Data Warehousing Project-2

(CITS5504)

Udaymithra Kalla (23858856)

Dharani Kumari Nagali (23870369)

Project Objective:

The objective of the project is to design, implement and use a graph database for efficient storage, retrieval and analysis of 2014 FIFA World Cup Data with special emphasis on the subtle relationship and interaction among players, clubs & countries.

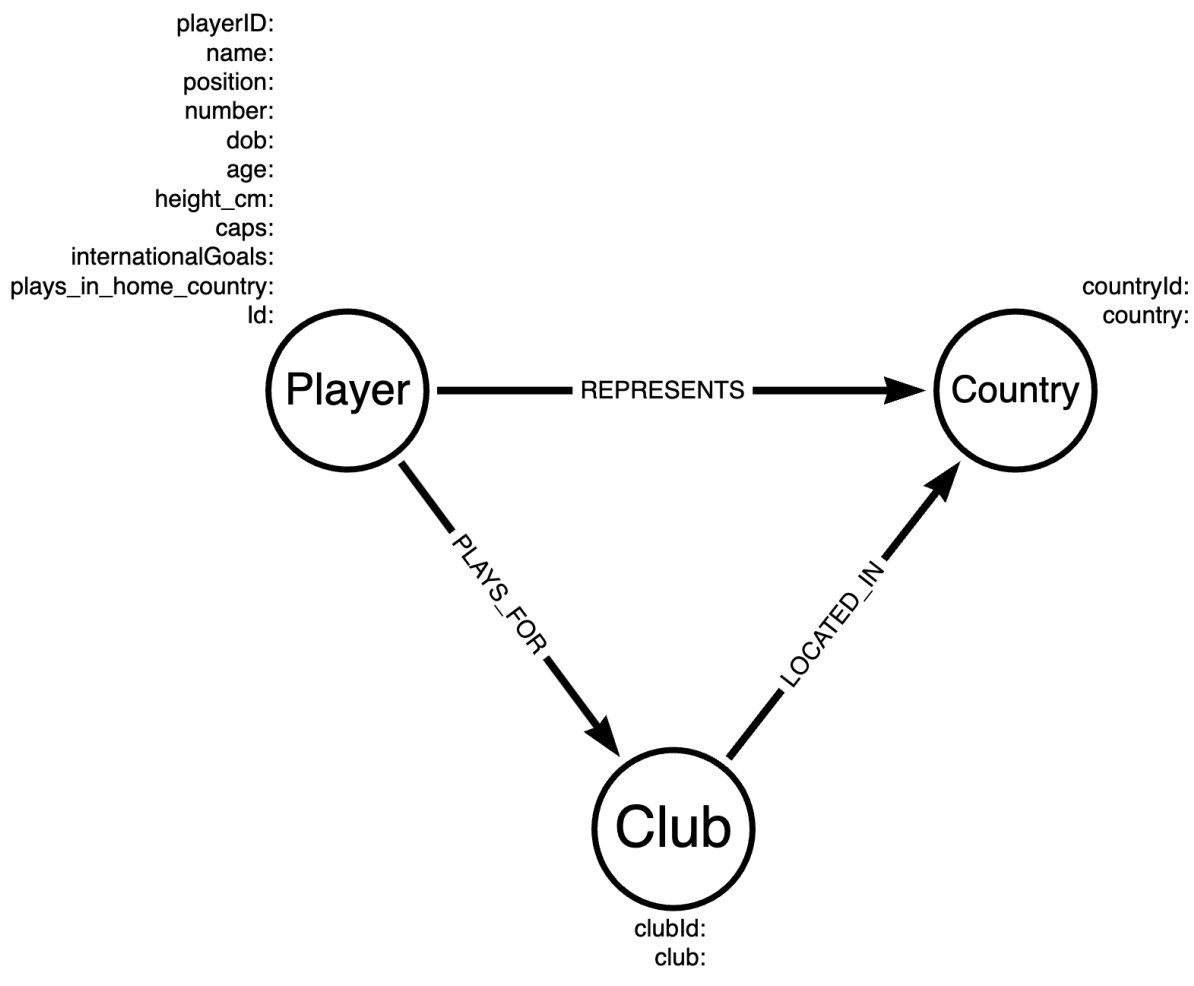
Graph Database:

A graph database is special database designed to manage and store the data in form of nodes and edges. Graph database excels when dealing with highly connected data. They do not need to store null values and can add new properties or nodes on the fly. The most common form of graph model is property graph model where the graph contains nodes and relationships. A node can have zero or more properties and relationships can also contain properties.[7]

Design and Implementation Process:

In this project, we designed a Neo4j graph database that facilitates the management of player, country and club data in FIFA World Cup context. We utilized the capabilities of Neo4j's database design to store complex relationships and employ powerful data modelling techniques to represent the sports domain. The database seamlessly integrated CSV data established relationships between entities, and prioritized scalability and flexibility for future expansions.

The following image describes the design of our database which efficiently answers/queries all the questions given in the project doc.



Node Structure and Relationships:

In our design, we have three nodes **Player**, **Club** and **Country**. Each node has its own properties which can been seen in the above image. The properties define the characteristics of the node.

