CITS5504 Data Warehousing

Project 2 – Graph Database Design and Query

Group 17: Trieu Huynh (23728208) and Pritam Suwal Shrestha (23771397)

26th May 2024

Table of Contents

1. DATASET

The dataset contains information about players who participated in the 2014 FIFA World Cup, an international soccer tournament where players represent their national teams rather than their club teams. This dataset, formatted as a CSV file, allows for analysis of the profiles and backgrounds of athletes competing in FIFA.

1.1 Dataset columns

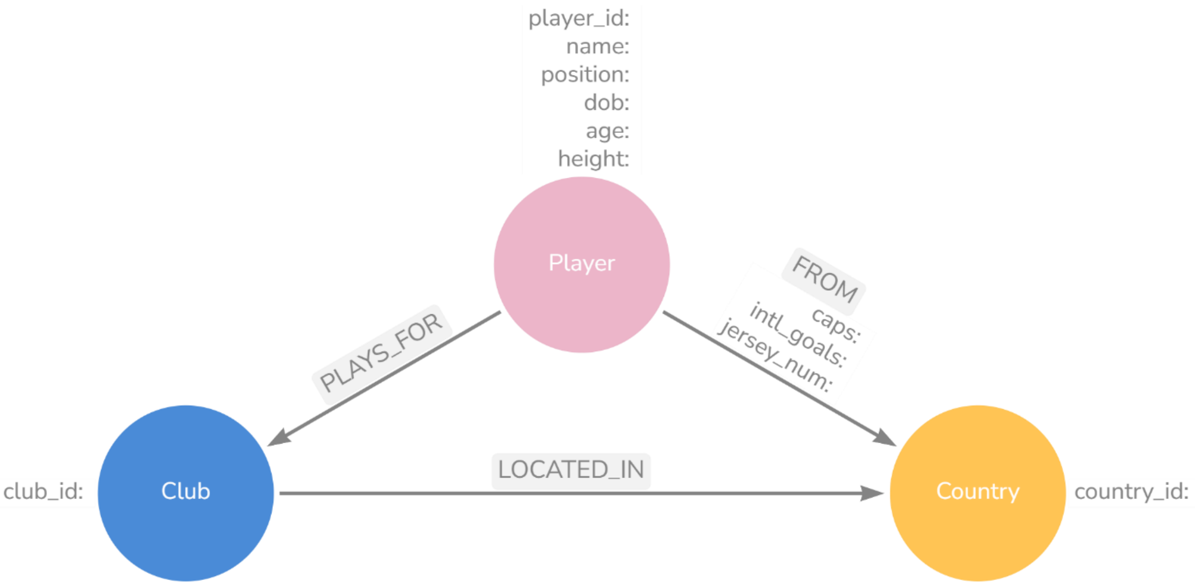
The columns present in the dataset are:

- **Player ID**: a unique identifying number for each player.
- **Player**: the name of the player.
- **Position**: the position played by the player, such as Forward, Midfielder, Defender, or Goalkeeper.
- **Number**: the jersey number worn by the player during FIFA matches.
- **Club**: the club team the player belongs to.
- **Club (country)**: the country where the player's club is based.
- **D.O.B.**: the date of birth of the player, formatted as DD.MM.YYYY.
- **Age**: the age of the player at the time of the 2014 FIFA World Cup.
- **Height (cm)**: the height of the player in centimetres.
- **Country**: the national team country of the player.
- **Caps**: the number of times the player has represented their national team in international matches before the 2014 FIFA World Cup.
- **International goals**: the number of goals scored by the player for their national team before the 2014 FIFA World Cup.
- **Plays in home country?**: a Boolean value indicating whether the player plays for a club in their home country.

2. GRAPH DATABASE DESIGN

Based on the source data and the queries that need to be answered, the following design was implemented for the graph database.

2.1 Property graph



2.2 Design process and Rationale

The database design began with planning the nodes and their relationships. Three distinct entities were identified – **Player**, **Club**, and **Country** – which formed the nodes. After establishing the nodes, the relationships between them were defined based on the interactions and dependencies between them. Appropriate labels were then assigned to the nodes and relationships to ensure clarity and ease of querying.

Relevant properties were added to support the required queries, with an emphasis on retaining as much information from the dataset as possible. However, the decision was made to exclude the **Plays in home country?** column as this information can be derived from a database query that compares the country the player is from and the country that their club is located. Additionally, only one query required this data, so constructing this query would not be overly taxing. By removing this redundancy, data integrity is improved, resulting in a more maintainable database design.

It was noted that the **Height** column is not used in any of the current queries, however it was retained to provide flexibility for future queries that might require this information.

Each club and country were given a unique identifier, which was not present in the original dataset. This addition ensures precise identification and management of these entities, facilitating future updates if needed, such as in cases where names change.

The scalability of the design was also a consideration. By using unique identifiers and maintaining a flexible schema, the database can accommodate new types of nodes and relationships, allowing it to evolve to meet future requirements.

3. ETL PROCESS

The ETL (Extract, Transform, Load) process for creating node and relationship using Python is outlined below. For further technical implementation details, please refer to the included Jupyter notebook.

3.1 Player node

Firstly, the player node was created. To ensure accuracy of the player data, it was confirmed that all players were unique and there were no duplicates. The dataset was examined for missing values, then the player ID and other properties were then extracted.

```python
print(f"Number of instances in the dataset: {len(fifa_df)}")

# Confirm that there are no duplicated players based on Player id
num_unique_players = len(fifa_df["Player id"].unique())
print(f"Number of unique players: {num_unique_players}")
```

Python

```
Number of instances in the dataset: 736
Number of unique players: 736
```

```python
# Create Player node
player_node = fifa_df[["Player id", "Player", "Position", "D.O.B", "Age", "Height (cm)"]]

# Rename columns
player_node.rename(columns={
    "Player id": "player_id",
    "Player": "name",
    "Position": "position",
    "D.O.B": "dob",
    "Age": "age",
    "Height (cm)": "height"
}, inplace=True)

player_node.to_csv('./data/player_node.csv', index=False)
```
Python

3.2 Club node

Next, the unique values for club names were extracted and inspected for any errors. Initially, there was some uncertainty regarding the club's name **1. FC Nuernberg** due to the presence of a number. Additionally, potential duplicates like **Manchester City FC** and **Manchester United FC** identified. However, after verifying these names against their official websites, it was confirmed that these were the correct names.

To uniquely identify each club, a **club_id** was added.

```python
# Create Club node
club_node = pd.DataFrame(fifa_df["Club"].sort_values().unique(), columns=["Club"])

# Add key column 'club_id'
club_node["club_id"] = range(1, len(club_node) + 1)
# Reorder columns so 'club_id' is first
club_node = club_node[["club_id", "Club"]]

# Merge 'club_id' back into fifa_df
fifa_df = pd.merge(fifa_df, club_node, on="Club", how="left")
# fifa_df[fifa_df["Club"] == "Tigres UANL"] # uncomment to see that merge worked correctly

# Rename columns
club_node.rename(columns={"Club": "club"}, inplace=True)

club_node.to_csv('./data/club_node.csv', index=False)
```
Python

3.3 Country node

There were two columns that represented country names, **Club (country)** and **Country**. To determine which column to use for populating the country node, the set difference between the two columns was calculated. It was found that **Club (country)** had unique values, whereas all values in **Country** were also present in **Club (country)**. As such, **Club (country)** was used to populate the country node.

The country names were manually inspected for any errors and anomalies. Then, as with the club node, a unique identifier, **country_id** was added.

```python
# Create Country node

# Check which column to use to extract countries - 'Club (country)' or 'Country'
club_ctry = set(fifa_df["Club (country)"].unique())
ctry = set(fifa_df["Country"].unique())
print(f"Number of countries that are unique to the 'Club (country)' column: {len(club_ctry - ctry)}")
print(f"Number of countries that are unique to the 'Country' column: {len(ctry - club_ctry)}")

# Use 'Club (country)' to construct Country node
country_node = pd.DataFrame(fifa_df["Club (country)"].sort_values().unique(), columns=["Country"])
# Add key column 'country_id'
country_node["country_id"] = range(1, len(country_node) + 1)
# Reorder columns so 'country_id' is first
country_node = country_node[["country_id", "Country"]]

# Merge 'country_id' back into fifa_df
fifa_df = pd.merge(fifa_df, country_node, left_on="Club (country)", right_on="Country", how="left")
fifa_df.rename(columns={"country_id": "club_country_id", "Country_x": "Country"}, inplace=True)
fifa_df.drop(columns=["Country_y"], inplace=True) # Drop duplicated column from merge

fifa_df = pd.merge(fifa_df, country_node, on="Country", how="left")
# fifa_df.head() # uncomment to see the merge worked correctly

# Rename column
country_node.rename(columns={"Country": "country"}, inplace=True)

country_node.to_csv('./data/country_node.csv', index=False)
```

Python

```
Number of countries that are unique to the 'Club (country)' column: 19
Number of countries that are unique to the 'Country' column: 0
```

3.4 Relationships

The **club_id** and **country_id** were merged back into the original dataset to enable look up of node instances by their IDs. Following this, the relationships between the nodes were extracted by obtaining the corresponding node keys. Utilising integer IDs instead of string identifiers may allow for faster look ups in Neo4j, improving performance.

The **FROM** relationship also included additional properties, which were extracted and included in the CSV.

```python
# Create PLAYS_FOR relationship
rel_plays_for = fifa_df[["Player id", "club_id"]]
# Rename column
rel_plays_for.rename(columns={"Player id": "player_id"}, inplace=True)
rel_plays_for.to_csv('./data/rel_plays_for.csv', index=False)
```

Python

```python
# Create FROM relationship
rel_from = fifa_df[["Player id", "country_id", "Caps", "International goals", "Number"]]
rel_from.rename(columns={
    "Player id": "player_id",
    "Caps": "caps",
    "International goals": "intl_goals",
    "Number": "jersey_num",
}, inplace=True)
rel_from.to_csv('./data/rel_from.csv', index=False)
```

Python

```python
# Create LOCATED_IN relationship
rel_located_in = fifa_df[["club_id", "club_country_id"]].drop_duplicates()

# Confirm that number of relationships matches the number of clubs
print(f"The number of relationships match the number of clubs: {len(rel_located_in) == len(club_node)}

rel_located_in.to_csv('./data/rel_located_in.csv', index=False)
```

Python

3.5 Transformations and loading

Column names were renamed for easier manipulation in Neo4j. Numeric fields, such as age, height, and international goals, were converted to integers. Additionally, players' date of birth was converted to a date type.

Finally, the prepared CSV were imported into Neo4j, allowing for database queries and analysis. The data was inspected post-loading to ensure that the integrity was maintained.

The Cypher queries used to load the data are presented below.

