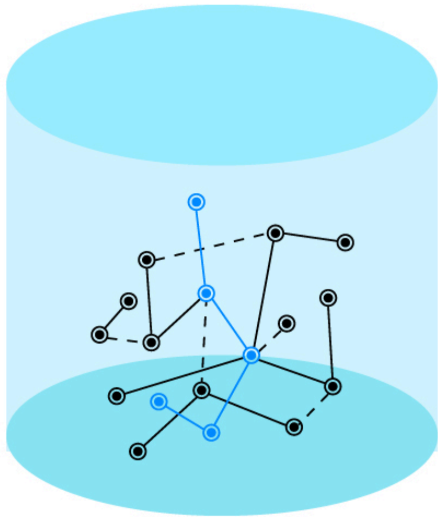
CITS5504 Data Warehouse
Project 2 Report



Graph Database Design and Cypher Query

Group 71

Thi Ngoc Han Hoang (23432313)
Yunhui Zhang(23839202)

1.Datasets
    1.1 Dataset Overview
    1.2 Assumptions
2. Graph Database Design
    2.1 Design Rationale
        2.11. Natural Data Modeling
        2.12 Node and Relationship Design
        2.13 Data Type Design for Node ID and Property
        2.14. Performance and Efficiency
        2.15. Neo4j Visualization
    2.2 Design Property Graph Via Arrows App
3. ETL
    3.1 CSV Files for Populating Graph Database
    3.2 Import CSVs Into Neo4j
    3.3 Created nodes and properties
4.Queries
    4.1 Required Queries
        1. What is the jersy number of the player with <a specific player id>?
        2. Which clubs are based in <a specific country>?
        3. Which club does <a specific player> play for?
        4. How old is <a specific player>?
        5. In which country is the club that <a specific player> plays for?
        6. Find a club that has players from <a specific country>.
        7. Find all players play at <a specific club>, returning in ascending orders of age.
        8. Find all <a specific position> players in the national team of <a specific country>, returning in descending order of caps.
        9. Find all players born in <a specific year> and in national team of <a specific country>, returning in descending order of caps.
        10. Find the players that belongs to the same club in national team of <a specific country>, returning in descending order of international goals.
        11. Count how many players are born in <a specific year>.
        12. Which age has the highest participation in the 2014 FIFA World Cup?
        13. Find the path with a length of 2 or 3 between <two specific clubs>.

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.
15. (CITS5504 only) 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.
4.2 Self-designed queries
Query 16.1 Find the player with the highest number of caps in Australia
Query 16.2 List all players who play for clubs in their home country.
Query 16.3: Find the average height of players in each national team, returning in descending order of average height
5. Discussion
5.1 Capability of Graph Databases Compared to Relational Databases
5.2 How Graph Data Science Applied to Practical Application
6.References
7. Software and Tool

Disclaimer: The project adopted recommendations from using ChatGPT 3.5 in help of identifying issues to validating Python syntax and complicated Neo4j queries, and checking grammars and spellings for the report (OpenAI, 2022). Moreover, we have no performance issue while running all the Cypher queries, thus we did not apply data filtering.

1.Datasets

1.1 Dataset Overview

The dataset contains information about 736 players who participated in the 2014 FIFA World Cup. The columns include:

- Player id: Unique identifier for each player.
- Player: Name of the player.
- Position: Position played by the player (e.g., Forward, Midfielder, Defender, Goalkeeper).
- Number: Jersey number of the player.
- Club: Club the player belongs to.
- Club (country): Country where the player's club is located.
- D.O.B: Date of birth of the player in DD.MM.YYYY format.
- Age: Age of the player during the 2014 FIFA World Cup.
- Height (cm): Height of the player in centimeters.
- Country: National team country of the player.
- Caps: Number of times the player has represented their national team in international matches before the 2014 FIFA World Cup.
- International goals: Number of goals scored by the player for the national team before the 2014 FIFA World Cup.
- Plays in home country?: Indicates whether the player plays for a club in their home country (TRUE/FALSE).

1.2 Assumptions

In the 2014 FIFA World Cup dataset, each player is unique and associated with only one club based on the below observation excel figure. Consequently, one player only plays for one club and finding a path of length 2 between two clubs via a player (i.e., Club ← Player → Club) is not possible.

To identify paths of at least length 2 between clubs, it needs to incorporate additional entities and relationships into graph database design. Specifically:

1. Club Country: Represent the country each club is based in.
2. BASES_IN Relationship: Establish a relationship between each club and its respective country.

This allows finding paths like:

- Club → Club Country ←Club: A path where clubs are connected through club located countries.

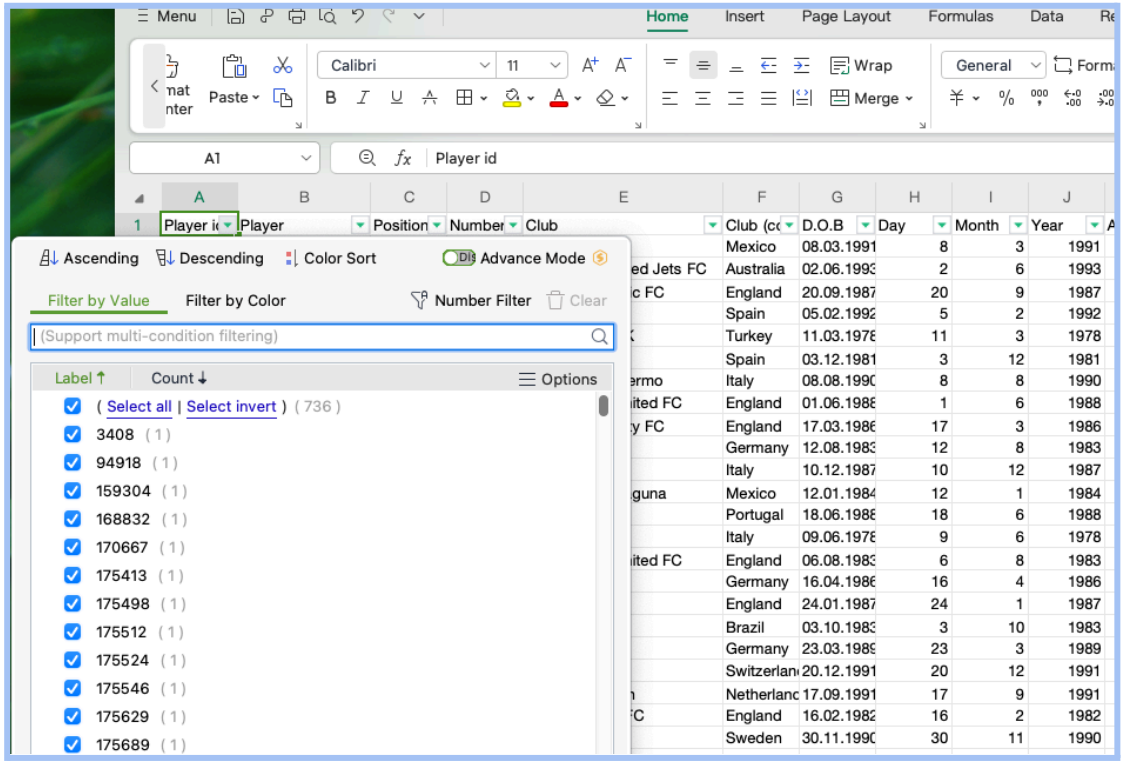


Figure 1.2. Each player only plays for one club.

2. Graph Database Design

2.1 Design Rationale

This project will adopt graph database design and its design rationale are as below:

2.11. Natural Data Modeling

- Entities and Relationships: Graph databases use nodes to represent entities and edges to represent relationships in FIFA2014-all players dataset, mirroring FIFA 2014 players' relationships and entities more naturally than relational databases.
- Flexibility: Graph databases support schema-less data structures, allowing for easier and more flexible updates to the data model as requirements evolve in the future.

2.12 Node and Relationship Design

The reason why the project is designed with below nodes and relationships, as it can satisfy all the potential queries with simple relationships, and especially "find the length of 2 between two clubs" via common clubCountry node, instead of via player node taking at least 4 length between clubs considering one player can only play for one club.

Node	Properties
Player	playerID, player, position, number, DOB (day, month, year), age, height, caps, internationalGoals, playsInHomeCountry
Club	clubID, club
Country	countryID, country
ClubCountry	clubCountryID, club_country

Relationship	From Node	To Node
PLAYS_FOR	Player	Club
REPRESENTS	Player	Country
BASES_IN	Club	ClubCountry

Figure 2.12. Nodes and Relationship Design Overview

2.13 Data Type Design for Node ID and Property

The project will ***utilize the original dataset's player ID*** since it is unique and sufficient for the project's scope, despite the original generation algorithm being unknown. Additionally, Neo4j can automatically generate ***surrogate keys*** for nodes during creation. As the project only involves ***one dataset,*** there is no need to create custom IDs for easier integration with other data sources. Consequently, ClubID, CountryID, and ClubCountryID are self-created, while the player ID retains its original format from the dataset.