Relationships between nodes:

- **Player** and **Club** nodes are connected through **PLAYS_FOR** Relationship. **PLAYS_FOR** relationship contains **Id** of the player and **clubId** which denotes the unique number for each club.
- **Player** and **Country** nodes are connected through **REPRESENTS** Relationship. **REPRESENTS** relationship contains Id of the **player** and **playerCountryId** which represents the unique number to each country where the player represents in FIFA World cup.

- **Club** and **Country** nodes are connected through **LOCATED_IN** Relationship. LOCATED_IN relationship contains **clubId** and **clubCountryId** which is a unique number to the countries where clubs are located.

Rationale behind the design:

- This design separates the main entities (players, clubs and countries) into distinct nodes, each with relevant properties. This enhances data clarity and makes database easier to manage and query.
- This structure allows for adding new properties or relationships as the database evolves without disrupting/changing existing data structures.
- This design supports efficient querying for all the queries given in the project description.
- The PLAYS_FOR relationship provides information of the players with their respective club affiliations. This helps in determining in queries related to clubs or stats of the players with their clubs.
- The LOCATED_IN relationship between club and countries determines the location of the clubs located in different parts of the world.
- REPRESENTS relationship determines the nationality of the player for FIFA 2014 world cup which help us in efficient querying of any statistics of a player in world cup.

This structure allows us to represent the connections between players, clubs, and countries in a clear and organized way.

ETL Process:

This Extract, Transform, Load (ETL) process streamlines data extraction from a dataset[CSV file (FIFA2014-allplayers.csv)], transform and load into separate CSV files for nodes i.e. Player, Club, and Country data and for three relationships i.e. PLAYS_FOR, REPRESENTS, LOCATED_IN. The process involves:

- To create the csv files for nodes and relationships in our design, we used Python dataframe which was available in **pandas library**.
- Then loaded the dataset[original csv file (FIFA2014-allplayers.csv)] into pandas data frame and stored in a variable called **fifa_table**.

[1]:
```
#import pandas library
import pandas as pd
```

[2]:
```
#Loading the csv file into pandas data frame called fifa_table
fifa_table = pd.read_csv('./FIFA2014-all players.csv')
```

[3]:
```
#Prints few rows of data from fifa_table
fifa_table.head()
```

[3]:

	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

- We have created a new data frame called players with all necessary columns(player details) from **fifa_table** and stored in **player** dataframe and reset the index with **reset_index(drop = True)** method.

- The **D.O.B(date of birth) column** of the players is currently as string data type in format of **dd.mm.yyyy** and we converted it into date data type and format into **yyyy-mm-dd**(which is mostly used format of date in Neo4j browser which helps for easy querying) with the help of **to_datetime()** and **dt.strftime()**

methods.

```
[4]: #Create a new data frame called players with all player details from fifa_table and reset the index
players = fifa_table[['Player id', 'Player', 'Position', 'Number', 'D.O.B', 'Age', 'Height (cm)', 'Caps', 'International goals',
```

```
[5]: # Convert the 'D.O.B' column to datetime format using the specified 'dd.mm.yyyy' format, and then reformat the dates to the 'yyy
players.loc[:, 'D.O.B'] = pd.to_datetime(players['D.O.B'], format='%d.%m.%Y').dt.strftime('%Y-%m-%d')
```

- After resetting the index, started the index from 1 instead of zero and created a new column called Id which contains the index numbers. New column is created which acts as primary key which identifies each node uniquely and is helpful while creating relationships.

- Next, rearranged the column order to make Id to come at first position and assigned the order to players data frame.

```
[6]: #Start the index from one rather instead of zero
players.index += 1
#Create a new column Id which has index values
players['Id'] = players.index
```

```
[7]: #Rearrange the column order
column_order = ['Id', 'Player id', 'Player', 'Position', 'Number', 'D.O.B', 'Age', 'Height (cm)', 'Caps', 'International goals',
```

```
[8]: #Reorder the columns
players = players[column_order]
```

- After rearranging the columns, exported the players data frame to a csv file called Players.csv with the help of **to_csv ()** method

```
[9]: # Export the 'players' DataFrame to a CSV file named 'Players.csv'
players.to_csv('./Players.csv', index=False)
```

- After Players.csv file is created. we created a clubs data frame which stores **Clubs** column. There will be duplicate rows of clubs ,so to remove them we have used a method called **drop_duplicates()** to keep unique clubs in the data frame and reset the index with reset_index() method
- After resetting the index ,incremented the index from 1 for the dataframe and created a new column called clubId in clubs dataframe which stores the index values which are unique across the rows.

- After creation of new column, arranged the order of columns for clubs dataframe and exported the data frame to csv file called Clubs.csv file.

```
[10]: #Create a clubs data frame with Club column in fifa_table and also remove duplicate rows,reset the index
clubs = fifa_table[['Club']].drop_duplicates().reset_index(drop=True)
```

```
[11]: # Increment the index of the 'clubs' DataFrame to start from 1 instead of 0.
clubs.index += 1
#Create a new column clubId which has index values
clubs['clubId'] = clubs.index
```

```
[12]: #Rearrange the column order
column_order = ['clubId','Club']
```

```
[13]: #Reorder the columns for clubs dataframe
clubs = clubs[column_order]
```

```
[14]: # Export the 'clubs' DataFrame to a CSV file named 'Clubs.csv'
clubs.to_csv('./Clubs.csv', index=False)
```