Step 1: Load nodes

```
CALL apoc.import.csv(
  [
    {fileName: 'file:/club_node.csv', labels: ['Club']},
    {fileName: 'file:/country_node.csv', labels: ['Country']},
    {fileName: 'file:/player_node.csv', labels: ['Player']}
  ],
  [],
  {}
)
```

Step 2: Convert properties to correct data types for Player node

```
MATCH (p:Player)
SET p.player_id = toInteger(p.player_id)
SET p.age = toInteger(p.age)
SET p.height = toInteger(p.height)
SET p.dob = date({ year: toInteger(substring(p.dob, 6, 4)), month: toInteger
(substring(p.dob, 3, 2)), day: toInteger(substring(p.dob, 0, 2))})
```

Step 3: Convert properties to correct data types for Club node

```
MATCH (c:Club)
SET c.club_id = toInteger(c.club_id)
```

Step 4: Convert properties to correct data types for Player node

```
MATCH (co:Country)
SET co.country_id = toInteger(co.country_id)
```

Step 5: Load relationship between Player and Club

```
LOAD CSV WITH HEADERS FROM 'file:///rel_plays_for.csv' AS row
MATCH (p:Player {player_id: toInteger(row.player_id)}),
```

```
   (c:Club {club_id: toInteger(row.club_id)})
MERGE (p)-[:PLAYS_FOR]->(c)
```

Step 6: Load relationship between Club and Country

```
LOAD CSV WITH HEADERS FROM 'file:///rel_located_in.csv' AS row
MATCH (c:Club {club_id: toInteger(row.club_id)}),
  (y:Country {country_id: toInteger(row.club_country_id)})
MERGE (c)-[:LOCATED_IN]->(y)
```

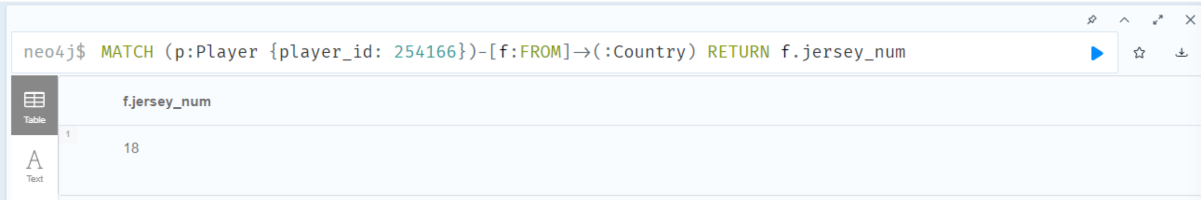
Step 7: Load relationship between Player and Country

```
LOAD CSV WITH HEADERS FROM 'file:///rel_from.csv' AS row
MATCH (p:Player {player_id: toInteger(row.player_id)}),
  (c:Country {country_id: toInteger(row.country_id)})
MERGE (p)-[r:FROM]->(c)
SET r.caps = toInteger(row.caps)
SET r.intl_goals = toInteger(row.intl_goals)
SET r.jersey_num = toInteger(row.jersey_num)
```

4. CYPHER QUERIES

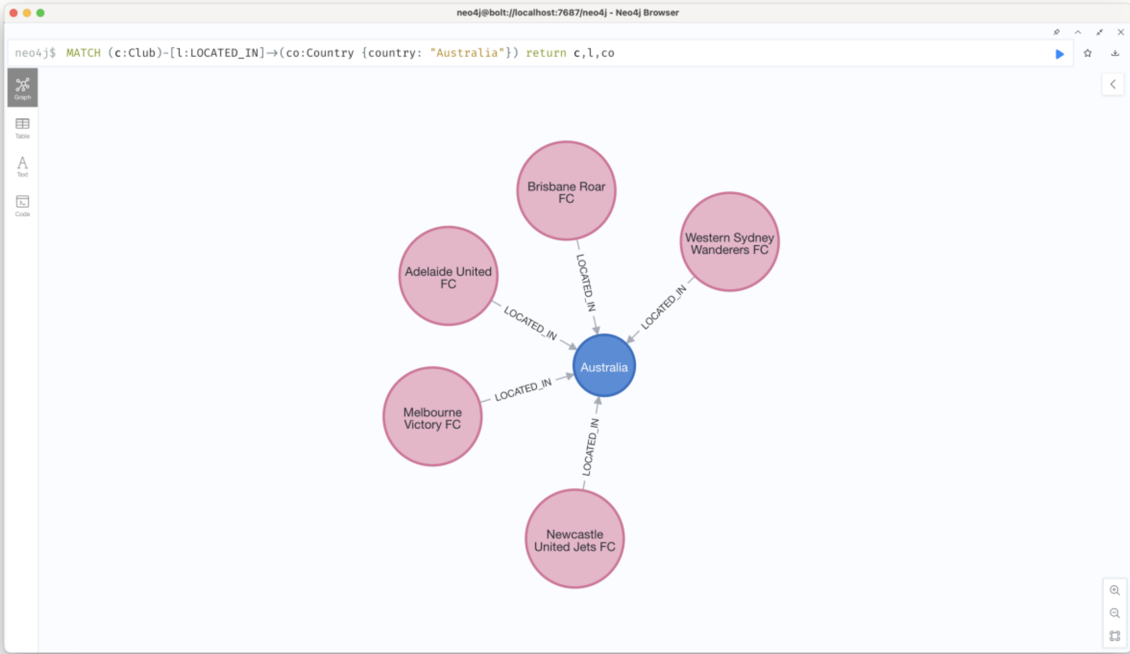
4.1 What is the jersey number of the player with player id <254166>?

```
MATCH (p:Player { player_id: 254166 })-[f:FROM]->(:Country)
RETURN f.jersey_num
```



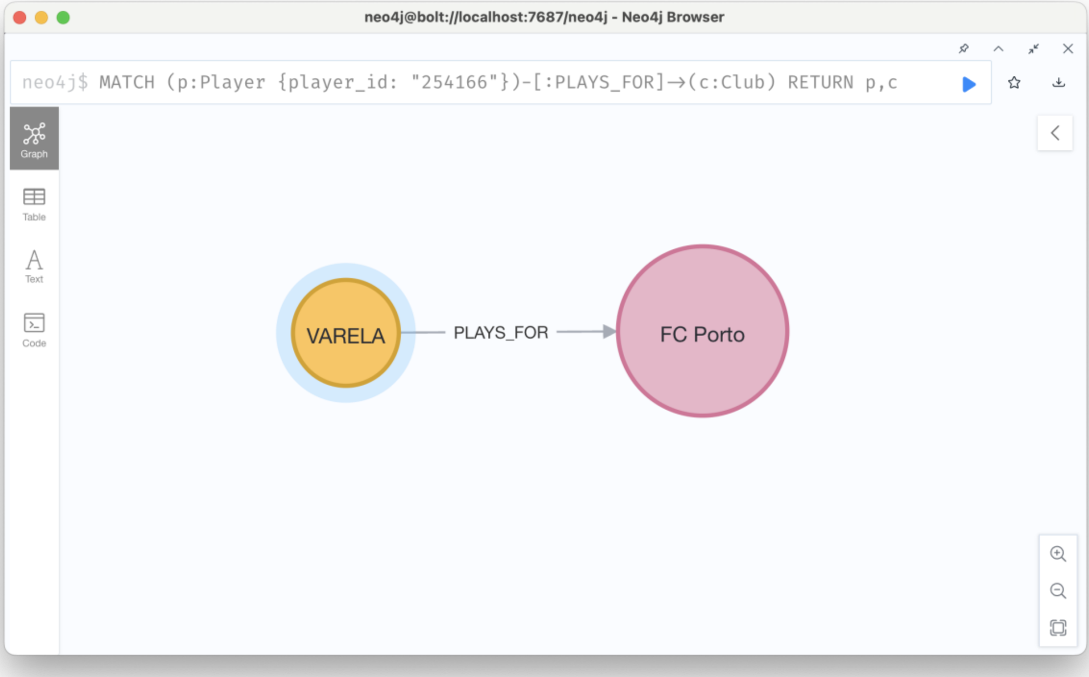
4.2 Which clubs are based in <Australia>?

```
MATCH (c:Club)-[l:LOCATED_IN]->(co:Country { country: "Australia" })
RETURN c, l, co
```



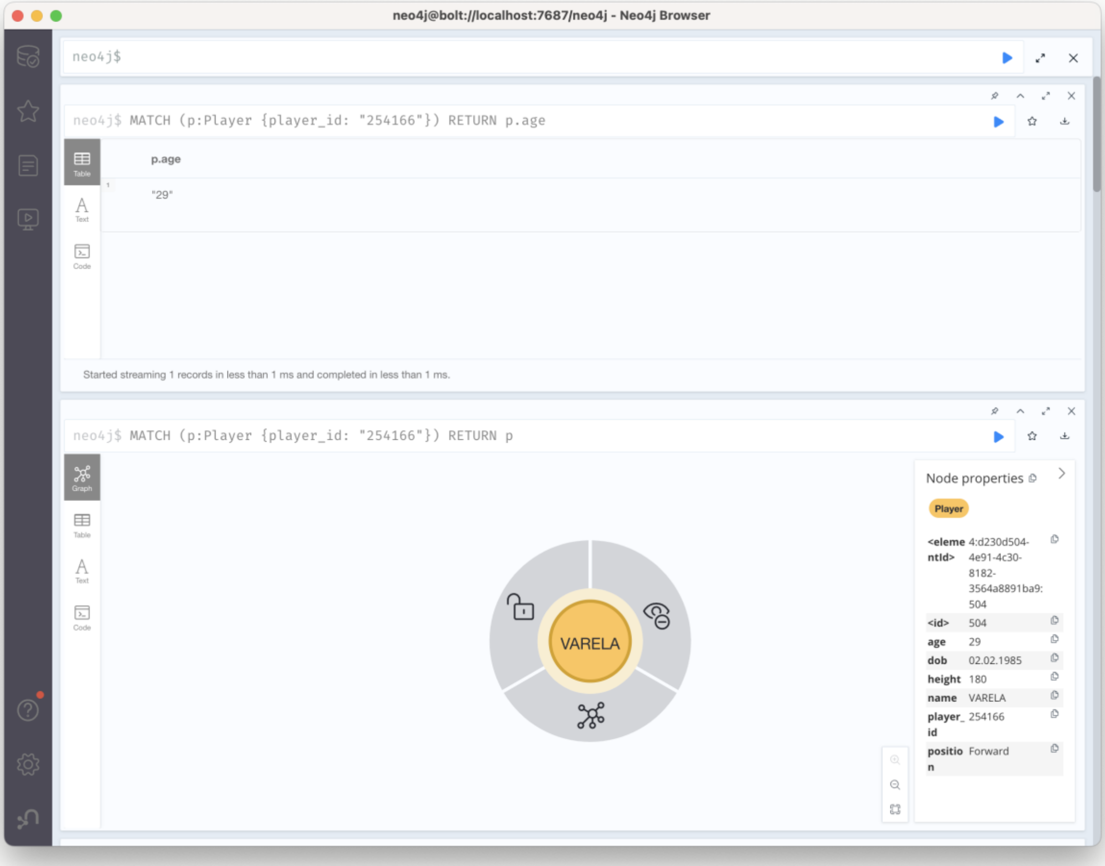
4.3 Which club does <player with id 254166> play for?

```
MATCH (p:Player { player_id: 254166 })-[:PLAYS_FOR]->(c:Club)
RETURN p, c
```



