-vs-relational-database>
- [3] Gartner. (n.d.). Graph data science (GDS). Gartner IT Glossary. Retrieved from <https://www.gartner.com/en/information-technology/glossary/graph-data-science-gds>
- [4] Holme, P. (2019). The friendship paradox in real and model networks. Applied Network Science, 4(1), 1-10.  
<https://appliednetsci.springeropen.com/articles/10.1007/s41109-019-0190-8#:~:text=The%20friendship%20paradox%20was%20first,collection%20of%20the%20individuals%20themselves>
- [5] Hodas, NO, Kooti F, Lerman K (2013) Friendship paradox redux: Your friends are more interesting than you. Int AAAI Conf Web Soc Med (ICWSM) 13:8–10.