- For Countries.csv file, we have combined the **Country and Club(country) columns** from fifa_table data frame and stored only unique values by using **unique()** method and stored in countries variable.
- Later, converted the countries variable into a data frame and reset the index.
- After the index reset, incremented its value from 1 and created a new column countryId and inserted in the countries_df data frame.
- Exported the countries_df frame after rearranging the column to make sure that countryId is the first column in the csv file.

```
[15]: # Combine the 'Country' and 'Club (country)' columns from the 'fifa_table' DataFrame, remove duplicates, and store the unique va
countries = pd.concat([fifa_table['Country'], fifa_table['Club (country)']]).unique()

[16]: #Convert countries variable into a data frame and name it as countries_df,reset the index
countries_df = pd.DataFrame(countries, columns=['Country']).reset_index(drop=True)

[17]: # Increment the index of the 'countries_df' DataFrame to start from 1 instead of 0.
countries_df.index += 1
#Create a new column countryId which has index values
countries_df['countryId'] = countries_df.index

[18]: #Rearrange the column order
column_order = ['countryId','Country']

[19]: #Reorder the columns for countries_df
countries_df = countries_df[column_order]

[20]: # Export the 'countries_df' DataFrame to a CSV file named 'Countries.csv'
countries_df.to_csv('./Countries.csv', index=False)
```

- After all the csv files for nodes are created, to create the csv files for relationships need to merge the data frames into **fifa_table** dataframe to get the new columns(Id,clubId,countryId) into it.
- To **merge** the players data frame into fifa_table, used **merge()** function on **Player id** column using **inner join**.
- In similar way, merged the clubs data frame with fifa_table on Club column which is present in both data frames by using inner join.

```
[21]: # Merge the 'players' DataFrame with the 'fifa_table' DataFrame on the 'Player id' column using an inner join.
fifa_table = fifa_table.merge(players[['Player id', 'Id']], on='Player id', how='inner')

[22]: #Print the few records of fifa_table dataframe after insertion of Id column
fifa_table.head()
```

[22]:

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

```
[23]: # Merge the 'clubs' DataFrame with the 'fifa_table' DataFrame on the 'Club' column using an inner join.
fifa_table = fifa_table.merge(clubs[['Club', 'clubId']], on='Club', how='inner')
```

- Now , to merge **countries_df** dataframe which contains **countryId** and **Country** column which represents both countries of club and as well nationality of player, so need to perform two merges, one to get **clubCountryId** and other to get **playerCountryId**.
- To get playerCountryId in **fifa_table** dataframe, merged the **fifa_table** with countries_df on Country column using inner join ,which results Country column again along with **countryId** used suffixes with value _dup to get rid of them after merging.
- After merging is performed, renamed the countryId to playerCountryId in fifa_table to distinguish between **playerCountry** and **clubCountry**.
- Similar approach is again performed to get clubCountryId which is achieved by renaming the countryId to clubCountryId after merging which is done with **inner join** on **Country** column from countries_df frame and **Club (country)** column of fifa_table dataframe.

```
[24]: # Merge the 'countries_df' DataFrame with the 'fifa_table' DataFrame on the 'Country' column using an inner join , and add a suf
fifa_table = fifa_table.merge(countries_df[['countryId','Country']],how='inner',on='Country',suffixes=('', '_dup'))
fifa_table.rename(columns={'countryId': 'playerCountryId'}, inplace=True)
```

```
[25]: # Merge the 'countries_df' DataFrame with the 'fifa_table' DataFrame on the 'Club(Country)' column using an inner join , and ado
fifa_table = fifa_table.merge(countries_df[['countryId','Country']], left_on='Club (country)', right_on='Country', how='inner',
fifa_table.rename(columns={'countryId': 'clubCountryId'}, inplace=True)
```

```
[26]: fifa_table.head()
```

[26]:

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

- After merging is performed, we dropped the duplicate column named Country_dup in fifa_table dataframe and extracted the modified data frame into a new csv file called ModifiedFIFA2014-all players.csv.

```
[27]: # Drop duplicate country columns
fifa_table.drop(columns=['Country_dup'], inplace=True)
```

```
[28]: fifa_table.head()
```

[28]:

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

```
[29]: #Extract the modified fifa_table dataframe to a csv file
fifa_table.to_csv('./ModifiedFIFA2014-all players.csv', index=False)
```

- Now, to create a csv file for relationship **PLAYS_FOR** , extracted two columns namely **Id** and **clubId** from **fifa_table** and stored them in a new data frame called **plays_for**.
- To drop any duplicates present, we used **drop_duplicates()** method on the **plays_for** dataframe and then exported the data frame into a csv file with the help of **to_csv()** method.

```
[30]: #Extract 'Id' and 'clubId' columns from the fifa_table DataFrame and store them in a new DataFrame named plays_for.
plays_for = fifa_table[['Id', 'clubId']]
```

```
[31]: #Drop any duplicate rows if present
plays_for = plays_for.drop_duplicates()
```

```
[32]: # Export the 'plays_for' DataFrame to a CSV file named 'Plays_For_Relationship.csv'
plays_for.to_csv('./Plays_For_Relationship.csv', index=False)
```

- Similarly, for relationship **REPRESENTS**, extracted two columns **Id** and **playerCountryId** from fifa_table data frame and stored in a new data frame called **represents** and drop duplicates if present and exported it into a new csv file with **to_csv()** method.