4.4 How old is <player with id 254166>?

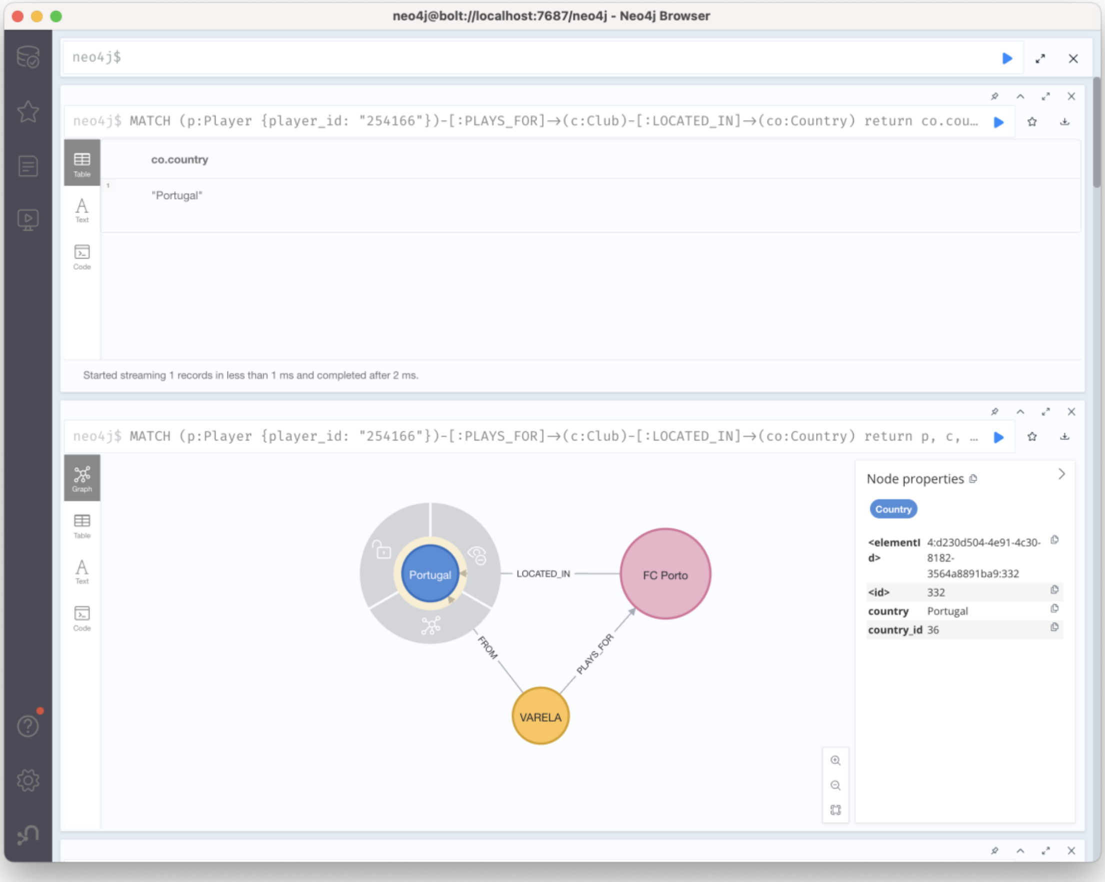
```
MATCH (p:Player { player_id: 254166 })
RETURN p.age
```



4.5 In which country is the club that <player with id 254166> plays for?

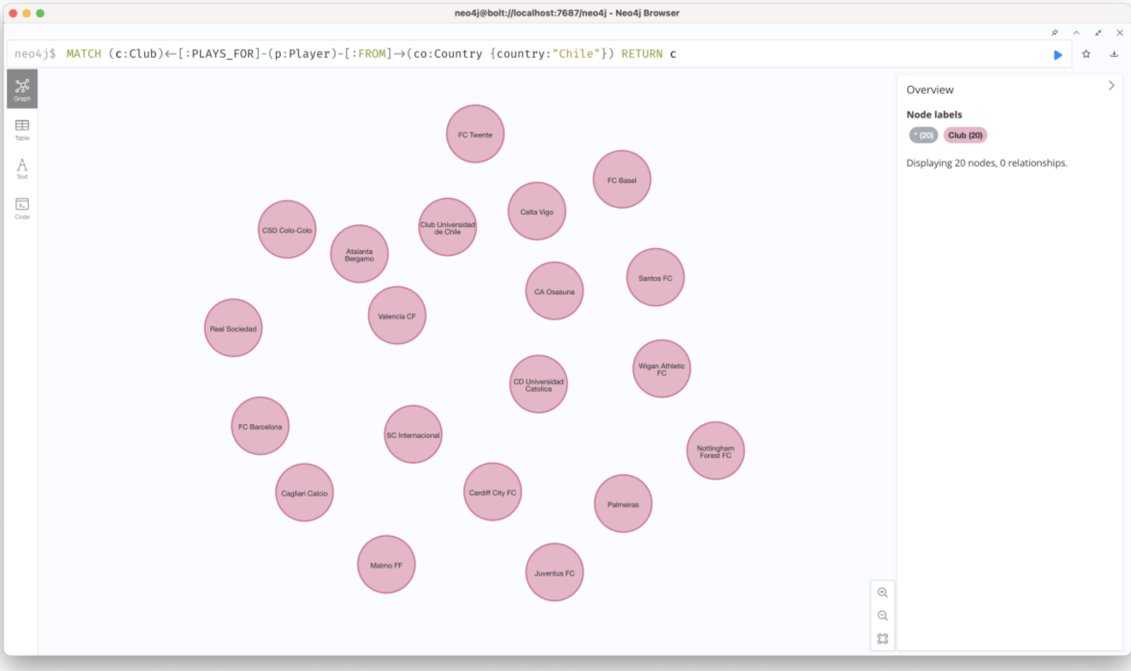
```
MATCH (p:Player { player_id: 254166 })-[:PLAYS_FOR]->(c:Club)-[:LOCATED_IN]->(co:Country)
RETURN co.country
```

```
MATCH (p:Player { player_id: 254166 })-[:PLAYS_FOR]->(c:Club)-[:LOCATED_IN]->(co:Country)
RETURN p, c, co
```



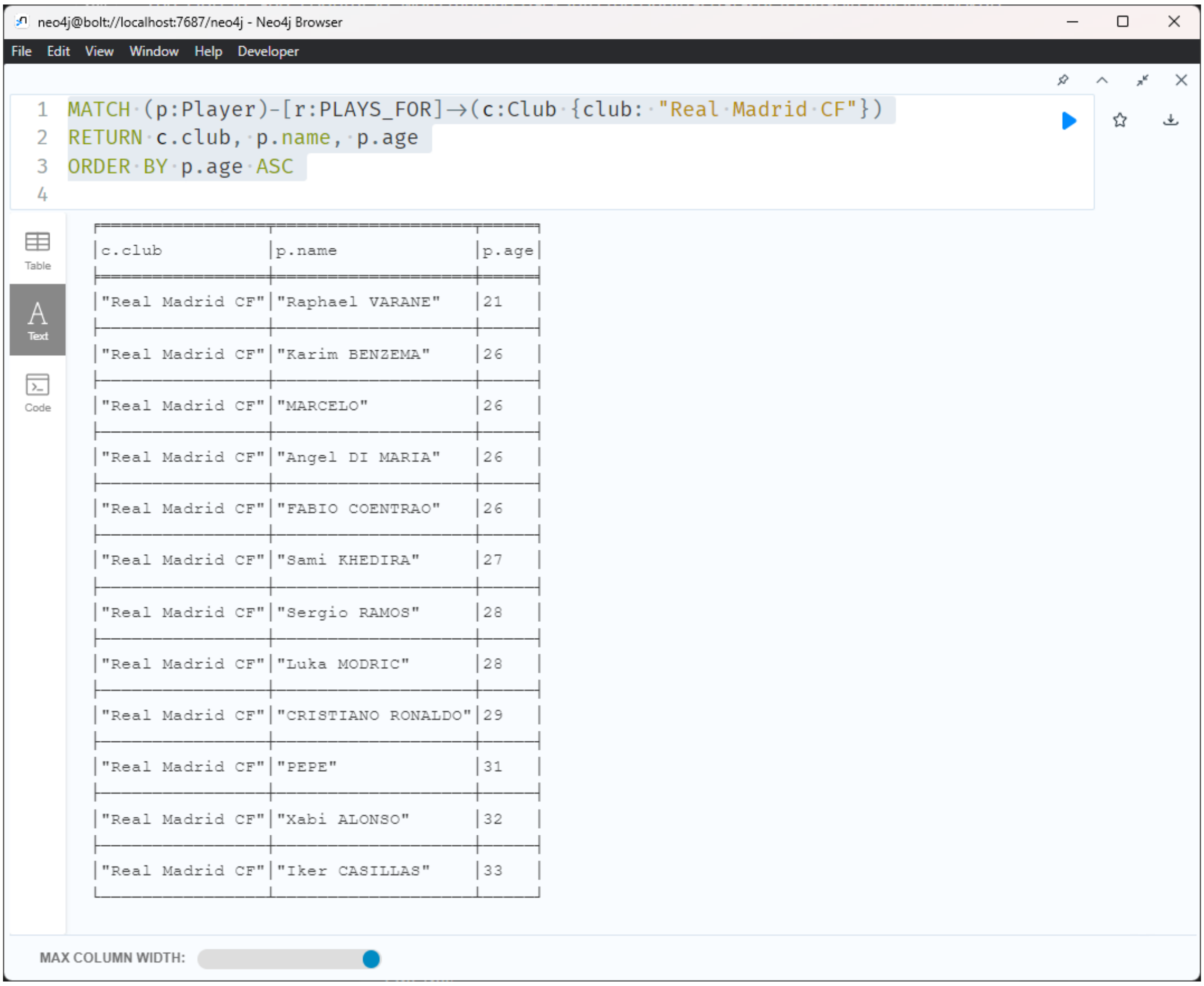
4.6 Find a club that has players from <Chile>.

```
MATCH (c:Club)<-[:PLAYS_FOR]-(p:Player)-[:FROM]->(co:Country { country: "Chile" })
RETURN c
```