Additionally, we will design the Node ID as an integer rather than a string. ***Using integers for Node IDs*** has several ***advantages***:

1. Memory Efficiency: Integers consume less memory compared to strings. This is particularly important in large datasets, as it reduces the overall memory footprint of the database.
2. Performance: Integer comparisons and lookups are faster than string comparisons. This improves the performance of queries, especially those involving large numbers of nodes and relationships.
3. Indexing: Indexing on integer fields is more efficient and faster than indexing on string fields. This enhances the speed of retrieval operations and ensures quicker access to nodes.
4. Storage Optimization: Integer IDs take up less storage space compared to string IDs. This optimization is beneficial for both in-memory operations and disk storage, contributing to overall database efficiency.
5. Consistency: Using integers ensures consistency in the format of Node IDs, avoiding issues related to varying string formats (e.g., case sensitivity, leading/trailing spaces).

By opting for integer Node IDs, we can achieve better performance, ***lower memory consumption***, and more efficient storage, ultimately leading to a more robust and scalable graph database design.

Property	age, caps, height (cm) and International goals	Integer
Property	playsInHomeCountry	Boolean(true/false)
Property	player, position, number, DOB,day, month, year	String
Node ID	playerID,clubID,countryID,clubCountryID	Integer

Figure 2.13: Data type of properties and nodes

2.14. Performance and Efficiency

Graph database are optimized for traversing relationships, making it efficient to answer complex queries such as:

- Finding the country where the club of a specific player is based.
- Listing all players at a specific club in ascending order of age.
- Listing all players in a specific position for a national team, ordered by caps.
- Finding players born in a specific year and their national team, ordered by caps.
- Identifying players from the same club in a national team, ordered by international goals.
- Find the path with a length of 2 or 3 between <two specific clubs>
- Counting the number of players born in a specific year.
- Identifying the age with the highest participation in the 2014 FIFA World Cup.
- Identifying top countries with players having the highest average international goals.
- Identify pairs of players from the same national team who play in different positions but have the closest number of caps.

2.15. Neo4j Visualization

The project will use Neo4j for Cypher queries. Neo4j graph databases integrate well with visualization tools, enabling intuitive exploration of the data, such as visualizing the network of player movements between clubs and national teams.

2.2 Design Property Graph Via Arrows App

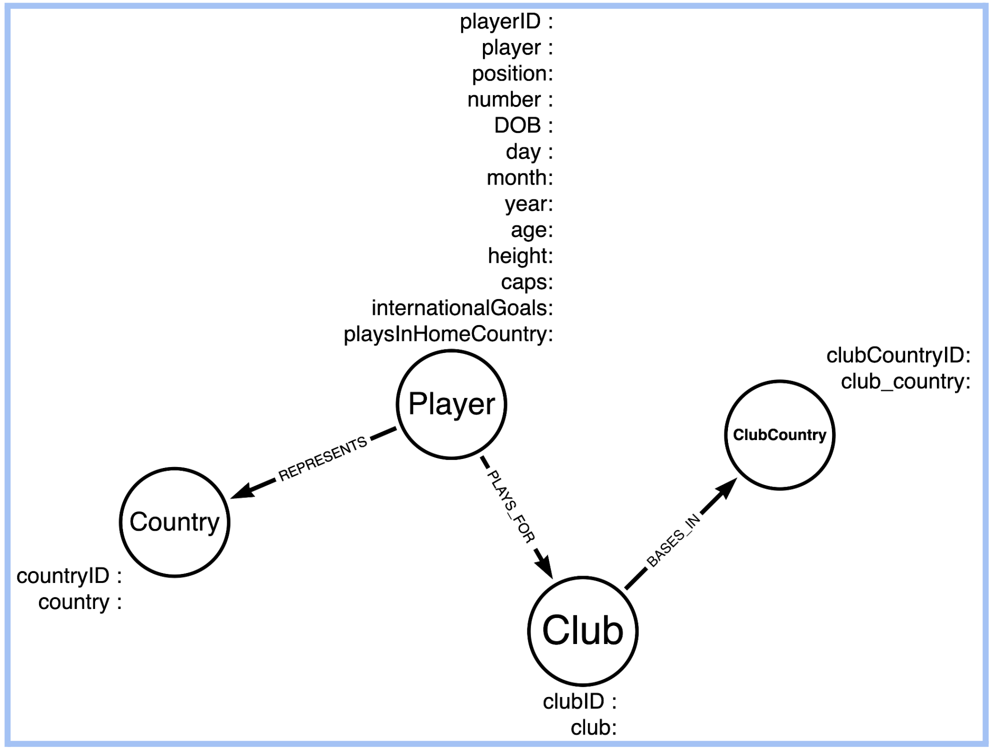


Figure 2.2. Property Graph

<u>Explanation of Nodes and Relationships</u>

Nodes:

1. Player: Represents each player in the dataset. Key properties include player ID, name, position, jersey number, date of birth, day,month, year, age, height, number of international caps, number of international goals, and whether the player plays in their home country.
2. Club: Represents the football club each player belongs to. Key properties include club ID and club.
3. Country: Represents the national team country of the player. Key properties include country ID and country name.
4. ClubCountry: Represents the country where a player's club is located. Key properties include club country ID and country.

Relationships:

1. PLAYS_FOR: Connects a Player node to a Club node, indicating the club a player plays for.
2. REPRESENTS: Connects a Player node to a Country node, indicating the national team the player represents.
3. BASES_IN: Connects a Club node to a ClubCountry node, indicating the country where the club is based.

3. ETL

We use the Python 3.8.8 version to implement the ETL steps. We first import the original raw data "FIFA2024 - all players.csv", and we conduct a test to check if there is any missing value or null entries in the data. Identifying and handling missing values can prevent potential errors or inconsistencies before we populate all the processed files to Neo4j.

3.1 Data Processing

1. Loading dataset

```
[ ] #read the csv file
    fifa_player = pd.read_csv("FIFA2014 - all players.csv")

    # Convert boolean values in 'Plays in home country?' column to strings and uppercase
    fifa_player['Plays in home country?'] = fifa_player['Plays in home country?'].astype(str).str.upper()

    fifa_player.head()
```

	Player id	Player	Position	Number	Club	Club (country)	D.O.B	Age	Height (cm)	Country	Caps	International goals	Plays in home country?
0	336722	Alan PULIDO	Forward	11	Tigres UANL	Mexico	08.03.1991	23	176	Mexico	5	4	TRUE
1	368902	Adam TAGGART	Forward	9	Newcastle United Jets FC	Australia	02.06.1993	21	172	Australia	4	3	TRUE
2	362641	Reza GHOOCHANNEJAD	Forward	16	Charlton Athletic FC	England	20.09.1987	26	181	Iran	13	9	FALSE
3	314197	NEYMAR	Forward	10	FC Barcelona	Spain	05.02.1992	22	175	Brazil	48	31	FALSE
4	212306	Didier DROGBA	Forward	11	Galatasaray SK	Turkey	11.03.1978	36	180	Ivory Coast	100	61	FALSE

2. Check missing values and missing values' count

```
# Check for missing values
missing_values = fifa_player.isnull().sum()

# Display the missing values count for each column
print(missing_values)
```

```
Player id                 0
Player                    0
Position                  0
Number                    0
Club                      0
Club (country)            0
D.O.B                     0
Age                       0
Height (cm)               0
Country                   0
Caps                      0
International goals       0
Plays in home country?    0
dtype: int64
```

3.2 Create CSV Tables for Populating Graph Database

Based on the graph database, we need two types of CSV files which are node tables and relationship tables to connect the nodes. The node CSV files include *club.csv, club_country.csv, country.csv,* and *player.csv.* The relationship tables include rel_plays_for.csv, *rel_represents.csv* and *rel_based_in.csv*

First, we start with the node tables:

- ***club.csv***: We create a def club () function to extract the club from fifa_player data frame which includes Club with ID for each unique club. We have dropped duplications for the file in the function.

```python
def club(dataframe):
    """
    Create Club.csv file from the dataframe
    Columns: Club ID, Club

    Parameters
    ----------
    dataframe : dataframe
        The dataframe that contains the Fifa player data

    Returns
    -------
    club_table : dataframe
        The club table
    """

    club_table = dataframe[['Club']].copy()

    club_table = club_table.dropna().drop_duplicates().reset_index(drop=True)

    club_table.insert(0, 'Club ID', range(1, 1 + len(club_table)))

    club_table.to_csv('club.csv')

    return club_table
```

club.csv

```
,Club ID,Club
0,1,Tigres UANL
1,2,Newcastle United Jets FC
2,3,Charlton Athletic FC
3,4,FC Barcelona
4,5,Galatasaray SK
5,6,Atletico Madrid
6,7,US Citta di Palermo
7,8,Manchester United FC
8,9,Manchester City FC
9,10,FC Schalke 04
10,11,SSC Napoli
11,12,Club Santos Laguna
12,13,Sporting CP
13,14,SS Lazio
14,15,FSV Mainz 05
15,16,Liverpool FC
16,17,Fluminense FC
17,18,FC Basel
18,19,SC Heerenveen
19,20,Southampton FC
```