- To get csv file for relationship **LOCATED_IN** , extracted clubId and clubCountryId from fifa_table data frame and stored it in **located_in** data frame then removed duplicate rows with the help of **drop_duplicates()** function.
- Exported the data frame to a csv file called **Located_In_Relationship.csv**.

```
[33]: #Extract 'id' and 'playerCountryId' columns from the fifa_table DataFrame and store them in a new DataFrame named represents
represents = fifa_table[['Id', 'playerCountryId']]

[34]: #drop any duplicates if present
represents = represents.drop_duplicates()

[35]: # Export the 'represents' DataFrame to a CSV file named 'Represents_Relationship.csv'
represents.to_csv('./Represents_Relationship.csv', index=False)

[36]: #Extract 'clubId' and 'playerCountryId' columns from the fifa_table DataFrame and store them in a new DataFrame named located_in
located_in = fifa_table[['clubId', 'clubCountryId']]

[37]: #drop any duplicates if present in the dataframe
located_in = located_in.drop_duplicates()

[38]: # Export the 'located_in' DataFrame to a CSV file named 'Located_In_Relationship.csv'
located_in.to_csv('./Located_In_Relationship.csv', index=False)
```

- After all the relationships csv files are created, then ETL Process is completed which overall involves the original dataset to create three csv files for three nodes and three csv files for relationship.

Importing CSVs into Neo4j using Cypher:

- To Import the created csv files into neo4j, first step is to create a project in **Neo4j desktop**. We created a project named **Project2**.
- After the project is created, we added a **local dbms** and configured it with **name** and **password**.
- Then imported the created csv files(both nodes and relationship csv files) into Import folder of the project and started the project ,once it becomes active ,need to click on open button which opens Neo4j Browser.
- To create the Player Node in Neo4j,In the query box, load the players.csv file with **LOAD CSV WITH HEADERS FROM** Command.
- Then, created the Player node with properties mentioned in the design and headers from the csv file.(**properties** are **before semicolon**(:) and **Headers** are **after semicolon** which should be same as in the file including **case-sensitivity**.
- **row** refers to the current row in the csv file which it is processing.
- We have converted some columns to integer when loading into Neo4j and it is done with the help of **toInteger()** function and dob is converted to date with help of **date()** function.
- After running the query, player node label gets created in Neo4j browser.
- Backtick symbol(`) is used when there are spaces between the header in the csv file.

```
LOAD CSV WITH HEADERS FROM 'file:///players.csv' AS row
CREATE (p:Player {
Id:toInteger(row.Id),
playerID: toInteger(row.`Player id`),
name: row.Player,
position: row.Position,
number: toInteger(row.Number),
dob: date(row.`D.O.B`),
age: toInteger(row.Age),
height_cm: toInteger(row.`Height (cm)`),
caps: toInteger(row.Caps),
internationalGoals: toInteger(row.`International goals`),
plays_in_home_country: row.`Plays in home country?`
})
```

- Similarly, **Club** node and **Country** node are created by loading them with **LOAD CSV WITH HEADERS command** and create nodes with the respective properties and it can be seen below.

```
//Club Node

LOAD CSV WITH HEADERS FROM 'file:///clubs.csv' AS row
CREATE (cl:Club {
clubId:toInteger(row.clubId),
club:row.Club
})
```

```
//Country Node

LOAD CSV WITH HEADERS FROM 'file:///countries.csv' AS row
CREATE (c:Country {
    countryId:toInteger(row.countryId),
    country:row.Country
})
```

- After the nodes, get created in Neo4j. We created relationships between the nodes. First, to create **PLAYS_FOR** Relationship between **Player** and **Club**, we loaded the csv file with **LOAD CSV WITH HEADERS FROM** Command and then matched the Player node and Club node with **MATCH()** Command and inserted the relationship between them and after running the code, relationship **PLAYS_FOR** gets created.

```
//PLAYS_FOR Relationship

LOAD CSV WITH HEADERS FROM 'file:///Plays_For_Relationship.csv'
AS row
MATCH (p:Player {Id: toInteger(row.Id)})
MATCH (cl:Club {clubId: toInteger(row.clubId)})
MERGE (p)-[:PLAYS_FOR]→(cl);
```