4.7 Find all players who play at <Real Madrid CF>, returning in ascending order of age.

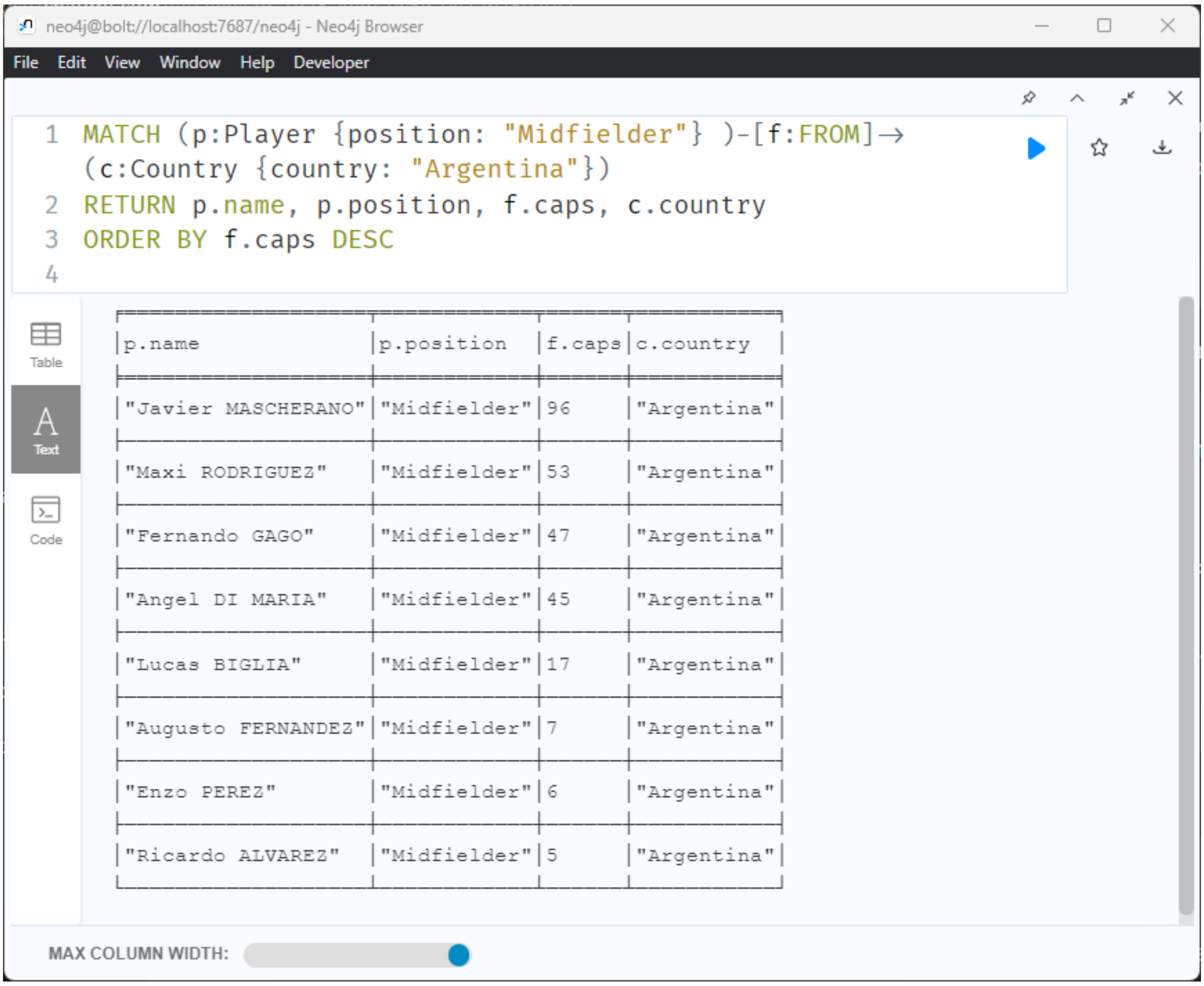
```
MATCH (p:Player)-[r:PLAYS_FOR]->(c:Club { club: "Real Madrid CF" })
RETURN c.club, p.name, p.age
ORDER BY p.age ASC
```



c.club	p.name	p.age
"Real Madrid CF"	"Raphael VARANE"	21
"Real Madrid CF"	"Karim BENZEMA"	26
"Real Madrid CF"	"MARCELO"	26
"Real Madrid CF"	"Angel DI MARIA"	26
"Real Madrid CF"	"FABIO COENTRAO"	26
"Real Madrid CF"	"Sami KHEDIRA"	27
"Real Madrid CF"	"Sergio RAMOS"	28
"Real Madrid CF"	"Luka MODRIC"	28
"Real Madrid CF"	"CRISTIANO RONALDO"	29
"Real Madrid CF"	"PEPE"	31
"Real Madrid CF"	"Xabi ALONSO"	32
"Real Madrid CF"	"Iker CASILLAS"	33

4.8 Find all <midfielder> players in the national team of <Argentina>, returning in descending order of caps.

```
MATCH (p:Player { position: "Midfielder" })-[f:FROM]->(c:Country {country: "Argentina"})
RETURN p.name, p.position, f.caps, c.country
ORDER BY f.caps DESC
```

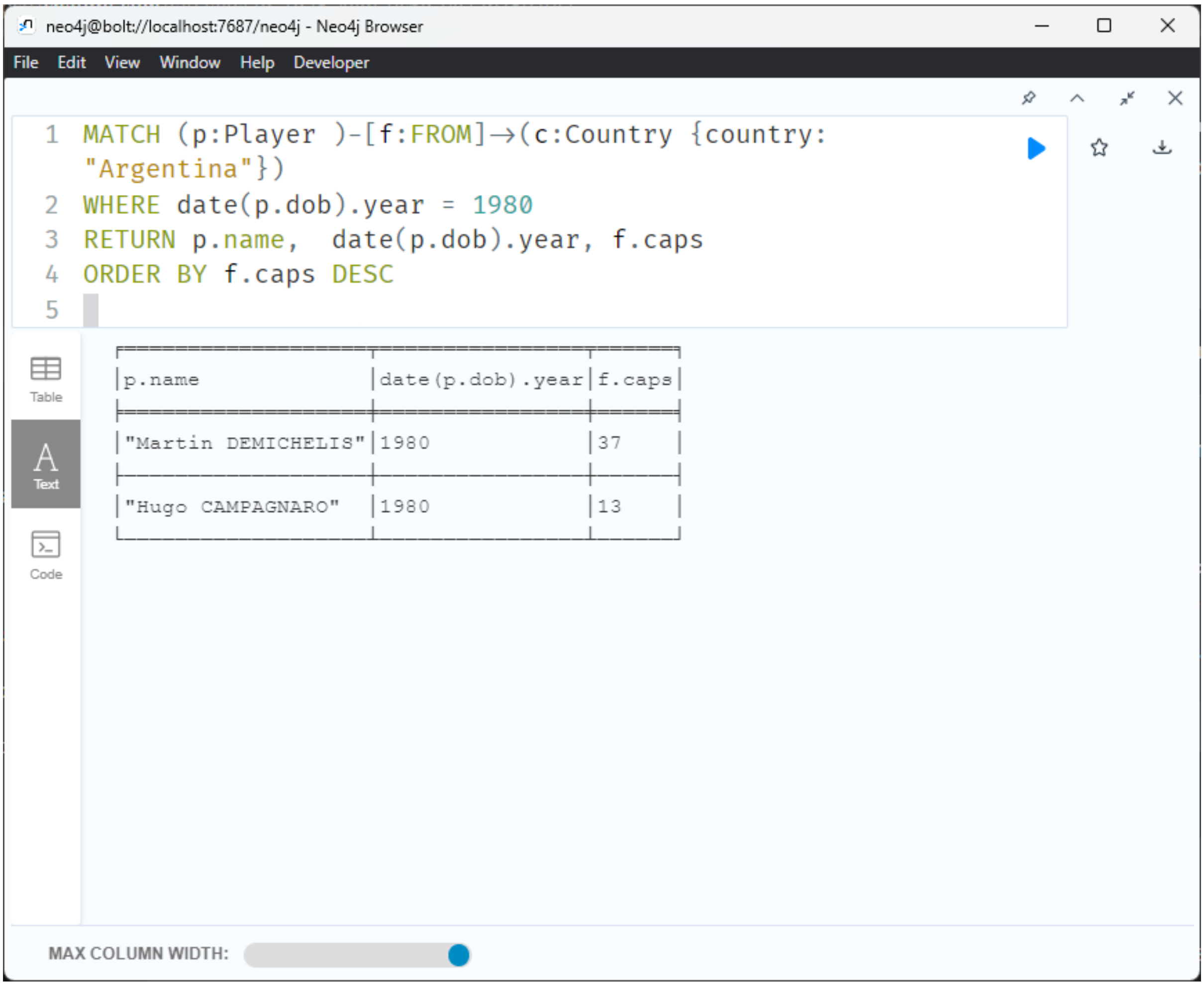


```
MATCH (p:Player {position: "Midfielder"} )-[f:FROM]→(c:Country {country: "Argentina"})
RETURN p.name, p.position, f.caps, c.country
ORDER BY f.caps DESC
```

p.name	p.position	f.caps	c.country
"Javier MASCHERANO"	"Midfielder"	96	"Argentina"
"Maxi RODRIGUEZ"	"Midfielder"	53	"Argentina"
"Fernando GAGO"	"Midfielder"	47	"Argentina"
"Angel DI MARIA"	"Midfielder"	45	"Argentina"
"Lucas BIGLIA"	"Midfielder"	17	"Argentina"
"Augusto FERNANDEZ"	"Midfielder"	7	"Argentina"
"Enzo PEREZ"	"Midfielder"	6	"Argentina"
"Ricardo ALVAREZ"	"Midfielder"	5	"Argentina"