❖ ***club_country.csv***: This file will contain information about the country of the club. We create the def clubcountry() function to extract the Club (country) column from the fifa_player data frame with the ID for each unique value. We have removed duplications for the file in the function.

```python
def clubcountry(dataframe):
    """
    Create ClubCountry.csv file from the dataframe
    Columns: Club Country ID, Club (country)

    Parameters
    ----------
    dataframe : dataframe
        The dataframe that contains the Fifa player data

    Returns
    -------
    clubcountry_table : dataframe
        The clubcountry table
    """

    clubcountry_table = dataframe[['Club (country)']].copy()

    clubcountry_table = clubcountry_table.dropna().drop_duplicates().reset_index(drop=True)

    clubcountry_table.insert(0, 'Club Country ID', range(1, 1 + len(clubcountry_table)))

    clubcountry_table.to_csv('club_country.csv')

    return clubcountry_table
```

club_country.csv

```
,Club Country ID,Club (country)
0,1,Mexico
1,2,Australia
2,3,England
3,4,Spain
4,5,Turkey
5,6,Italy
6,7,Germany
7,8,Portugal
8,9,Brazil
9,10,Switzerland
10,11,Netherlands
11,12,Sweden
12,13,United Arab Emirates
13,14,Croatia
14,15,USA
15,16,Ukraine
16,17,Honduras
17,18,Japan
```

- ***country.csv***: We create a def country () function to extract the country from the fifa_player data frame which includes Country with ID for each unique country. We have removed duplications for the file in the function.

```python
def country(dataframe):
    """
    Create country.csv file from the dataframe
    Columns: Country

    Parameters
    ----------
    dataframe : dataframe
        The dataframe that contains the Fifa player data

    Returns
    -------
    country_table : dataframe
        The country table
    """

    country_table = dataframe[['Country']].copy()

    country_table = country_table.dropna().drop_duplicates().reset_index(drop=True)

    country_table.insert(0, 'Country ID', range(1, 1 + len(country_table)))

    country_table.to_csv('country.csv')

    return country_table
```

country.csv

country.csv

```
,Country ID,Country
0,1,Mexico
1,2,Australia
2,3,Iran
3,4,Brazil
4,5,Ivory Coast
5,6,Spain
6,7,Uruguay
7,8,Bosnia & Herzegovina
8,9,Netherlands
9,10,Argentina
10,11,Algeria
11,12,Germany
```

- ***player.csv***: In our player dataset, we utilise FIFA player IDs as they provide a unique identifier for each player. To come with this decision, we check the player Id column to ensure they are unique values. These IDs are distinct, therefore, we employ FIFA player IDs to maintain consistency across datasets and systems. We create a def player () function to extract relevant information of the player from fifa_player data frame which includes *Player id, Player , Position, Number, D.O.B','Age, Height (cm), Caps, International goals, Plays in home country?*. We have removed duplications for the file in the function. To make the data more suitable for utilization in Neo4j and to avoid challenges with data types later on, we split the 'Date of Birth' into three separate columns: Day, Month, and Year.

```python
# Check if the 'Player id' column is unique column
playerid_unique = fifa_player['Player id'].is_unique

print(f"Is 'Player id' column unique? {playerid_unique}")
```

```
Is 'Player id' column unique? True
```

```python
def player(dataframe):
    """
    Create Player.csv file from the dataframe
    Columns: Player

    Parameters
    ----------
    dataframe : dataframe
        The dataframe that contains the Fifa player data

    Returns
    -------
    player_table : dataframe
        The player table
    """

    player_table = dataframe[['Player id','Player','Position','Number','D.O.B','Age','Height (cm)','Caps','Internati

    # Split the 'D.O.B' column into separate columns for year, month, and day
    player_table[['Day', 'Month','Year']] = player_table['D.O.B'].str.split('.', expand=True)

    # Convert 'Year', 'Month', and 'Day' columns to integers
    player_table[['Day', 'Month','Year']] = player_table[['Day', 'Month','Year']].astype(int)

    player_table = player_table.dropna().drop_duplicates().reset_index(drop=True)

    player_table.to_csv('player.csv', index=False)

    return player_table
```

```
player.csv
player.csv
Player id,Player,Position,Number,D.O.B,Age,Height (cm),Caps,International goals,Plays in home country?,Day,M
336722,Alan PULIDO,Forward,11,08.03.1991,23,176,5,4,True,8,3,1991
368902,Adam TAGGART,Forward,9,02.06.1993,21,172,4,3,True,2,6,1993
362641,Reza GHOOCHANNEJAD,Forward,16,20.09.1987,26,181,13,9,False,20,9,1987
314197,NEYMAR,Forward,10,05.02.1992,22,175,48,31,False,5,2,1992
212306,Didier DROGBA,Forward,11,11.03.1978,36,180,100,61,False,11,3,1978
229884,David VILLA,Forward,7,03.12.1981,32,175,95,56,True,3,12,1981
305372,Abel HERNANDEZ,Forward,8,08.08.1990,23,186,12,7,False,8,8,1990
228599,Javier HERNANDEZ,Forward,14,01.06.1988,26,175,61,35,False,1,6,1988
300409,Edin DZEKO,Forward,11,17.03.1986,28,192,62,35,False,17,3,1986
184615,Klaas Jan HUNTELAAR,Forward,19,12.08.1983,30,187,61,34,False,12,8,1983
271550,Gonzalo HIGUAIN,Forward,9,10.12.1987,26,184,36,20,False,10,12,1987
227851,Oribe PERALTA,Forward,19,12.01.1984,30,177,27,15,True,12,1,1984
354859,Islam SLIMANI,Forward,13,18.06.1988,26,188,19,10,False,18,6,1988
182206,Miroslav KLOSE,Forward,11,09.06.1978,36,182,131,68,False,9,6,1978
217315,Robin VAN PERSIE,Forward,9,06.08.1983,30,186,84,43,False,6,8,1983
286278,Shinji OKAZAKI,Forward,9,16.04.1986,28,174,75,38,False,16,4,1986
270775,Luis SUAREZ,Forward,9,24.01.1987,27,181,77,39,False,24,1,1987
233952,FRED,Forward,9,03.10.1983,30,186,32,16,True,3,10,1983
312316,Eric CHOUPO MOTING,Forward,13,23.03.1989,25,190,26,13,False,23,3,1989
356411,Fabian SCHAER,Defender,22,20.12.1991,22,186,6,3,True,20,12,1991
338673,Uche NWOFOR,Forward,19,17.09.1991,22,177,6,3,False,17,9,1991
373224,Rickie LAMBERT,Forward,18,16.02.1982,32,188,4,2,True,16,2,1982
379894,Miiko ALBORNOZ,Defender,3,30.11.1990,23,180,2,1,False,30,11,1990
208353,Asamoah GYAN,Forward,3,22.11.1985,28,186,77,38,False,22,11,1985
296994,El Arabi SOUDANI,Forward,15,25.11.1987,26,177,21,10,False,25,11,1987
339508,Chris WONDOLOWSKI,Forward,18,28.01.1983,31,182,19,9,True,28,1,1983
213001,Tim CAHILL,Forward,4,06.12.1979,34,180,68,32,False,6,12,1979
214384,EDUARDO_1,Forward,22,25.02.1983,31,177,62,29,False,25,2,1983
170667,Samuel ETOO,Forward,9,10.03.1981,33,181,117,54,False,10,3,1981
271414,Carlo COSTLY,Forward,13,18.07.1982,31,190,68,31,True,18,7,1982
275096,Yoichiro KAKITANI,Forward,11,03.01.1990,24,177,11,5,True,3,1,1990
```

Next, we move on to the relationship tables:

- ***rel_plays_for.csv**:* This file will contain information about the relationship between players and clubs. The columns are included in this table such as *Player id and Club*, indicating which player plays for which club.

```python
def player_club(dataframe, club_table):
    """
    Create Player_Club.csv file from the dataframe
    Columns: Player id, ClubID

    Parameters
    ----------
    dataframe : dataframe
        The dataframe that contains the player data
    club_table : dataframe
        The dataframe that contains the club data

    Returns
    -------
    player_club : dataframe
        The player_club table
    """

    player_club = dataframe.merge(club_table, how='left')
    player_club = player_club[['Player id', 'Club']]
    player_club.to_csv(
        'rel_plays_for.csv', columns=['Player id', 'Club'], header=True, index=False, sep=',')
    return player_club
```

rel_plays_for.csv

rel_plays_for.csv

```
Player id,Club
336722,Tigres UANL
368902,Newcastle United Jets FC
362641,Charlton Athletic FC
314197,FC Barcelona
212306,Galatasaray SK
229884,Atletico Madrid
305372,US Citta di Palermo
228599,Manchester United FC
300409,Manchester City FC
184615,FC Schalke 04
271550,SSC Napoli
227851,Club Santos Laguna
354859,Sporting CP
182206,SS Lazio
217315,Manchester United FC
286278,FSV Mainz 05
270775,Liverpool FC
233952,Fluminense FC
312316,FSV Mainz 05
356411,FC Basel
338673,SC Heerenveen
373224,Southampton FC
379894,Malmo FF
208353,Al Ain FC
296994,GNK Dinamo Zagreb
339508,San Jose Earthquakes
213001,New York Red Bulls
```