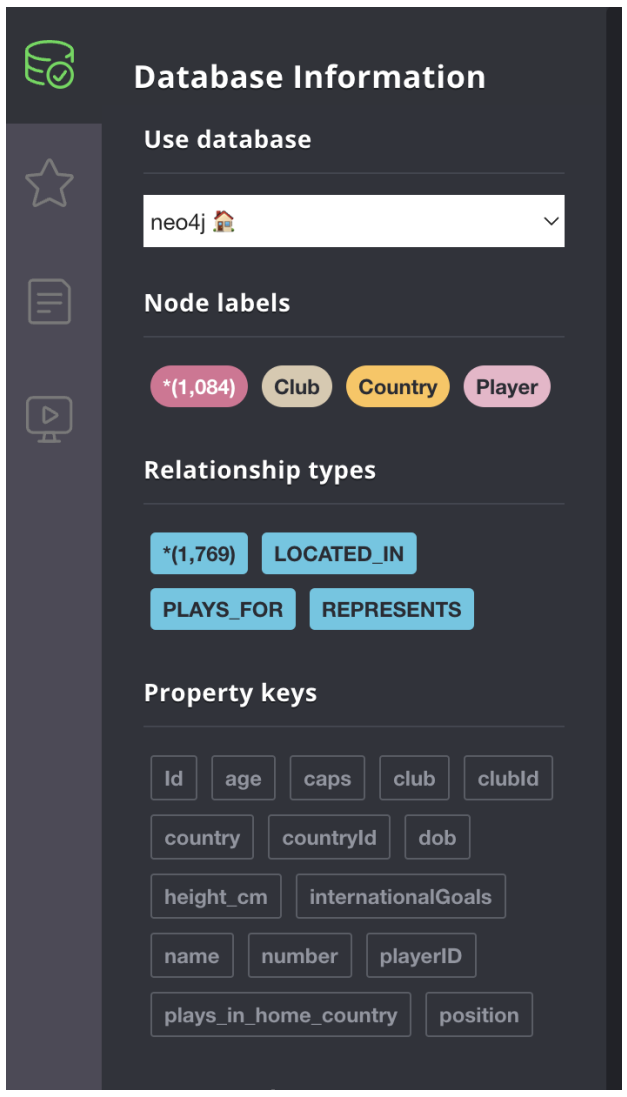
- Next, **REPRESENTS** relationship is created between **Player** node and **Country** Node. After loading the **represents** relationship csv file and matching the player node and country node with **MATCH()** Command, then inserted the relationship between them. after running the run button relationship **REPRESENTS** type gets created.

```
//Represents Relationship

LOAD CSV WITH HEADERS FROM 'file:///Represents_Relationship.csv'
AS row
MATCH (p:Player {Id: toInteger(row.Id)})
MATCH (c:Country {countryId:toInteger(row.playerCountryId)})
MERGE (p)-[:REPRESENTS]→(c);
```

- Similarly, **LOCATED_IN** Relationship is inserted between **Club** node and **Country** node. After running the run button LOCATED_IN relationship type gets created in Neo4j Browser.

```
//LOCATED_IN Relationship

LOAD CSV WITH HEADERS FROM 'file:///Located_In_Relationship.csv'
AS row
MATCH (cl:Club {clubId:toInteger(row.clubId)})
MATCH (c:Country {countryId:toInteger(row.`clubCountryId`)})
MERGE (cl)-[:LOCATED_IN]→(c);
```

- After, creation of nodes and relationships we can see/verify them in the database of Neo4j as below. Properties of all nodes can also be seen.



CYPHER QUERIES:

- After Nodes and Relationships got created in Neo4j, To answer the questions, we wrote the following cypher queries and the output of the queries can be seen in following images:

NOTE: Here for the following queries, we have chosen our own <Specific items>.

1. What is the jersey number of the player with *<a specific player id>*? (Here we choose id as 201200)

2. Which clubs are based in *<a specific country>*? (Here we choose Country as “Australia”)

```
1 //Query-2 Which clubs are based in Australia?
2 MATCH (cl:Club)-[:LOCATED_IN]→(c:Country {country: "Australia"})
3 RETURN cl.club AS ClubName;
```

Table | Text | Code

	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 6 ms and completed after 6 ms.

3. Which club does *<a specific player>* play for? (Here the Chosen Player is “Lionel Messi”)

```
1 //Query-3 Which club does <a specific player> play for?
2 MATCH (p:Player {name: "Lionel MESSI"})-[:PLAYS_FOR]→(cl:Club)
3 RETURN p.name AS PlayerName, cl.club AS ClubName;
```

Table | Text | Code

	PlayerName	ClubName
1	"Lionel MESSI"	"FC Barcelona"

Started streaming 1 records after 7 ms and completed after 8 ms.

4. How old is *<a specific player>*? (Here our player is “Oscar”)

```
1 //Query-4 How old is <a specific player>?
2 MATCH (p:Player {name: 'OSCAR'})
3 RETURN p.name AS PlayerName, p.age AS Age;
```

Table | Text | Code

	PlayerName	Age
1	"OSCAR"	22

Started streaming 1 records after 5 ms and completed after 6 ms.

5. In which country is the club that *<a specific player>* plays for? (Here, the player is “Neymar”)

```
1 //QUERY-5 In which country is the club that <a specific player> plays for?
2 MATCH (p:Player {name: 'NEYMAR'})-[:PLAYS_FOR]→(cl:Club)-[:LOCATED_IN]→
  (c:Country)
3 RETURN p.name AS PlayerName,cl.club AS ClubName, c.country AS CountryName;
```

Table | Text | Code

	PlayerName	ClubName	CountryName
1	"NEYMAR"	"FC Barcelona"	"Spain"

Started streaming 1 records after 7 ms and completed after 8 ms.