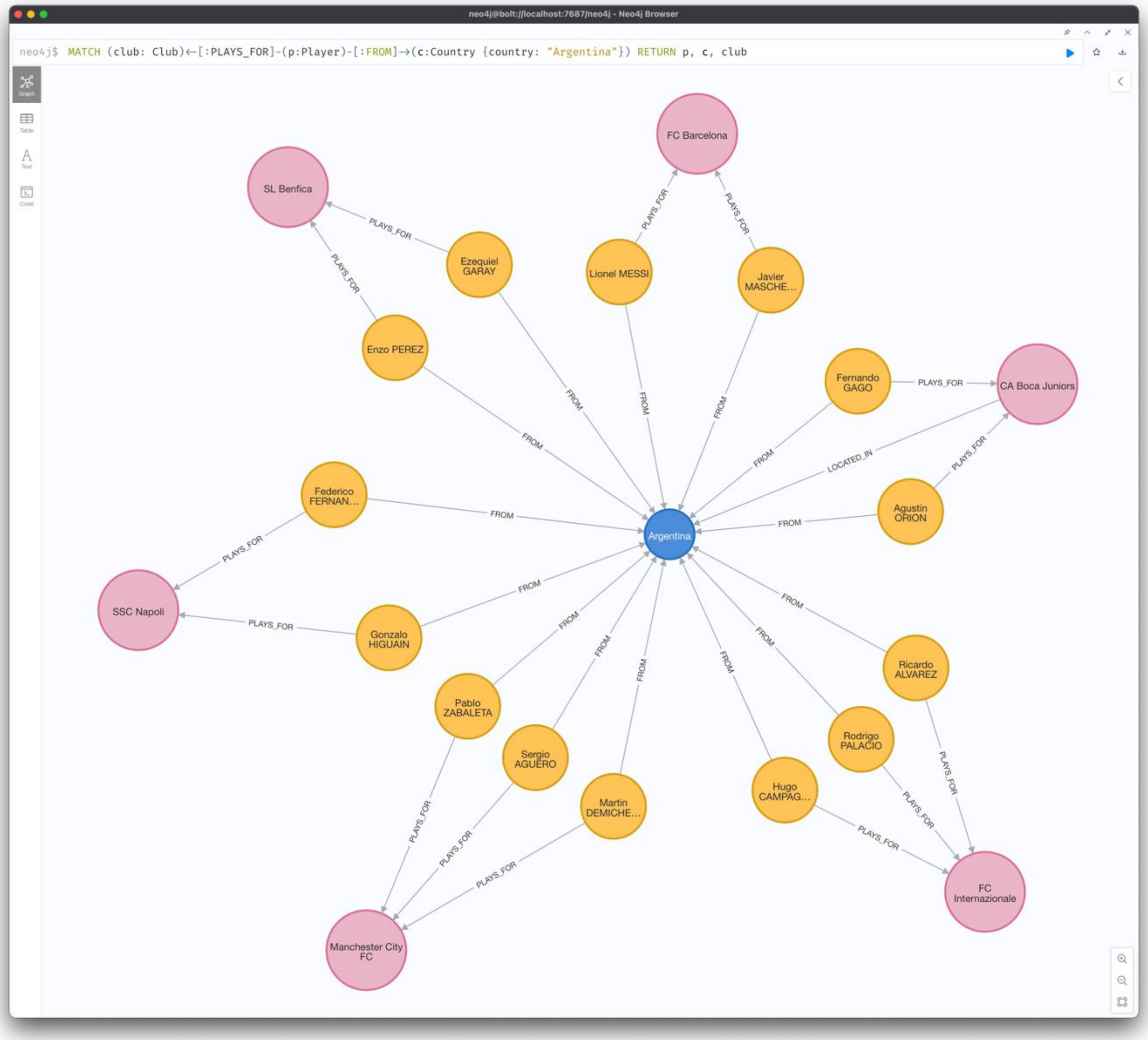
4.9 Find all players born in <1980> and in the national team of <Argentina>, returning in descending order of caps.

```
MATCH (p:Player )-[f:FROM]->(c:Country { country: "Argentina" })
WHERE date(p.dob).year = 1980
RETURN p.name, date(p.dob).year, f.caps
 ORDER BY f.caps DESC
```



4.10 Find the players that belongs to the same club in the national team of <Argentina>, returning in descending order of international goals.

```
MATCH (club:Club)<-[:PLAYS_FOR]-(p:Player)-[r:FROM]->(c:Country { country: "Argentina" })
WITH club, collect({ player: p, intlGoals: r.intl_goals }) AS players
WHERE size(players) > 1
UNWIND players AS playerData
RETURN club.club AS club, playerData.player.name AS name, playerData.intlGoals AS intl_goals
 ORDER BY club.club, intl_goals DESC
```

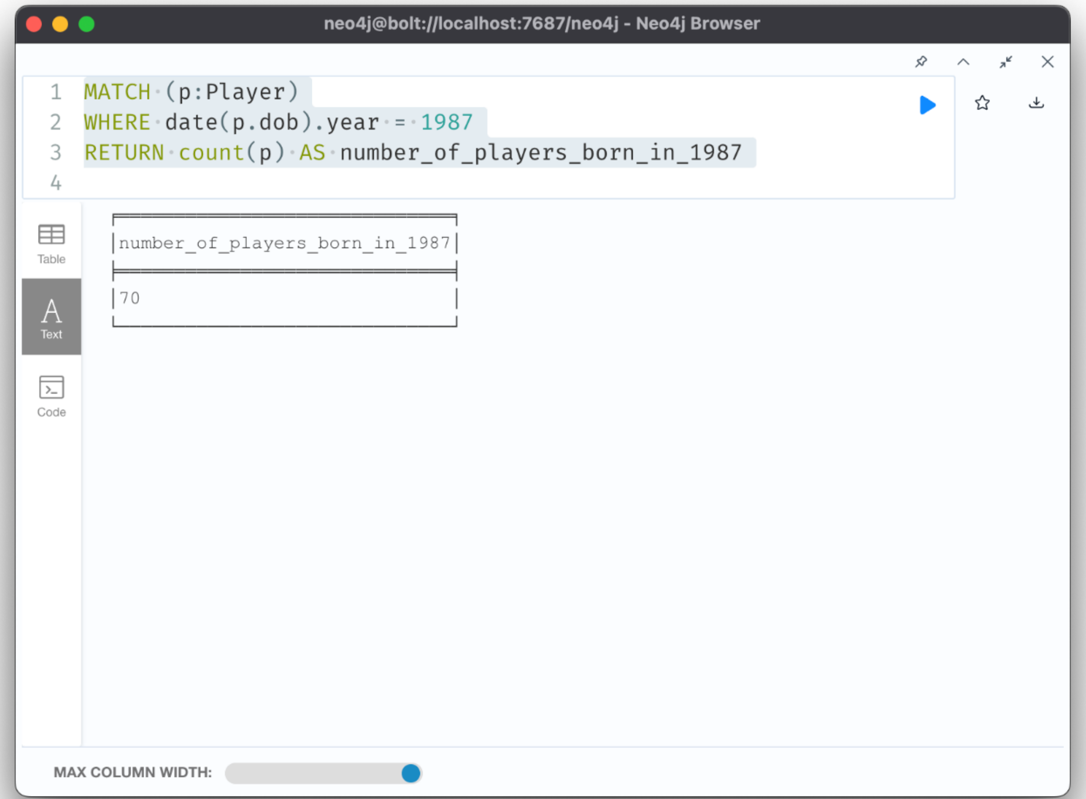


```
MATCH (club:Club)<-[:PLAYS_FOR]-(p:Player)-[r:FROM]->(c:Country {country: "Argentina"})
WITH club, collect({player: p, intlGoals: r.intl_goals}) AS players
WHERE size(players) > 1
UNWIND players AS playerData
RETURN club.club AS club, playerData.player.name AS name, playerData.intlGoals AS intl_goals
ORDER BY club.club, intl_goals DESC
```

club	name	intl_goals
"CA Boca Juniors"	"Agustin ORION"	0
"CA Boca Juniors"	"Fernando GAGO"	0
"FC Barcelona"	"Lionel MESSI"	37
"FC Barcelona"	"Javier MASCHERANO"	2
"FC Internazionale"	"Rodrigo PALACIO"	2
"FC Internazionale"	"Hugo CAMPAGNARO"	0
"FC Internazionale"	"Ricardo ALVAREZ"	0
"Manchester City FC"	"Sergio AGUERO"	21
"Manchester City FC"	"Martin DEMICHELIS"	2
"Manchester City FC"	"Pablo ZABALETA"	0
"SL Benfica"	"Enzo PEREZ"	1
"SL Benfica"	"Ezequiel GARAY"	0
"SSC Napoli"	"Gonzalo HIGUAIN"	20
"SSC Napoli"	"Federico FERNANDEZ"	2

4.11 Count how many players are born in <1987>.

```
MATCH (p:Player)
WHERE date(p.dob).year = 1987
RETURN COUNT(p) AS number_of_players_born_in_1987
```

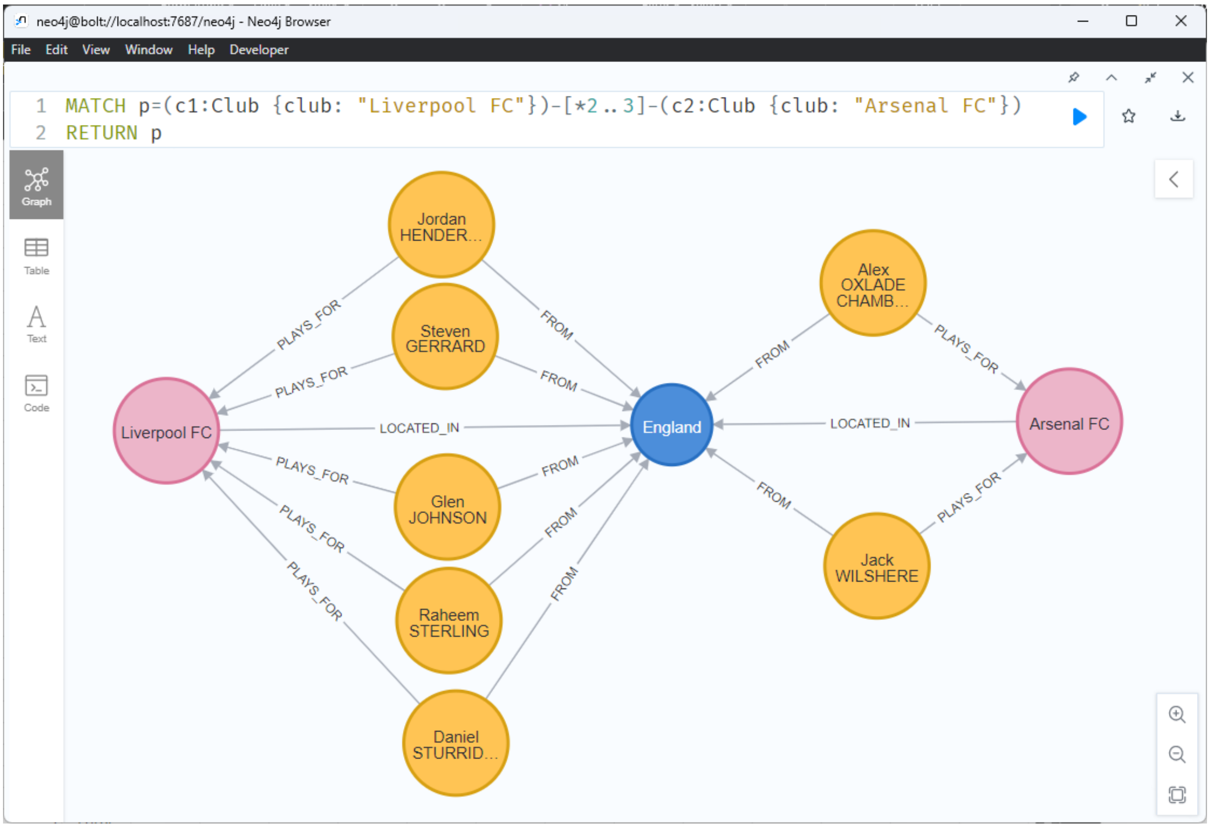


4.12 Which age has the highest participation in the 2014 FIFA World Cup?

```
MATCH (p:Player)
WITH p.age AS age, count(p) AS playerCount
RETURN age, playerCount
ORDER BY playerCount DESC
LIMIT 1
```