❖ ***rel_represents.csv***: This file will contain information about the relationship between players and countries. The country represents the national team of the player. It could include columns such as *Player id* and *Country*.

```python
def player_country(dataframe, country_table):
    """
    Create Player_Country.csv file from the dataframe
    Columns: Player id, CountryID

    Parameters
    ----------
    dataframe : dataframe
        The dataframe that contains the player data
    country_table : dataframe
        The dataframe that contains the country data

    Returns
    -------
    player_country : dataframe
        The player_country table
    """

    player_country = dataframe.merge(country_table, how='left')
    player_country = player_country[['Player id', 'Country']]
    player_country.to_csv(
        'rel_represents.csv', columns=['Player id', 'Country'], header=True, index=False, sep=',')
    return player_country
```

rel_represents.csv

```
Player id,Country
336722,Mexico
368902,Australia
362641,Iran
314197,Brazil
212306,Ivory Coast
229884,Spain
305372,Uruguay
228599,Mexico
300409,Bosnia & Herzegovina
184615,Netherlands
271550,Argentina
227851,Mexico
354859,Algeria
182206,Germany
217315,Netherlands
286278,Japan
270775,Uruguay
233952,Brazil
312316,Cameroon
356411,Switzerland
338673,Nigeria
373224,England
379894,Chile
208353,Ghana
296994,Algeria
339508,USA
213001,Australia
```

❖ ***rel_based_in.csv*:** This file will contain information about the relationship between clubs and the countries of the clubs they are based in. It could include columns such as ClubID and Club (country), indicating which club is based in which country. We have removed duplications for the file in the function.

```python
def club_clubcountry(dataframe, clubcountry_table, club_table):
    """
    Create Player_Country.csv file from the dataframe
    Columns: Player id, CountryID

    Parameters
    ----------
    dataframe : dataframe
        The dataframe that contains the player data
    country_table : dataframe
        The dataframe that contains the country data

    Returns
    -------
    player_country : dataframe
        The player_country table
    """

    club_clubcountry = dataframe.merge(club_table, how='left')
    club_clubcountry = dataframe.merge(clubcountry_table, how='left')
    club_clubcountry = club_clubcountry[['Club ID', 'Club (country)']]
    club_clubcountry = club_clubcountry.dropna().drop_duplicates().reset_index(drop=True) #remove duplicate rows
    club_clubcountry.to_csv(
        'rel_based_in.csv', columns=['Club ID', 'Club (country)'], header=True, index=False, sep=',')
    return club_clubcountry
```

rel_based_in.csv X

rel_based_in.csv

```
Club ID,Club (country)
1,Mexico
2,Australia
3,England
4,Spain
5,Turkey
6,Spain
7,Italy
8,England
9,England
10,Germany
11,Italy
12,Mexico
13,Portugal
```

4. Create Graph Database and Data Loading

4.1 Create Graph Database and import CSVs

In Neo4j Desktop, we will create a DBMS and start to activate the database. We then put all the CSV files as proceeded to the Neo4j import folder and click 'Open' for the Neo4j browser shown in Figure 3.2a-c as follows.