6. Find a club that has players from *<a specific country>*? (For this Query the Country is “Australia”)

```
1 //Query-6 Find a club that has players from <a specific country>?
2 MATCH (p:Player)-[:REPRESENTS]→(c:Country {country: 'Australia'})
3 MATCH (p)-[:PLAYS_FOR]→(cl:Club)
4 RETURN DISTINCT cl.club AS ClubName;
```

Table | Text | Code

	ClubName
1	"Shandong Luneng Taishan FC"
2	"FC Luzern"
3	"Swindon Town FC"
4	"Brisbane Roar FC"
5	"Adelaide United FC"
6	"Melbourne Victory FC"
7	"Crystal Palace FC"
8	"Newcastle United Jets FC"
9	"New York Red Bulls"
10	"FC Utrecht"
11	"Western Sydney Wanderers FC"
12	"Borussia Dortmund"

13	"Club Brugge KV"
14	"Jeonbuk Hyundai Motors FC"
15	"FK Austria Wien"
16	"Preston North End FC"
17	"FC Sion"
18	"SC Heracles Almelo"
19	"FSV Frankfurt"
20	"Fortuna Duesseldorf"
21	"Al Gharafa SC"

7. Find all players play at *<a specific club>,* returning in ascending orders of age?(Club we chosen is “Real Madrid CF”)

```
1 //Query-7 Find all players play at <a specific club>, returning in ascending orders
  of age.
2 MATCH (p:Player)-[:PLAYS_FOR]→(cl:Club {club: 'Real Madrid CF'})
3 RETURN p.name AS PlayerName, p.age AS Age
4 ORDER BY p.age ASC;
```

Table

Text

Code

	PlayerName	Age
2	"Karim BENZEMA"	26
3	"MARCELO"	26
4	"Angel DI MARIA"	26
5	"FABIO COENTRAO"	26
6	"Sami KHEDIRA"	27
7	"Sergio RAMOS"	28

Started streaming 12 records after 8 ms and completed after 9 ms.

Code

8	"Luka MODRIC"	28
9	"CRISTIANO RONALDO"	29
10	"PEPE"	31
11	"Xabi ALONSO"	32
12	"Iker CASILLAS"	33

8. Find all *<a specific position>* players in national team of *<a specific country>*, returning in descending order of caps? (Here, Country is “England” and Position is “Defender”)

```
//Query-8 Find all <a specific position> players in national team of <a specific
country>, returning in descending order of caps.
MATCH (p:Player)-[:REPRESENTS]→(c:Country {country: 'England'})
WHERE p.position = 'Defender'
RETURN p.name AS PlayerName, p.caps AS Caps
ORDER BY p.caps DESC;
```

Table | Text | Code

	PlayerName	Caps
1	"Glen JOHNSON"	51
2	"Phil JAGIELKA"	25
3	"Leighton BAINES"	23
4	"Gary CAHILL"	23
5	"Chris SMALLING"	11
6	"Phil JONES"	9
7	"Luke SHAW"	1

Started streaming 7 records after 6 ms and completed after 8 ms.

9. Find all players born in *<a specific year>* and in national team of *<a specific country>*, returning in descending order of caps? (Here, the year is 1990 and country is “Germany”)

```
//Query-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)-[:REPRESENTS]→(c:Country {country: 'Germany'})
WHERE date(p.dob).year = 1990
RETURN p.name AS PlayerName, p.caps AS Caps
ORDER BY p.caps DESC;
```

Table | Text | Code

	PlayerName	Caps
1	"Toni KROOS"	43
2	"Andre SCHUERRLE"	32

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

10. Find the players that belongs to the same club in national team of *<a specific country>*, returning in descending order of international goals (Here the country chosen is “Spain”)

```
//Query-10
MATCH (p:Player)-[:PLAYS_FOR]→(cl:Club),
(p)-[:REPRESENTS]→(c:Country {country: 'Spain'})
with p,cl
ORDER BY p.internationalGoals DESC
RETURN cl.club AS ClubName, collect(p.name) AS PlayerNames, collect(p.internationalGoals) AS Goals
```

Table | Text | Code

	ClubName	PlayerNames	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	"Manchester United FC"	["Juan MATA", "David DE GEA"]	[9, 0]
8	"FC Bayern Muenchen"	["Javi MARTINEZ"]	[0]
9	"SSC Napoli"	["Raul ALBIOL", "Pepe REINA"]	[0, 0]

11. Count how many players are born in *<a specific year>*? (Here, the year is 1985)

```
//Query-11 Count how many players are born in <a specific year>.
MATCH (p:Player)
WHERE date(p.dob).year = 1985
RETURN count(p) AS NumberOfPlayersBornInSpecificYear;
```

Table | Text | Code

	NumberOfPlayersBornInSpecificYear
1	66

Started streaming 1 records after 6 ms and completed after 8 ms.

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

```
//Query-12 Which age has the highest participation in the 2014 FIFA World Cup?
MATCH (p:Player)
WITH p.age AS Age, count(p) AS NumberOfPlayers
ORDER BY NumberOfPlayers DESC
RETURN Age, NumberOfPlayers LIMIT 1;
```