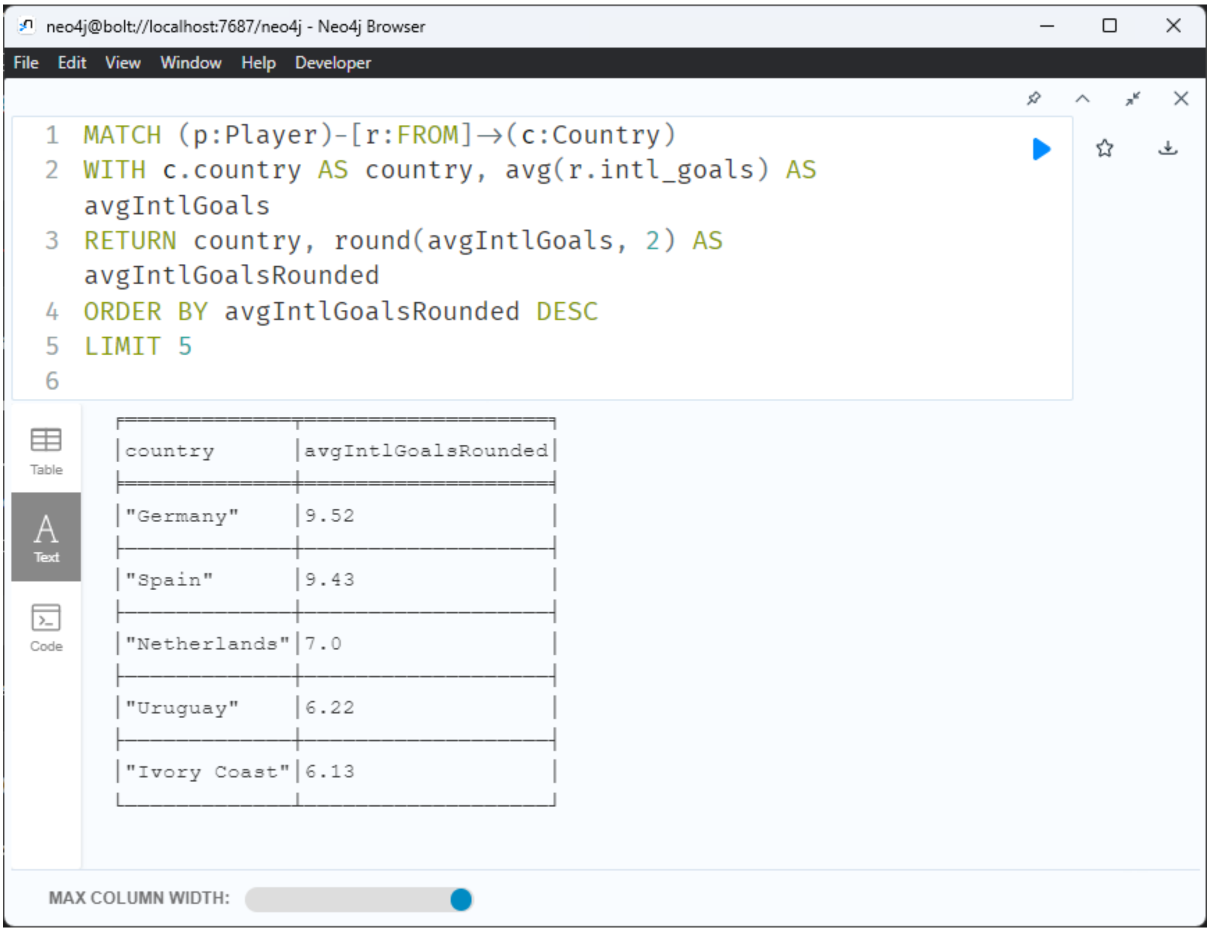
4.13 Find the path with a length of 2 or 3 between <"Liverpool FC" and "Arsenal FC">.

```
MATCH p=(c1:Club { club: "Liverpool FC" })-[*2..3]-(c2:Club {club: "Arsenal FC"})
RETURN p
```



4.14 Find the top 5 countries with players who have the highest average number of international goals. Return the countries and their average international goals in descending order.

```
MATCH (p:Player)-[r:FROM]->(c:Country)
WITH c.country AS country, avg(r.intl_goals) AS avgIntlGoals
RETURN country, round(avgIntlGoals, 2) AS avgIntlGoalsRounded
ORDER BY avgIntlGoalsRounded DESC
LIMIT 5
```

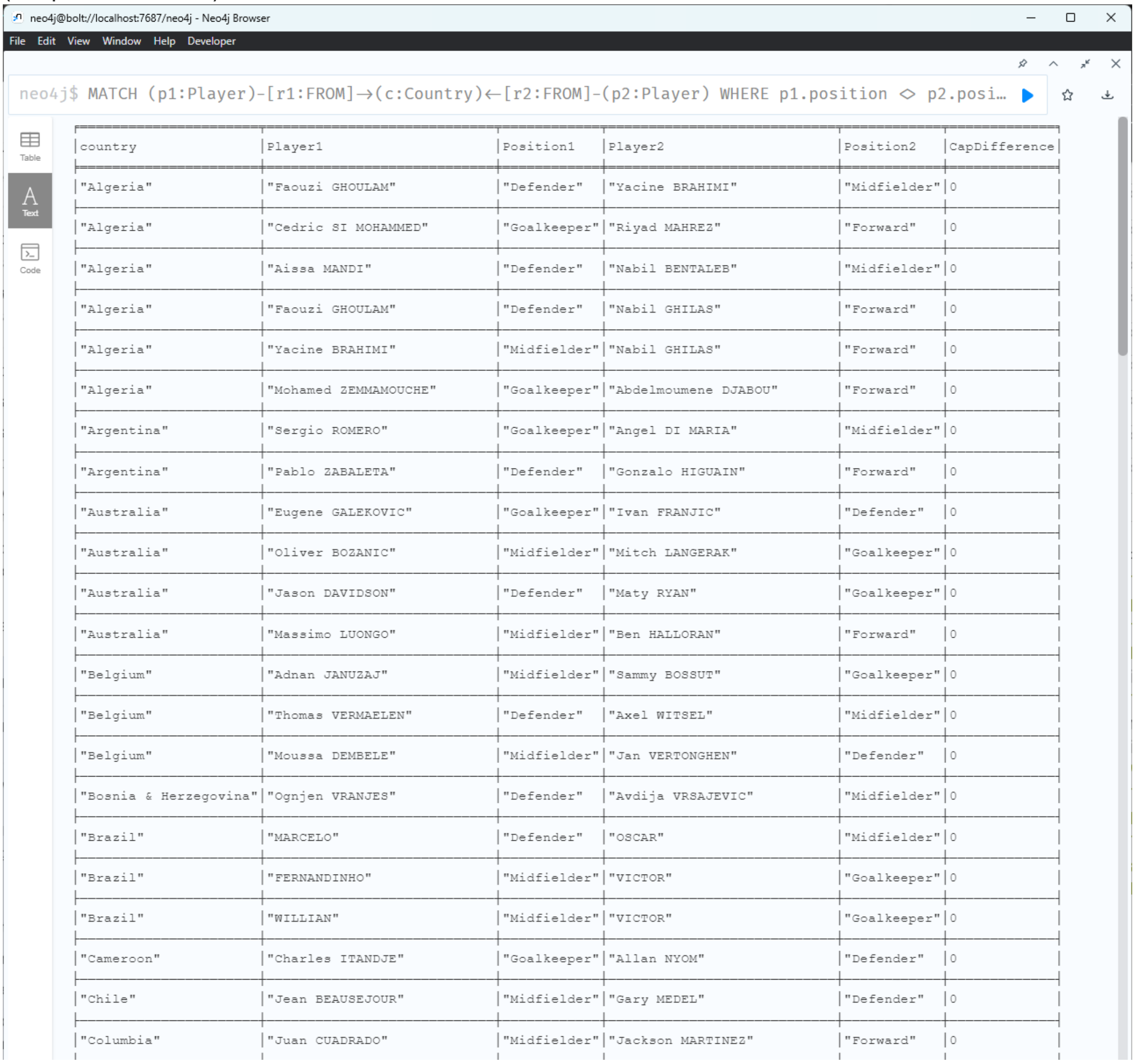


country	avgIntlGoalsRounded
"Germany"	9.52
"Spain"	9.43
"Netherlands"	7.0
"Uruguay"	6.22
"Ivory Coast"	6.13

4.15 Identify pairs of players from the same national team who play in different positions but have the closest number of caps. Return these pairs along with their positions and the difference in caps.

```
MATCH (p1:Player)-[r1:FROM]->(c:Country)<-[r2:FROM]-(p2:Player)
WHERE p1.position <> p2.position AND p1.player_id < p2.player_id
WITH c.country AS country, p1, p2, ABS(r1.caps - r2.caps) AS capDifference
 ORDER BY country, capDifference
// Find the minimum cap difference for each country
WITH country, collect({ Player1: p1, Player2: p2, capDifference: capDifference }) AS playerPairs, min(cap
Difference) AS minCapDiff
// Filter to include only pairs with the minimum cap difference
UNWIND playerPairs AS pair
WITH country, pair
WHERE pair.capDifference = minCapDiff
RETURN country, pair.Player1.name AS Player1, pair.Player1.position AS Position1, pair.Player2.name A
S Player2, pair.Player2.position AS Position2, pair.capDifference AS CapDifference
 ORDER BY country
```

(output truncated)