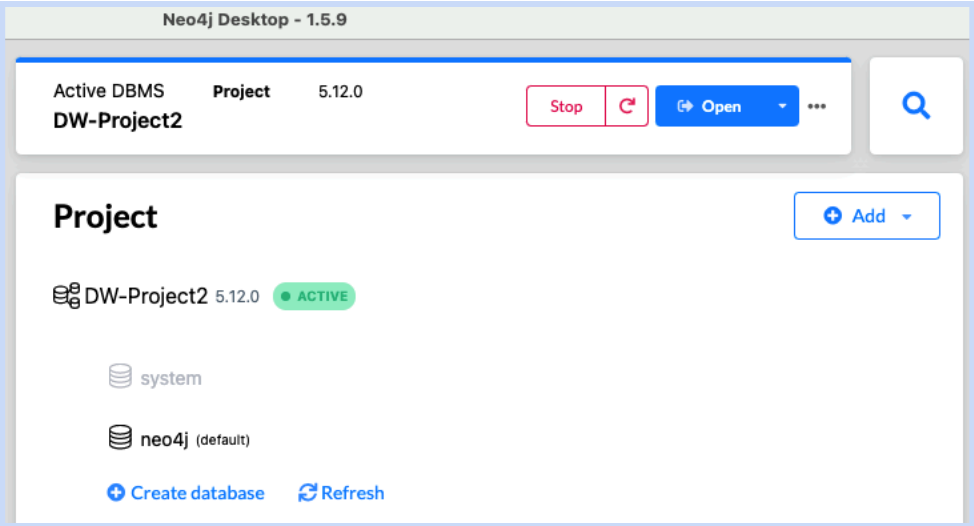


Figure 4.1a. Create Graph database DW-Project2 on Neo4j

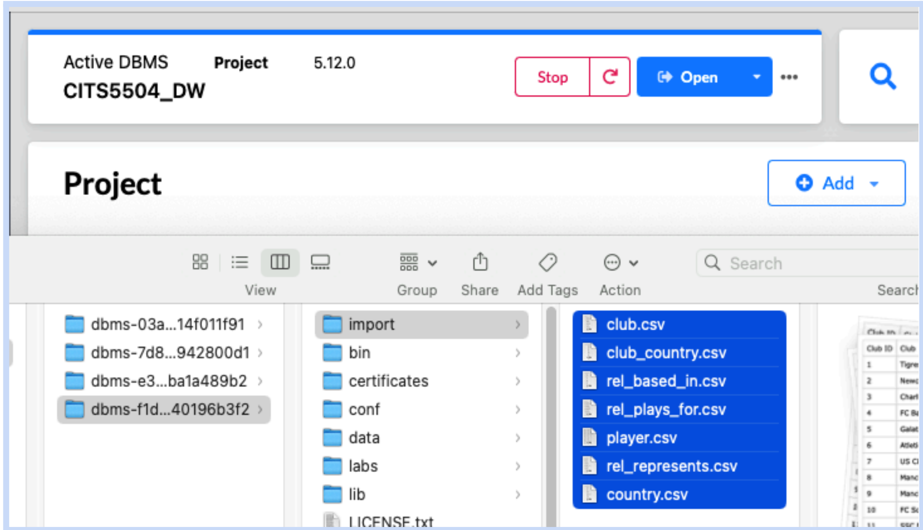


Figure 4.1b. Put CSVs into Neo4j import folder

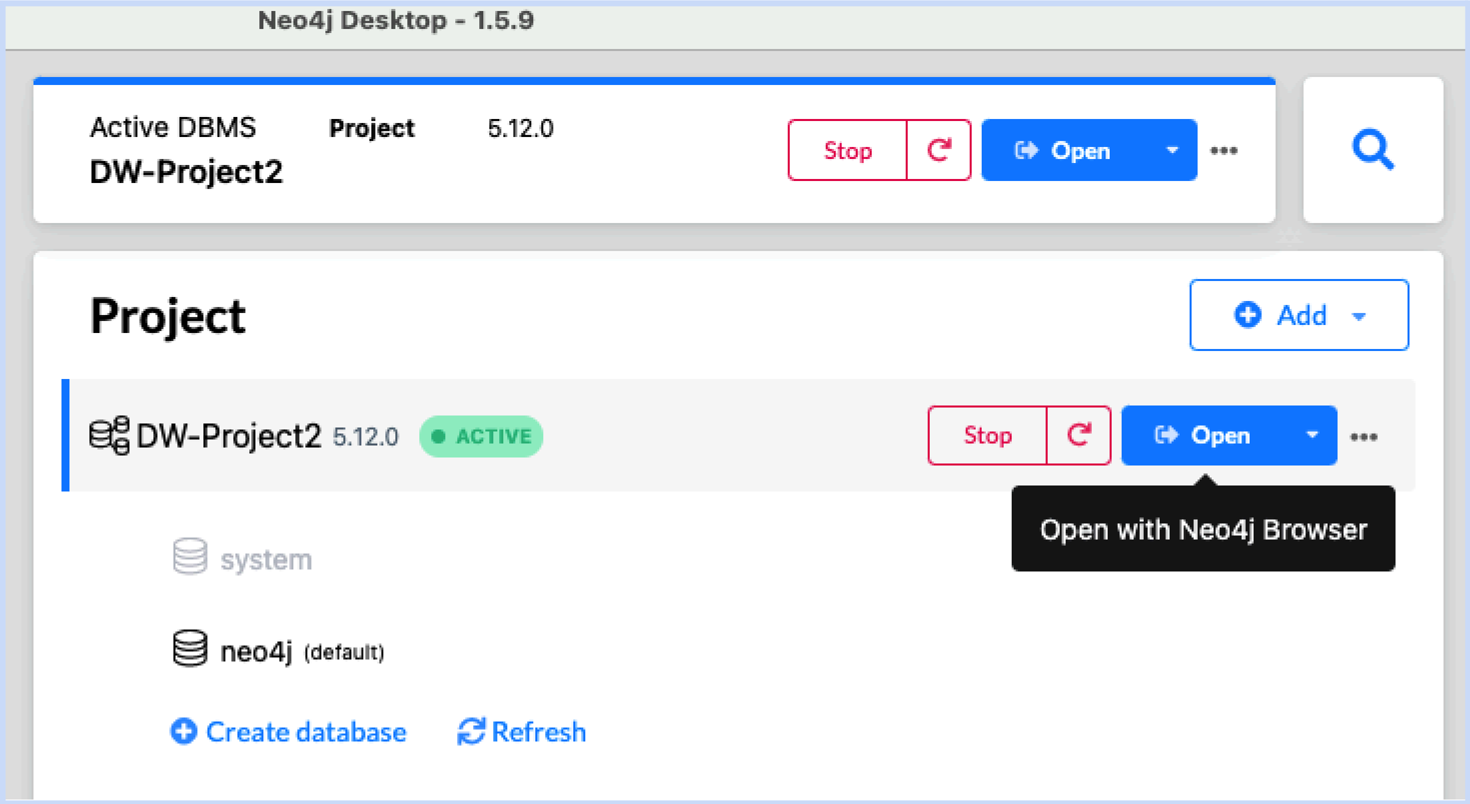


Figure 4.1c. Click open with Neo4j Browser

Refer to Import_data.txt file, it shows all the codes that we use to load node tables and relationship tables.

❖ 4.11 Load Node Tables

```
//LOAD NODE TABLES
//Load player.csv file
LOAD CSV WITH HEADERS FROM 'file:///player.csv' AS row
CREATE (p:Player {
    playerID: TOINTEGER(row.`Player id`),
    player: row.Player,
    position: row.Position,
    number: row.Number,
    DOB: row.`D.O.B`,
    day: row.Day,
    month: row.Month,
    year: row.Year,
```

```
    age: TOINTEGER(row.Age),
    height: TOINTEGER(row.`Height (cm)`),
    caps: TOINTEGER(row.Caps),
    internationalGoals: TOINTEGER(row.`International goals`),
    playsInHomeCountry: CASE row.`Plays in home country?`
                        WHEN 'True' THEN true     // convert to boolean datatype
                        WHEN 'False' THEN false
                        ELSE NULL END
})
```

Added 736 labels, created 736 nodes, set 8832 properties, completed after 1955 ms.

Figure 3.2d. Create Player node

```
//Load club.csv file
LOAD CSV WITH HEADERS FROM 'file:///club.csv' AS row
CREATE (cl:Club {
clubID: TOINTEGER(row.`Club ID`),
club: row.Club
})
```

Added 297 labels, created 297 nodes, set 594 properties, completed after 330 ms.

Figure 3.2e. Creat Club node

```
//Load club_country.csv file
LOAD CSV WITH HEADERS FROM 'file:///club_country.csv' AS row
CREATE (cc:ClubCountry {
    clubCountryID: toInteger(row.`Club Country ID`),
    `club_country`: row.`Club (country)`
})
```

Added 51 labels, created 51 nodes, set 102 properties, completed after 85 ms.

Figure 3.2f. Create ClubCountry node

```
//Load country.csv file
LOAD CSV WITH HEADERS FROM 'file:///country.csv' AS row
CREATE (nation:Country {
    countryID: toInteger(row.`Country ID`),
    country: row.Country
})
```

Added 32 labels, created 32 nodes, set 64 properties, completed after 85 ms.

Figure 3.2g. Create Country node

❖ 4.12 Load Relationship Tables

```
//LOAD RELATIONSHIP TABLES
LOAD CSV WITH HEADERS FROM 'file:///rel_based_in.csv'
AS row
MATCH (cl:Club {clubID: toInteger(row.`Club ID`)})
MATCH (cc:ClubCountry {club_country: row.`Club
(country)`})
CREATE (cl)-[:BASED_IN]->(cc)
```

Created 297 relationships, completed after 915 ms.

Figure created BASED_IN relationship

```
LOAD CSV WITH HEADERS FROM 'file:///rel_represents.csv'
AS row
MATCH (p:Player {playerID: toInteger(row.`Player id`)})
MATCH (nation:Country {country: row.Country})
CREATE (p)-[:REPRESENTS]->(nation)
```

Created 736 relationships, completed after 841 ms.

Figure create relationship 'REPRESENTS'

```
LOAD CSV WITH HEADERS FROM 'file:///rel_plays_for.csv'
AS row
MATCH (p:Player {playerID: toInteger(row.`Player id`)})
MATCH (cl:Club {club: row.`Club`})
CREATE (p)-[:PLAYS_FOR]->(cl)
```

Created 736 relationships, completed after 1070 ms.

Figure created relationship 'PLAYS_FOR'

4.2 Created Nodes and Properties

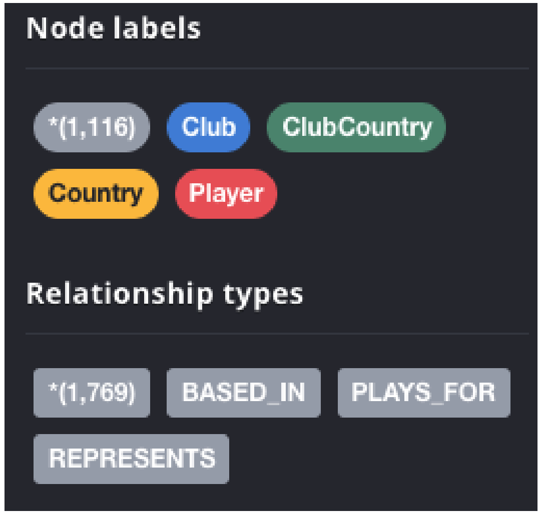


Figure 3.3: created nodes and relationships

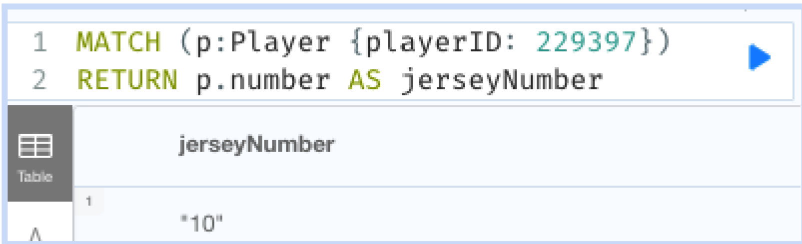
NODE/RELATIONSHIP	COUNT
Club	297
ClubCountry	51
Country	32
Player	736
Node subtotal	1116
BASED_IN	297
PLAYS_FOR	736
REPRESENTS	736
Relationship subtotal	1769

5.Queries

5.1 Required Queries

1. What is the jersy number of the player with <a specific player id>?

   Figure query 1. What is the jersy number of the player with playerID: 229397?



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

```
MATCH (cl:Club)-[:BASED_IN]→
(cc:ClubCountry {club_country:
"Australia"})
RETURN cl.club AS clubName
```

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

Started streaming 5 records after 1 ms and completed after 17 ms.

Figure query 2. Which clubs are based in Australia?

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

```
MATCH (p:Player {player: 'Lionel
MESSI'})-[r:PLAYS_FOR]→(cl:Club)
RETURN cl.club AS club
```

	club
1	"FC Barcelona"

Figure query 3. Which club does Lionel MESSI play for?

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

MATCH (p:Player {player: 'Lionel MESSI'})
RETURN p.age AS age

```
MATCH (p:Player {player: 'Lionel
MESSI'})
RETURN p.age AS age
```

	age
1	26

Figure query 4. How old is Lionel MESSI?

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

```
MATCH (p:Player {player: 'Lionel MESSI'})-[:PLAYS_FOR]→(cl:Club)-[:BASED_IN]→(cc:ClubCountry)
RETURN cc.club_country AS Country
```

	Country
1	"Spain"

Figure query 5. In which country is the club that Lionel MESSI plays for?

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

```
MATCH (p:Player)-[:REPRESENTS]→(c:Country {country: 'Australia'})
MATCH (p)-[:PLAYS_FOR]→(cl:Club)
RETURN DISTINCT cl.club AS clubs
```

	clubs
6	"Crystal Palace FC"
7	"Club Brugge KV"
8	"Brisbane Roar FC"
9	"Preston North End FC"
10	"SC Heracles Almelo"
11	"FSV Frankfurt"

Started streaming 21 records after 1 ms and completed after 8 ms.

Figure query 6. A club that has players from Australia

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

```
MATCH (p:Player)-[:PLAYS_FOR]->(cl:Club {club: 'Melbourne Victory FC'})
RETURN p.player AS player, p.age AS age
ORDER BY p.age ASC
```

	player	age
1	"James TROISI"	25
2	"Mark MILLIGAN"	28

Started streaming 2 records after 1 ms and completed after 10 ms.