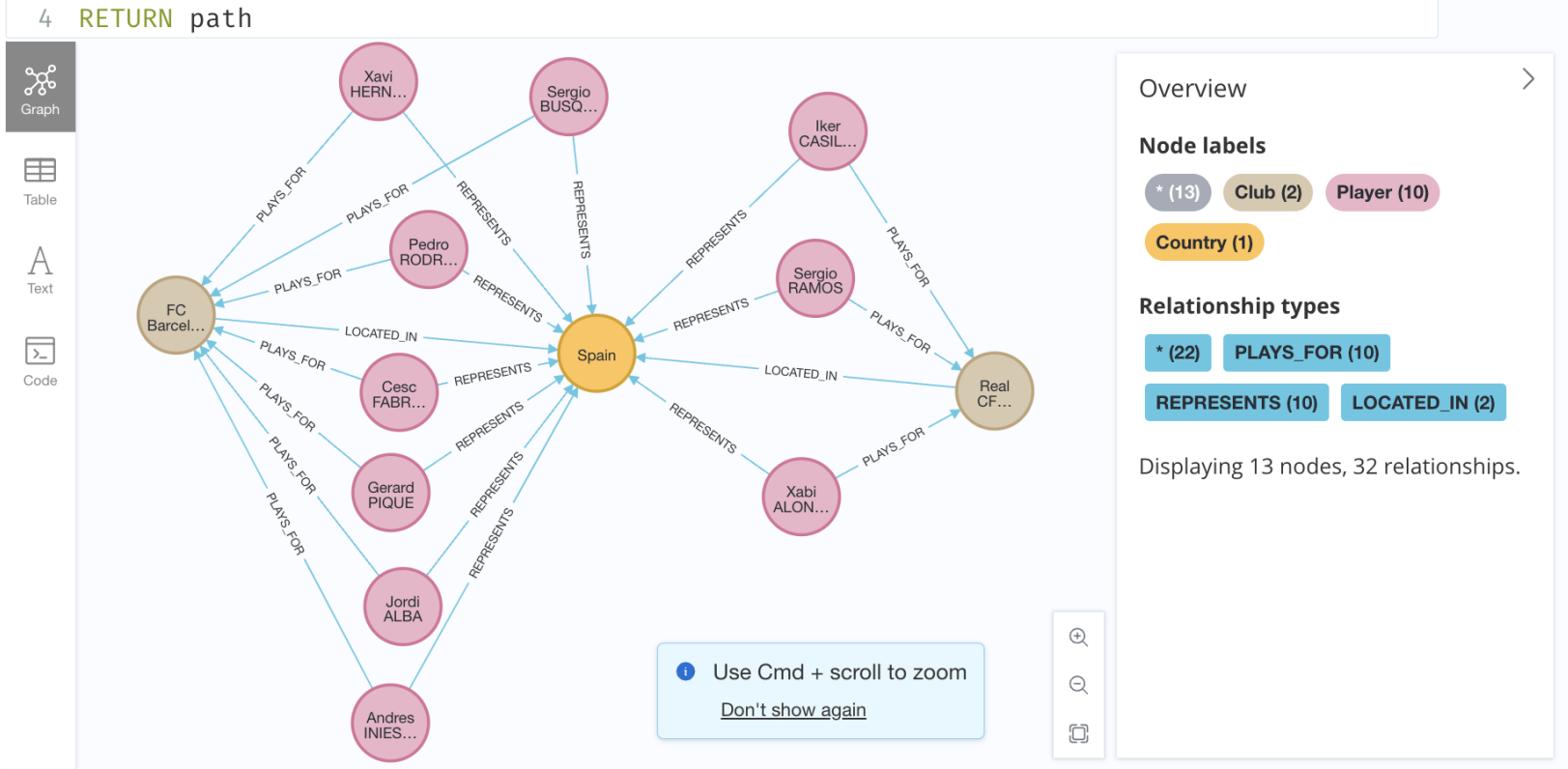
Table | Text | Code

	Age	NumberOfPlayers
1	27	77

Started streaming 1 records after 5 ms and completed after 14 ms.

13. Find the path with a length of 2 or 3 between *<two specific clubs>?* (Here the clubs chosen are "FC Barcelona" and "Real Madrid CF")

```
1  //Query-13 Find the path with a length of 2 or 3 between <two specific clubs>.
2  MATCH path = (c1:Club {club: 'FC Barcelona'})-[*2..3]-
3  (c2:Club {club: 'Real Madrid CF'})
4  RETURN path
```



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?

```
1  //Query-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.
2  MATCH (p:Player)-[:REPRESENTS]→(c:Country)
3  WHERE p.internationalGoals IS NOT NULL
4  WITH c, toInteger(avg(p.internationalGoals)) AS avgGoals
5  ORDER BY avgGoals DESC
6  RETURN c.country AS Country, avgGoals
7  LIMIT 5;
8
```

Table | Text | Code

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

Started streaming 5 records after 6 ms and completed after 7 ms.

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?

```
//Query-15

MATCH (p1:Player)-[:REPRESENTS]→(c:Country),
(p2:Player)-[:REPRESENTS]→(c)
WHERE p1.position <> p2.position AND p1.name <> p2.name AND p1.Id < p2.Id
WITH p1, p2, abs(p1.caps - p2.caps) AS capDifference
ORDER BY capDifference ASC
RETURN p1.name AS Player1, p1.position AS Position1, p1.caps AS Caps1,
p2.name AS Player2, p2.position AS Position2, p2.caps AS Caps2,
capDifference
LIMIT 10;
```