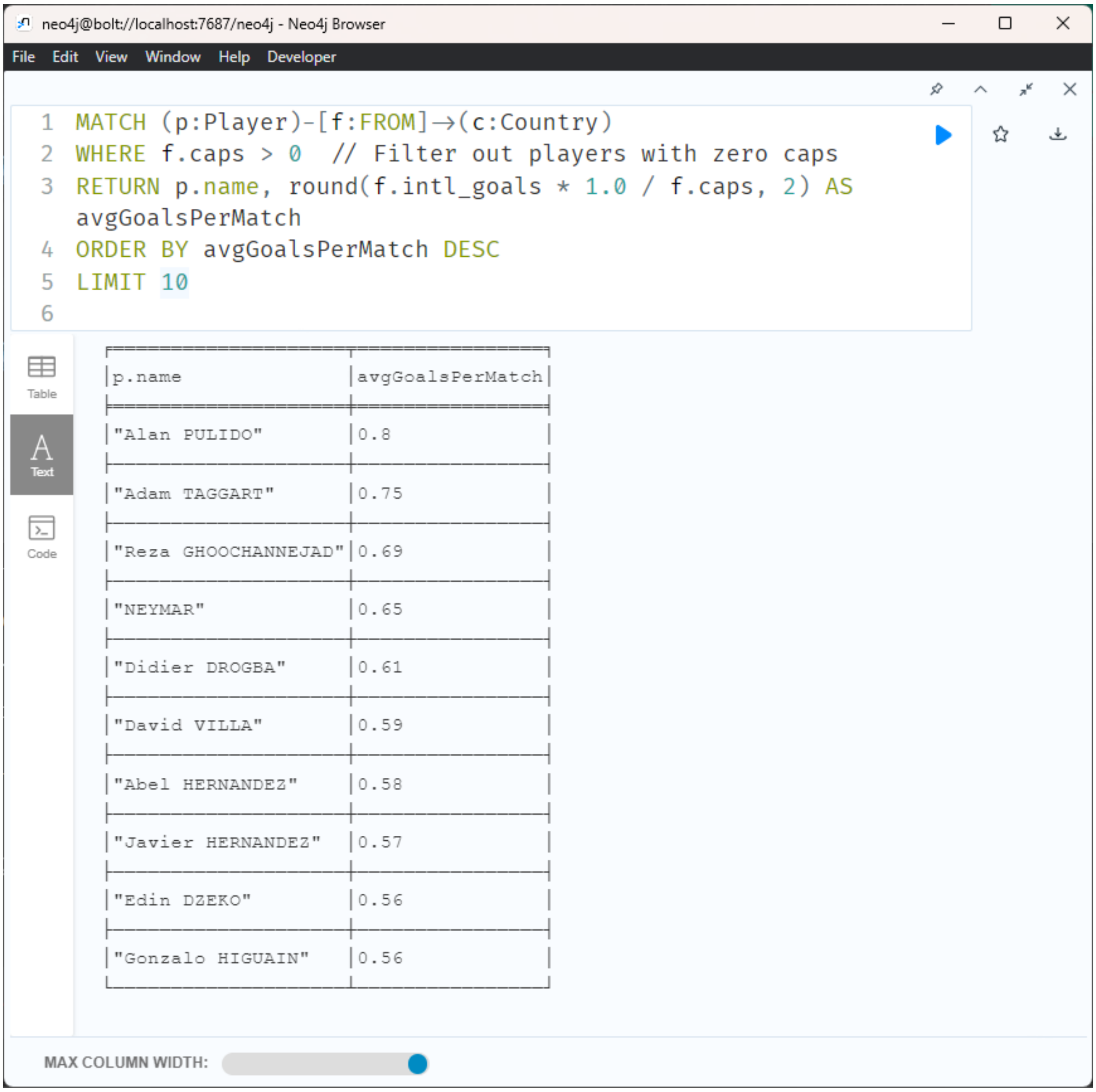


country	Player1	Position1	Player2	Position2	CapDifference
"Algeria"	"Faouzi GHOULAM"	"Defender"	"Yacine BRAHIMI"	"Midfielder"	0
"Algeria"	"Cedric SI MOHAMMED"	"Goalkeeper"	"Riyad MAHREZ"	"Forward"	0
"Algeria"	"Aissa MANDI"	"Defender"	"Nabil BENTALEB"	"Midfielder"	0
"Algeria"	"Faouzi GHOULAM"	"Defender"	"Nabil GHILAS"	"Forward"	0
"Algeria"	"Yacine BRAHIMI"	"Midfielder"	"Nabil GHILAS"	"Forward"	0
"Algeria"	"Mohamed ZEMMAMOUCHE"	"Goalkeeper"	"Abdelmoumene DJABOU"	"Forward"	0
"Argentina"	"Sergio ROMERO"	"Goalkeeper"	"Angel DI MARIA"	"Midfielder"	0
"Argentina"	"Pablo ZABALETA"	"Defender"	"Gonzalo HIGUAIN"	"Forward"	0
"Australia"	"Eugene GALEKOVIC"	"Goalkeeper"	"Ivan FRANJIC"	"Defender"	0
"Australia"	"Oliver BOZANIC"	"Midfielder"	"Mitch LANGERAK"	"Goalkeeper"	0
"Australia"	"Jason DAVIDSON"	"Defender"	"Maty RYAN"	"Goalkeeper"	0
"Australia"	"Massimo LUONGO"	"Midfielder"	"Ben HALLORAN"	"Forward"	0
"Belgium"	"Adnan JANUZAJ"	"Midfielder"	"Sammy BOSSUT"	"Goalkeeper"	0
"Belgium"	"Thomas VERMAELEN"	"Defender"	"Axel WITSEL"	"Midfielder"	0
"Belgium"	"Moussa DEMBELE"	"Midfielder"	"Jan VERTONGHEN"	"Defender"	0
"Bosnia & Herzegovina"	"Ognjen VRANJES"	"Defender"	"Avdija VRSAJEVIC"	"Midfielder"	0
"Brazil"	"MARCELO"	"Defender"	"OSCAR"	"Midfielder"	0
"Brazil"	"FERNANDINHO"	"Midfielder"	"VICTOR"	"Goalkeeper"	0
"Brazil"	"WILLIAN"	"Midfielder"	"VICTOR"	"Goalkeeper"	0
"Cameroon"	"Charles ITANDJE"	"Goalkeeper"	"Allan NYOM"	"Defender"	0
"Chile"	"Jean BEAUSEJOUR"	"Midfielder"	"Gary MEDEL"	"Defender"	0
"Columbia"	"Juan CUADRADO"	"Midfielder"	"Jackson MARTINEZ"	"Forward"	0

4.16 Write Cypher queries for at least two other meaningful queries

4.16.1 Find the top 10 players with the highest average international goals per match (caps).

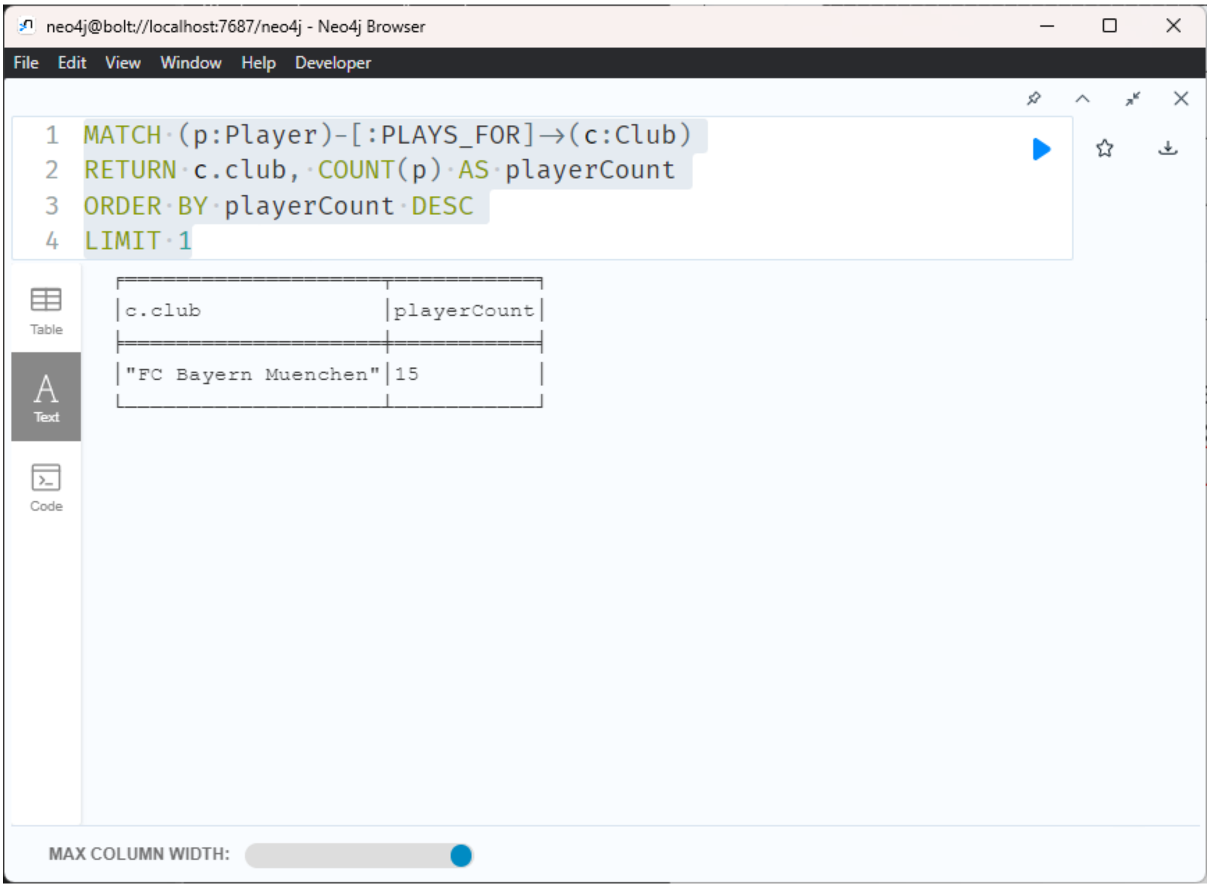
```
MATCH (p:Player)-[f:FROM]->(c:Country)
WHERE f.caps > 0 // Filter out players with zero caps
RETURN p.name, round(f.intl_goals * 1.0 / f.caps, 2) AS avgGoalsPerMatch
 ORDER BY avgGoalsPerMatch DESC
LIMIT 10
```



p.name	avgGoalsPerMatch
"Alan PULIDO"	0.8
"Adam TAGGART"	0.75
"Reza GHOOCHANNEJAD"	0.69
"NEYMAR"	0.65
"Didier DROGBA"	0.61
"David VILLA"	0.59
"Abel HERNANDEZ"	0.58
"Javier HERNANDEZ"	0.57
"Edin DZEKO"	0.56
"Gonzalo HIGUAIN"	0.56