Figure query 7. Find all players play at Melbourne Victory FC, returning in ascending orders of age

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: "Forward"})-[:REPRESENTS]->(nation:Country {country: "Argentina"})
RETURN p.player AS playerName, p.caps AS caps
ORDER BY p.caps DESC
```

	playerName	caps
1	"Lionel MESSI"	84
2	"Sergio AGUERO"	50
3	"Gonzalo HIGUAIN"	36
4	"Ezequiel LAVEZZI"	29
5	"Rodrigo PALACIO"	21

Started streaming 5 records after 1 ms and completed after 12 ms.

Figure query 8. Find all Forward players in the national team of Argentina, returning in descending order of caps.

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

```
MATCH (p:Player {year: "1987"})-
[:REPRESENTS]→(nation:Country
{country: "Argentina"})
RETURN p.player AS playerName,
p.caps AS caps
ORDER BY p.caps DESC
```

	playerName	caps
1	"Lionel MESSI"	84
2	"Sergio ROMERO"	45
3	"Gonzalo HIGUAIN"	36

Started streaming 3 records after 1 ms and completed after 2 ms.

Figure query 9. Find all players born in 1987 and in the national team of Argentina, returning in descending order of caps.

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

```
MATCH (p:Player)-[:REPRESENTS]->(nation:Country {country: "Argentina"})
MATCH (p)-[:PLAYS_FOR]->(cl:Club)
RETURN p.player AS playerName, p.internationalGoals AS goals
ORDER BY p.internationalGoals DESC
```

	playerName	goals
1	"Lionel MESSI"	37
2	"Sergio AGUERO"	21
3	"Gonzalo HIGUAIN"	20
4	"Maxi RODRIGUEZ"	15
5	"Angel DI MARIA"	9
6	"Ezequiel LAVEZZI"	4
7		

Started streaming 23 records after 1 ms and completed after 4 ms.

Figure query 10. Find the players that belongs to the same club in national team of Argentina, returning in descending order of international goals.

11. Count how many players are born in <*a specific year*>.

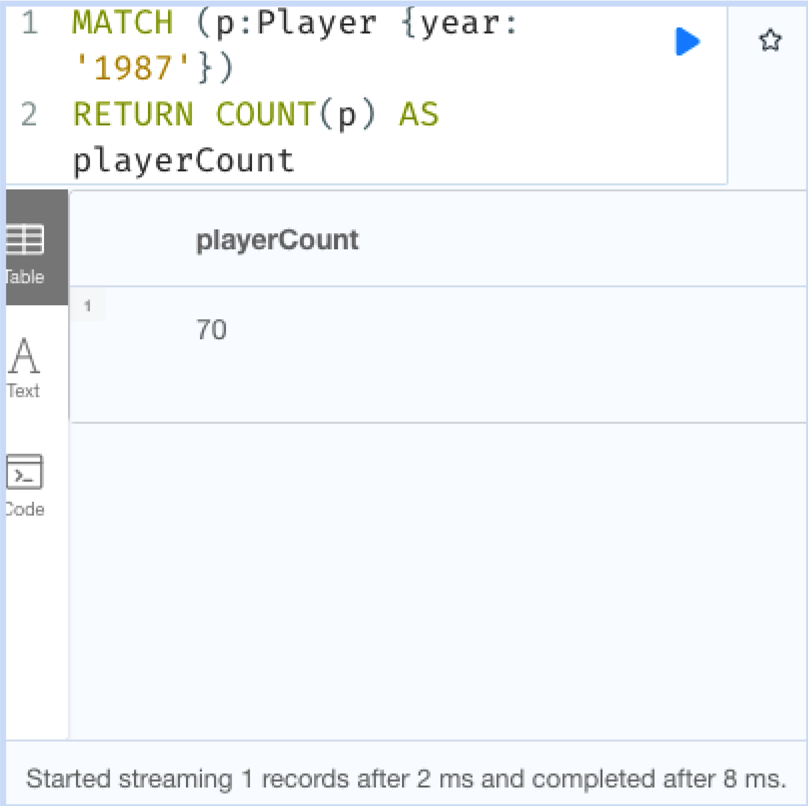


Figure query 11. Count how many players were born in 1987.

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

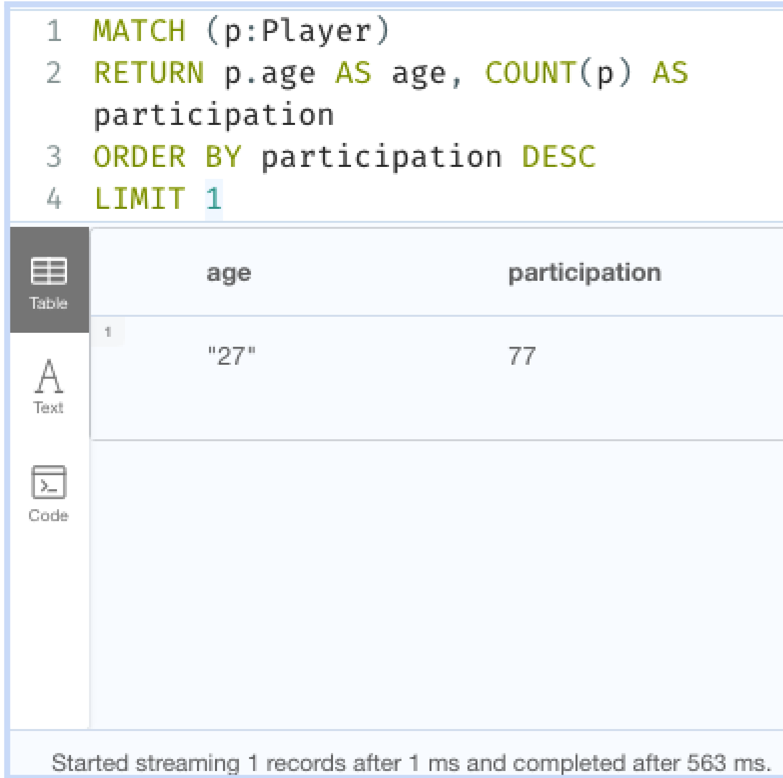


Figure query 12. Which age has the highest participation in the 2014 FIFA World Cup?

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

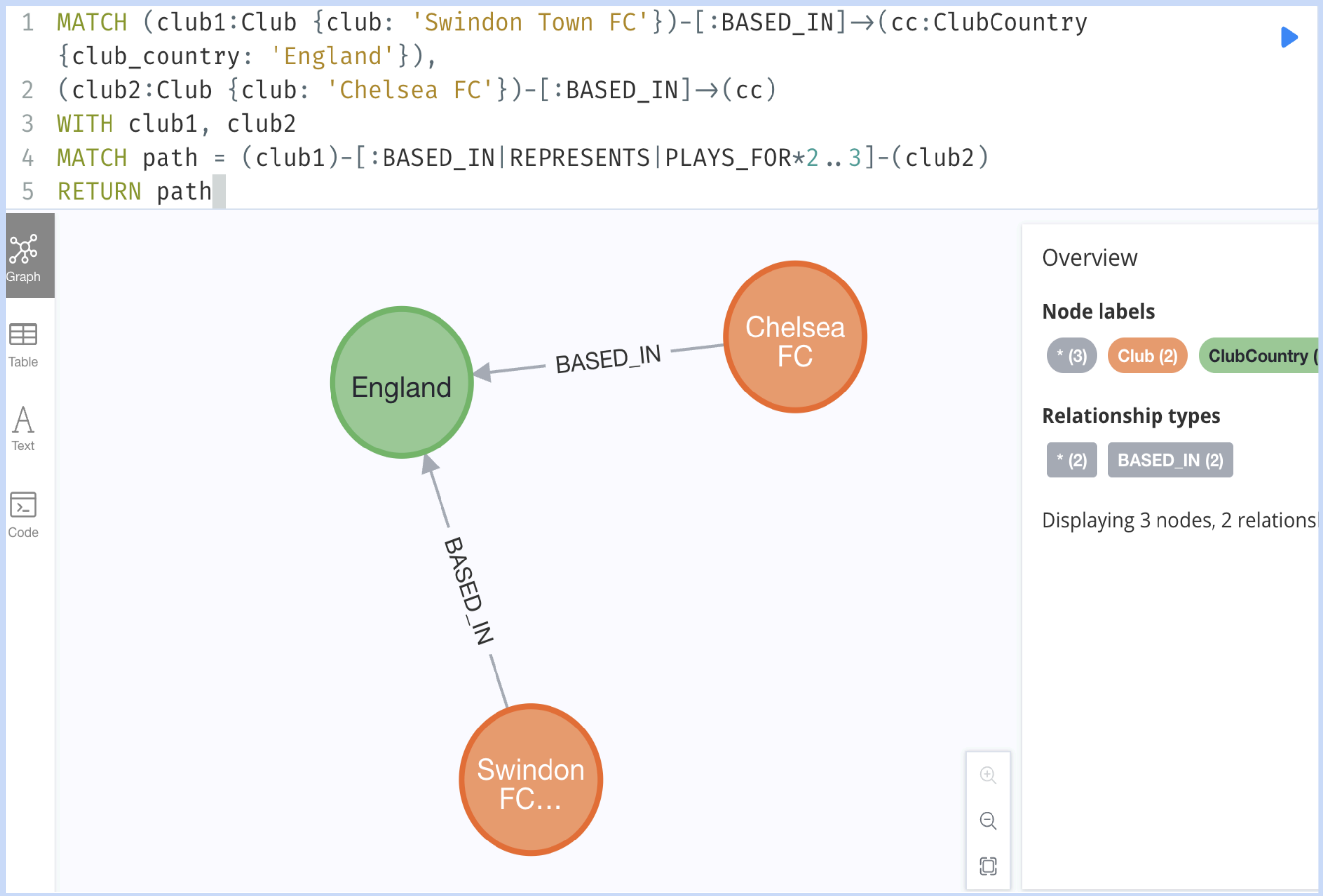


Figure query 13. The path with a length of 2 between two specific clubs

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)-[:REPRESENTS]→(nation:Country)
WITH nation.country AS country, toInteger(AVG(p.internationalGoals)) AS avgGoals
RETURN country, avgGoals
ORDER BY avgGoals DESC
LIMIT 5
```

	country	avgGoals
1	"Spain"	9
2	"Germany"	9
3	"Netherlands"	7
4	"Ivory Coast"	6
5	"Uruguay"	6

Started streaming 5 records after 3 ms and completed after 198 ms.