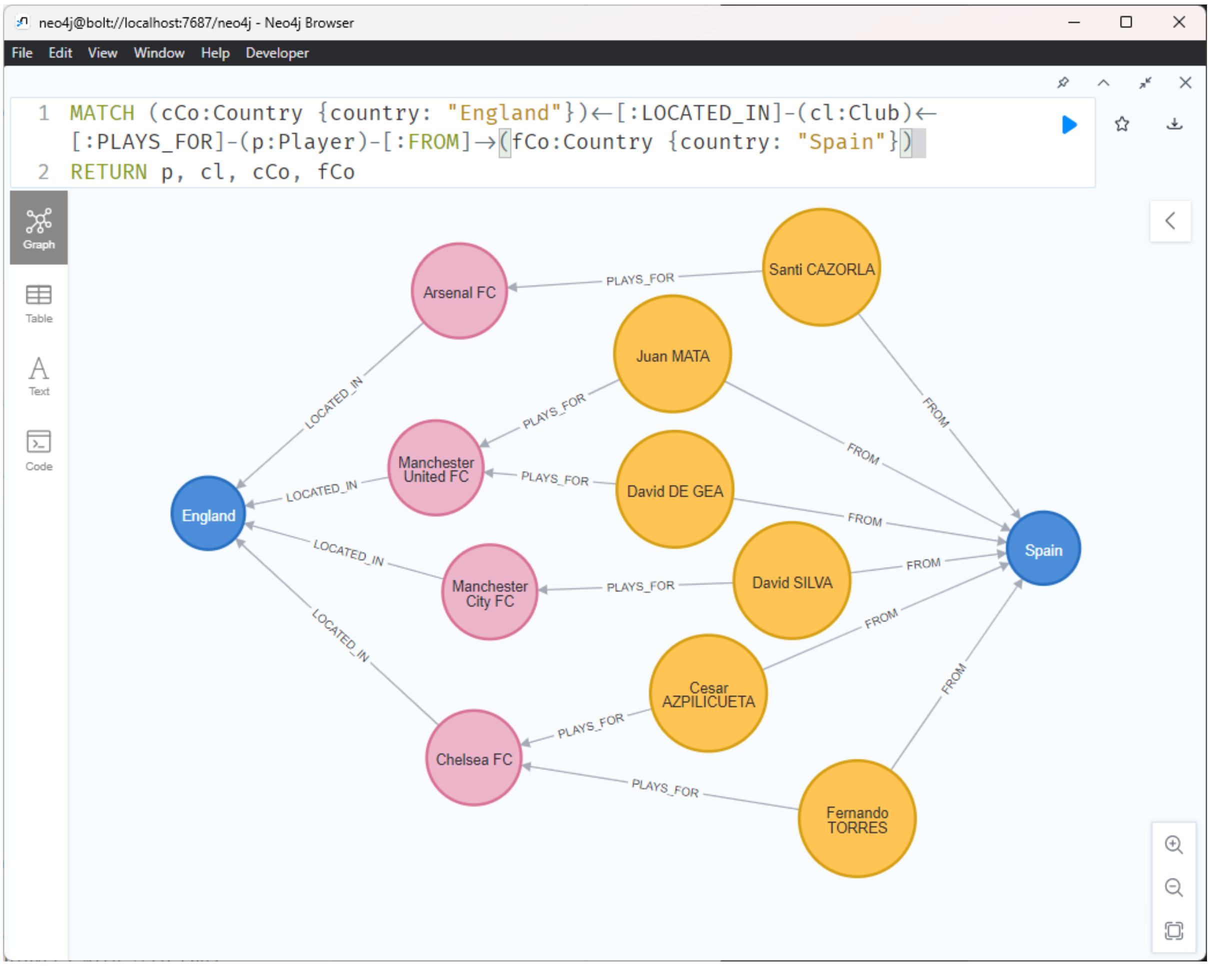
4.16.2 Which club has the most players participating in the 2014 FIFA World Cup?

```
MATCH (p:Player)-[:PLAYS_FOR]->(c:Club)
RETURN c.club, COUNT(p) AS playerCount
ORDER BY playerCount DESC
LIMIT 1
```



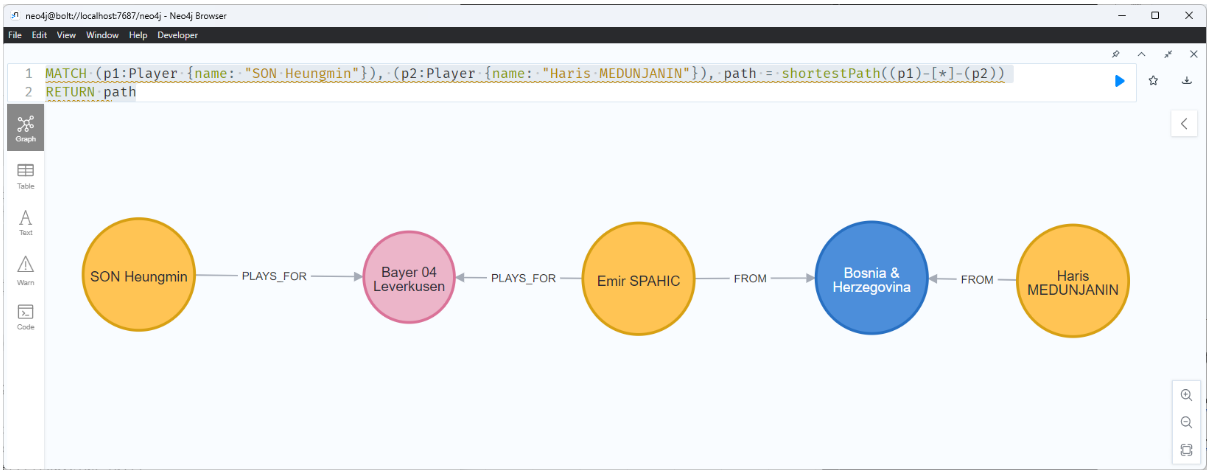
4.16.3 Who is from Spain, but plays for an English club?

```
MATCH (cCo:Country { country: "England" })<-[:LOCATED_IN]-(cl:Club)<-[:PLAYS_FOR]-(p:Player)-
[:FROM]->(fCo:Country {country: "Spain"})
RETURN p, cl, cCo, fCo
```



4.16.4 What is the shortest path between players, "SON Heungmin" and "Haris MEDUNJANIN"?

```
MATCH (p1:Player { name: "SON Heungmin" }), (p2:Player {name: "Haris MEDUNJANIN"}), path = shortest
Path((p1)-[*]-(p2))
RETURN path
```



5. GRAPH DATABASES

5.1 Capabilities compared to relational databases

Relational databases and graph databases serve different purposes and perform well in different areas.

Relational databases store data in a highly structured tabular format. Each row of a table is uniquely identified by a primary key, which serves as a reference for creating relationships with other tables via foreign keys. When executing queries, these databases perform JOIN operations by matching primary keys with foreign keys. These joins can be computationally intensive and require significant memory resources [1].

In contrast, graph databases are a type of NoSQL database in which data is represented as a collection of nodes and the relationships that connect them. Both nodes and relationships can have properties, and relationships can also have a direction. This structure allows for modelling of many real-world scenarios [2].

5.1.1 Use cases

Relational databases excel at handling transactional data, whereas graph databases are ideal for relationship-heavy use cases. This capability allows graph databases to be used for analysing complex datasets to uncover unexpected connections between data points. They can be used in a wide range of applications, including but not limited to [2]:

1. ***Social networks***: to model and query relationships such as friends, followers, and interactions.
2. ***Recommendation systems***: to generate real-time personalised recommendations from relationships and user behaviour.

3. ***Fraud detection***: to identify fraudulent patterns and connections within transactional data through relationship analysis.

5.1.2 Data modelling

Relational databases require a predefined schema, making it difficult to adapt to changes in the data model. Any modification to the schema, such as adding new columns, typically requires significant planning and can be disruptive to operation [3].

Graph databases, on the other hand, offer a dynamic and flexible schema. New types of nodes and relationships can be added without the need for extensive schema redesigns. This flexibility allows graph databases to adapt quickly to evolving data models, making them ideal for situations where the structure of the data is not completely known upfront or is subject to frequent changes. Furthermore, graph databases avoid NULL values, which can complicate data integrity in relational databases [2].

5.1.3 Recursive and complex queries

When handling recursive or complex queries, relational databases must perform many join operations which impacts performance and interpretability of the syntax of queries.

Meanwhile, graph databases can execute these types of queries quickly because they do not require join operations. Each node stores a list of relationship records that represent its connections to other nodes. These relationships are organised by type and direction and may include additional attributes. This structure allows queries to directly access connected node, reducing the time required for join-heavy queries from minutes to milliseconds [1]. Furthermore, the syntax for such queries is also much easier to construct and understand.

5.2 Graph data science applications

Graph data science represents an innovative approach to uncovering connections and relationships within data. By combining advanced analytics and machine learning, quick and accurate answers to complex business problems can be delivered to business leaders to make informed decisions. Several areas that graph data science can be applied are discussed below.

5.2.1 Finance

It is necessary for businesses to be proactive in combating fraud, whether it originates externally or from within an organisation. By analysing transaction data to identify anomalies, businesses can detect suspicious activities early. Community detection algorithms can be employed to spot patterns of suspicious users and transactions, significantly enhancing fraud detection capabilities. This proactive approach can save companies millions of dollars by preventing fraudulent activities before they cause substantial financial damage [4].

5.2.2 Supply chain

Graph data science can significantly enhance supply chain management by optimising transport routes. Supply chains inherently form graph structures with interconnected nodes, including suppliers, customers, and distribution hubs. By applying pathfinding algorithms, businesses can identify the shortest and most efficient routes through the supply chain network. Additionally, predicting disruptions and rerouting in real-time can further reduce delivery times, increase customer satisfaction, and result in substantial cost and time savings. This approach also contributes to sustainability efforts by reducing carbon emissions, which is particularly significant, especially as the transportation industry is responsible for over 25% of all greenhouse gas emissions [4].

5.2.3 Life sciences

Graph data science accelerates the discovery of new drugs by uncovering hidden relationships in biological and pharmacodynamic data. Drug companies can use graph databases to create a scalable biological knowledge system that consolidates the information in different datasets. Link prediction can then be used to discover connections between diseases and drug molecules, helping researchers to identify novel therapeutic targets and biomarkers. Typically, the process of discovering and refining a drug for development takes 3-5 years. By leveraging graph data science, this timeline can be significantly shortened so that effective drugs can be brought to market faster allowing potentially life-saving treatments to reach patients more quickly [4].

6. REFERENCES

[1] Neo4j, “Transition from relational to graph database,” Neo4j, 2024. [Online]. Available: https://neo4j.com/docs/getting-started/appendix/graphdb-concepts/graphdb-vs-rdbms/#relational-vs-graph. [Accessed 25 May 2024].

[2] I. Robinson, J. Webber and E. Eifrem, Graph Databases, California: O'Reilly, 2015.

[3] J. Delplanque, A. Etien, N. Anquetil and O. Auverlot, “Relational Database Schema Evolution: An Industrial Case Study,” in *EEE International Conference on Software Maintenance and Evolution (ICSME)*, Madrid, Spain, 2018.

[4] Neo4j, “Graph Data Science Use Case Selection Guide,” [Online]. Available: https://neo4j.com/whitepapers/graph-data-science-use-case-selection-guide. [Accessed 24 May 2024].