Figure query 14. Top 5 countries with players have highest average number of international goals

15. (CITS5504 only) 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.

Figure query 15 assumption 1-2: 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

**** Assumption 1**: We assume that the closest number of caps is 0 which means the players have the same number of caps.*

```
MATCH (p1:Player)-[:REPRESENTS]->(nation:Country)<-[:REPRESENTS]-(p2:Player)
WHERE p1.position <> p2.position AND p1.playerID < p2.playerID
WITH p1, p2, ABS(p1.caps - p2.caps) AS capDifference
WITH min(capDifference) AS minCapDifference
MATCH (p1:Player)-[:REPRESENTS]->(nation:Country)<-[:REPRESENTS]-(p2:Player)
WHERE p1.position <> p2.position AND p1.playerID < p2.playerID
WITH p1, p2, ABS(p1.caps - p2.caps) AS capDifference
WHERE capDifference = minCapDifference
RETURN p1.player AS player1, p1.position AS position1, p1.caps AS caps1, p2.player AS player2, p2.position AS
position2, p2.caps AS caps2, capDifference
```

player1	position1	caps1	player2	position2	caps2	capDifference
"Jason DAVIDSON"	"Defender"	6	"Maty RYAN"	"Goalkeeper"	6	0
"Eugene GALEKOVIC"	"Goalkeeper"	8	"Ivan FRANJIC"	"Defender"	8	0
"Massimo LUONGO"	"Midfielder"	1	"Ben HALLORAN"	"Forward"	1	0
"Oliver BOZANIC"	"Midfielder"	3	"Mitch LANGERAK"	"Goalkeeper"	3	0
"MARCELO"	"Defender"	30	"OSCAR"	"Midfielder"	30	0
"WILLIAN"	"Midfielder"	6	"VICTOR"	"Goalkeeper"	6	0
"FERNANDINHO"	"Midfielder"	6	"VICTOR"	"Goalkeeper"	6	0

Started streaming 104 records in less than 1 ms and completed after 59 ms.

Figure query 15: assumption 1

**** Assumption 2: In the case that the closest number of caps is 1, which is the minimum difference of caps of the two players in the same national team when they have a different number of caps.***

```
MATCH (p1:Player)-[:REPRESENTS]->(nation:Country)<-[:REPRESENTS]-(p2:Player)
WHERE p1.position <> p2.position AND p1.playerID < p2.playerID AND p1.caps <> p2.caps
WITH p1, p2, ABS(p1.caps - p2.caps) AS capDifference
WITH min(capDifference) AS minCapDifference
MATCH (p1:Player)-[:REPRESENTS]->(nation:Country)<-[:REPRESENTS]-(p2:Player)
WHERE p1.position <> p2.position AND p1.playerID < p2.playerID
WITH p1, p2, ABS(p1.caps - p2.caps) AS capDifference
WHERE capDifference = minCapDifference
RETURN p1.player AS player1, p1.position AS position1, p1.caps AS caps1, p2.player AS player2, p2.position AS
position2, p2.caps AS caps2, capDifference
```

	player1	position1	caps1	player2	position2	caps2	capDifference
1	"Alan PULIDO"	"Forward"	5	"Jose VAZQUEZ"	"Midfielder"	4	1
2	"Massimo LUONGO"	"Midfielder"	1	"Bailey WRIGHT"	"Defender"	0	1
3	"Ben HALLORAN"	"Forward"	1	"Bailey WRIGHT"	"Defender"	0	1
4	"Jason DAVIDSON"	"Defender"	6	"Mathew LECKIE"	"Forward"	7	1
5	"Eugene GALEKOVIC"	"Goalkeeper"	8	"Mathew LECKIE"	"Forward"	7	1
6	"Mathew LECKIE"	"Forward"	7	"Maty RYAN"	"Goalkeeper"	6	1
7							

Started streaming 180 records after 1 ms and completed after 57 ms.

Figure query 15: assumption 2

5.2 Self-designed queries

Query 16.1 Find the player with the highest number of caps in Australia

```
MATCH (p:Player)-[:REPRESENTS]→
(nation:Country {country:
"Australia"})
RETURN p.player AS playerName,
p.caps AS caps
ORDER BY p.caps DESC
LIMIT 1
```

	playerName	caps
1	"Mark BRESCIANO"	73

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

Figure Query 16.1. Find the player with the highest number of caps in Australia

Query 16.2 List all players who play for clubs in their home country.

```
MATCH (p:Player)-[:PLAYS_FOR]→(cl:Club)-[:BASED_IN]→(cc:ClubCountry)
WHERE p.playsInHomeCountry = true
RETURN p.player, cl.club, cc.club_country
```

	p.player	cl.club	cc.club_country
1	"Alan PULIDO"	"Tigres UANL"	"Mexico"
2	"Adam TAGGART"	"Newcastle United Jets FC"	"Australia"
3	"David VILLA"	"Atletico Madrid"	"Spain"
4	"Oribe PERALTA"	"Club Santos Laguna"	"Mexico"
5	"FRED"	"Fluminense FC"	"Brazil"
6	"Fabian SCHAER"	"FC Basel"	"Switzerland"
7			

Started streaming 260 records after 12 ms and completed after 27 ms.

Figure Query 16.2 List all players who play for clubs in their home country.

Query 16.3: Find the average height of players in each national team, returning in descending order of average height

```
MATCH (p:Player)-[:REPRESENTS]→(nation:Country)
RETURN nation.country AS country, TOINTEGER(AVG(p.height)) AS avgHeight
ORDER BY avgHeight DESC
```

	country	avgHeight
1	"Bosnia & Herzegovina"	185
2	"Germany"	185
3	"Croatia"	184
4	"Greece"	184
5	"Belgium"	184
6	"Iran"	183
7		

Started streaming 32 records after 1 ms and completed after 8 ms.

Figure Query 16.3. Find the average height of players in each national team, returning in descending order of average height

6. Discussion

6.1 Capability of Graph Databases Compared to Relational Databases

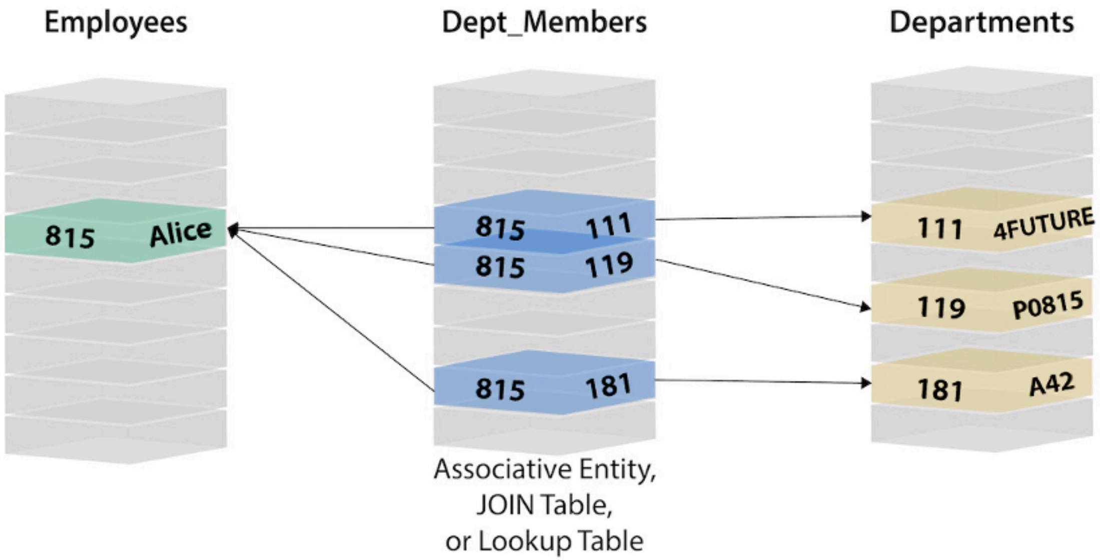


Figure 6.1a. Relational database model (Neo4j, n.d.)

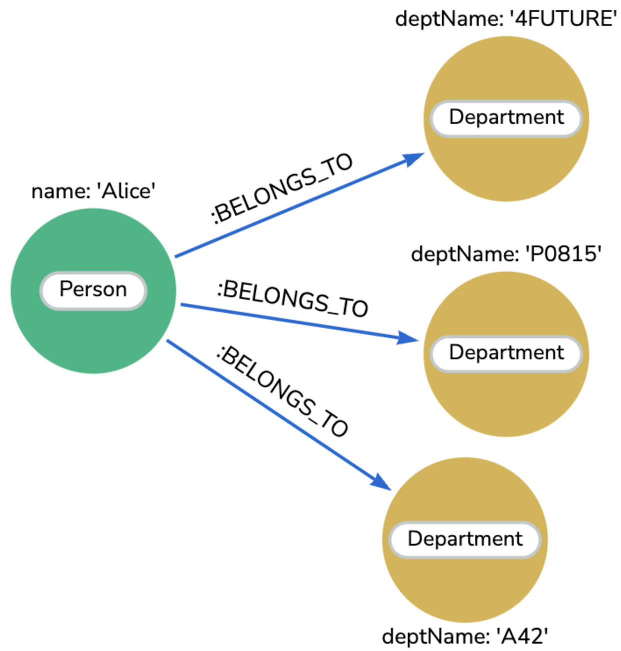


Figure 6.1b. Graph database model (Neo4j, n.d.)

There are pros and cons when using relational database models or graph database models. However, the transition from traditional relational database to graph database delivers key achievements which would be challenges and not feasible in the traditional model such as flexibility, simplicity, agility and particularity.

- ***Flexibility in relationship modeling and simplicity***: In graph databases, it offers benefits from data integration and data linkage due to its flexible relationship modeling capability. It allows the users to create node tables and relationship tables to connect between entities with explicit representation relationships in the model. This makes it more feasible and easier in a complex-interaction relationship database especially in the large dataset. Refer to Figure 5.1a, in the traditional relational database model, we have the person Alice with ID '815' in the Employees table JOIN with Dept_Members table which has three rows matching with the search for the employee ID '815', then we can locate the department of this employee. In the graph database, we can retrieve the department of the Person named 'Alice' with the 'BELONGS_TO' relationship which is more direct and simplifier (Neo4j, n.d.). Regarding many-to-many relationships, retrieving multiple JOIN tables with tangled relationships in the relational database, it would create complexity and unscalability as graph database modeling can scale up and down as required. The graph-based management implies straightforward relationships in properties, which enables simpler queries.

- ***Storage and handling large datasets:*** In the era of thriving rapid technology landscapes, handling large datasets for high performance of analytics is an ideal in mastering challenges posed by big data. The large dataset in regards of complexity and size, creating confrontation with outdated relational database management in processing large datasets as large datasets are non-relational or unstructured rising significant challenges when dealing with these semi-structured large datasets (Sivarajah et al., 2017). Neo4j employs compressed storage that increases more space for disk and maintains data storage efficient performance. A research found that there are 13.4% reduction in disk space and 30% improvement efficiency in retrieving by conducting to regulate the Neo4j database compared to the methods of relational database on oilfield ontology evaluation (Gong et al., 2018). Graph-database management is now widely approached by large-scale data enterprises such as software, telecommunication and internet companies (Marzi, 2012).

- ***Agility and adaptability***: Speed and adaptability are one of the key advantages for graph-based management which can be challenges in the relational database. It allows for adaptation to data modification and addition, in which a new relationship is added to the existing structure consistently without long processing steps such as in the traditional model. RDBMS delivers slow and long-consumed time for redesigning the schema that obstructs software development processes and impedes scale and innovation efficiently, whereas the graph database model is pacing in delivering dynamic modern applications and business requirements with its ability to change (Packer, 2021).

- ***Insurance fraud detection***: With regards to the ability of high interconnected data efficiency, graph-based database management does better than relational databases in supporting detection of anomalies and making the information system responsive in insurance companies (Graph tech, 2023). Anti-fraud investigators normally rely on automation tools when investigating clues and connections from various sources, therefore, with the automation system can deliver faster way for detection, which traditional rules-based systems produces false positive detection (Linkurious, 2022)

- ***Network system optimisation:*** The graph database can do well in network resource management optimisation as its automatic and direct relationship in connecting data, which supports in supply chain industry, chemical engineering, energy systems and other industries that required high quality in network system management, while relational databases takes longer time consuming in visualising topology (Graph tech, 2023).

6.2 How Graph Data Science Applied to Practical Application

- ***Revolutionising in Healthcare***: The application of Graph Data Science (GDS) emerges in the healthcare industry in efficiency of drug discoveries and patient journey improvement. Based on the graph storage-efficient ability, it can store information spanning over 50 years including genes, compounds, diseases, symptoms and side effects which concrete strong data source foundation for researchers delve into new drug predictions by evaluating relationships, network structures, and similarities within graph database to advocate potential implementation of existing drugs (Tech First, 2020). Moreover, GDS contributes to the improvement of the patient pathway in chronic or serious illness treatments in which these treatments usually make progress over a period of time as such researchers and healthcare providers deploy graph-based systems to have understanding what would influence patients through observing mapped graphs to sequence alternatives and path splits after visits (Tech First, 2020).

- ***Strengthening in cybersecurity***: Nowadays, cyber threats are posing significant risks and harms to not only businesses but also individuals. GDS enhances to detect and mitigate threats of cyber by analysing patterns and anomalies in the graph system through visualisation of graph topology, which can prevent cyber attacks and protect confidential data (Hong, 2023). The graph database inscribes what are restrictions of traditional relational databases resembling scalability and rigidity schema concerns, thus Cypher Query Language and Neo4j graph play an important role in developing cybersecurity practices by enforcing access controls and facilitating data encryption (Bhalekar et al., 2024). Therefore, graph-based management is also widely used in finance institutions and insurance companies where fraud detection is a priority pose for the business.

7.References

- Bhalekar, P. M., & Saini, J. R. (Publisher: IEEE). Comprehensive Exploration of the Role of Graph Databases like Neo4j in Cyber Security. *IEEE*. Retrieved from https://ieeexplore.ieee.org/document/10497325

- OpenAI. (2022). ChatGPT: Conversational AI developed by OpenAI. Retrieved from https://openai.com/chatgpt
- FIFA World Cup Data. (2014). FIFA dataset. Retrieved from the dataset provided in the project.
- Gong, F., Ma, Y., Gong, W., Li, X., Li, C., & Yuan, X. (2018). Neo4j graph database realizes efficient storage performance of oilfield ontology. *PLOS ONE*, 13(11), e0207595. https://doi.org/10.1371/journal.pone.0207595
- Graph tech. (2023, May 8). Graph database vs. relational database. Memgraph. Retrieved from https://memgraph.com/blog/graph-database-vs-relational-database
- Hong, Z. (2023, July 26). Understanding Graph Databases: Unleashing the Power of Connected Data in Data Science. *Medium*. Retrieved from https://medium.com/@zhonghong9998/understanding-graph-databases-unleashing-the-power-of-connected-data-in-data-science-2c6393f8c871
- Linkurious. (2022, April 21). Enhancing insurance fraud investigation with graph analytics. Linkurious. Retrieved from https://linkurious.com/blog/enhancing-insurance-fraud-investigation-with-graph-analytics/
- Marzi, M. (2012), “Introduction to Graph Databases. Quick look at trends in data, nosql, graph databases, and Neo4j”, *Chicago Graph Database Meet-Up*, April 29, 2012
- Neo4j. (n.d.). Graph databases vs. relational databases. Retrieved May 24, 2024, from https://neo4j.com/docs/getting-started/appendix/graphdb-concepts/graphdb-vs-rdbms/
- Packer, D. (2020, December 21). Graph Data Platforms: A Brief Introduction. Neo4j Blog. Retrieved from https://neo4j.com/blog/graph-data-platforms-a-brief-introduction/
- Sivarajah, U., Kamal, M. M., Irani, Z., & Weerakkody, V. (2017). Critical analysis of Big Data challenges and analytical methods. *Journal of Business Research*, 70, 263-286. https://doi.org/10.1016/j.jbusres.2016.08.001
- Tech First. (2020, November 27). Graph Data Science 101: Using Graph Data Science in the Real World. *Medium*. Retrieved from https://techfirst.medium.com/graph-data-science-101-using-graph-data-science-in-the-real-world-afc6c2663ecd
- The University of Western Australia. (n.d.). Teaching material from unit CITS5504.
- TribeHired. (n.d.). Cover page image. Retrieved from https://tribehired.com/

8. Software and Tool

1. Python 3.8.8
2. Neo4j Desktop - 1.5.9
3. Text files
4. Arrows.app
5. CSV files