Table | Text | Code

	Player1	Position1	Caps1	Player2	Position2	Caps2	capDifference
1	"Carlos PENA"	"Midfielder"	14	"Diego REYES"	"Defender"	14	0
2	"Ben HALLORAN"	"Forward"	1	"Massimo LUONGO"	"Midfielder"	1	0
3	"Alfredo TALAVERA"	"Goalkeeper"	14	"Diego REYES"	"Defender"	14	0
4	"Miguel PONCE"	"Defender"	7	"Isaac BRIZUELA"	"Midfielder"	7	0
5	"Oribe PERALTA"	"Forward"	27	"Paul AGUILAR"	"Defender"	27	0
6	"Marco FABIAN"	"Midfielder"	14	"Diego REYES"	"Defender"	14	0
7	"Marco FABIAN"	"Midfielder"	14	"Alfredo TALAVERA"	"Goalkeeper"	14	0
8	"Miguel LAYUN"	"Defender"	12	"Hector HERRERA"	"Midfielder"	12	0
9	"Carlos PENA"	"Midfielder"	14	"Alfredo TALAVERA"	"Goalkeeper"	14	0
10	"Mitch LANGERAK"	"Goalkeeper"	3	"Oliver BOZANIC"	"Midfielder"	3	0

SELF-QUERIES:

We have designed 4 meaningful queries and there are as follows:

16. Find the top 10 players with highest international caps in descending order of caps?

```
// Query-16 Find the top 10 players with highest caps in descending order of caps
MATCH (player:Player)
WITH player, player.caps AS caps
ORDER BY caps DESC
LIMIT 10
RETURN player.name AS PlayerName, player.caps AS Caps
```

Table | Text | Code

	PlayerName	Caps
1	"Iker CASILLAS"	153
2	"Yasuhito ENDO"	143
3	"Gianluigi BUFFON"	139
4	"Javad NEKOUNAM"	137
5	"Georgios KARAGOUNIS"	134
6	"Miroslav KLOSE"	131
7	"Xavi HERNANDEZ"	131
8	"Carlos SALCIDO"	120
9	"Noel VALLADARES"	120
10	"Rafael MARQUEZ"	119

17. Find the player who plays for home club? (We got 260 records, but displaying only 6 of them)

```
//Query-17 Query to find the player who plays for  homeclub
MATCH (p:Player)-[r:PLAYS_FOR]→(c:Club)
WHERE p.plays_in_home_country = "True"
RETURN p.name AS PlayerName, c.club AS ClubName;
```

Table | Text | Code

	PlayerName	ClubName
1	"Alan PULIDO"	"Tigres UANL"
2	"Adam TAGGART"	"Newcastle United Jets FC"
3	"David VILLA"	"Atletico Madrid"
4	"Oribe PERALTA"	"Club Santos Laguna"
5	"FRED"	"Fluminense FC"
6	"Fabian SCHAER"	"FC Basel"
7		

Started streaming 260 records after 6 ms and completed after 8 ms.

18. Find the top 5 countries with most players represented?

```
// Query-18 Query to find top 5 countries with most players represented
MATCH (p:Player)-[:REPRESENTS]→(c:Country)
RETURN c.country AS Country, count(p) AS NumberOfPlayers
ORDER BY NumberOfPlayers DESC
LIMIT 5;
```

Table | Text | Code

	Country	NumberOfPlayers
1	"Australia"	23
2	"Iran"	23
3	"Brazil"	23
4	"Ivory Coast"	23
5	"Mexico"	23

Started streaming 5 records after 6 ms and completed after 7 ms.

19. Find the top 5 players who have the highest height (in cm)?

```
//Query-19 To find the top 5 players who have the highest height
MATCH (p:Player)
RETURN p.name AS PlayerName, p.height_cm AS HeightInCentimeters
ORDER BY p.height_cm DESC
LIMIT 5
```

Table | Text | Code

	PlayerName	HeightInCentimeters
1	"Fraser FORSTER"	201
2	"LEE Bumyoung"	199
3	"Thibaut COURTOIS"	198
4	"Asmir BEGOVIC"	198
5	"Jasmin FEJZIC"	198

Started streaming 5 records in less than 1 ms and completed after 10 ms.

Graph database vs Relational database:

Graph databases and traditional relational databases are two different approaches to data management, each expert in different areas and suited to different types of applications.

- Graph databases such as Neo4j are useful in handling complex and interconnected data structures which involves dynamic data.
- On the other hand, relational databases like MYSQL are more commonly used for handling structured data that has well-defined relationships and cannot handle the data which are interconnected.
- In Graph database, each node can have multiple relationships with different nodes and can be traversed quickly among the nodes. This is ideal for scenarios like social networks such as Facebook.
- In relational database, for handling different levels of relationships ,it requires join which leads to increased complexity and decreased performance.
- Graph databases are also very good at graph traversal and pathfinding tasks. They are made to efficiently carry out breadth-first and depth-first searches, which are crucial for finding connections and patterns in data. In graph databases, algorithms such as shortest path, centrality, and community discovery work well and are naturally supported.
- Relational databases, on the other hand, are limited to using self-joins and recursive queries, which are less effective and require more work to implement.
- **One of major advantage in graph database is being able to adapt to changing and dynamic data structures. It's simple to incorporate new kind of relationships without requiring major changes in schema.**
- Relational database on other hand, have fixed schema, which makes it challenging to adapt to new requirements without major changes.
- In conclusion, relational databases and graph databases each have their advantages, but graph databases are much better at handling dynamic relationships, traversing graphs quickly, and adjusting to changing data structures.

Unlocking Efficient Relationship Handling:

Graph databases outperform relational databases in handling complex relationships and dynamic data structures. Key advantages include:

• **Native Relationship Handling**: Efficiently manage relationships without foreign keys and joins.
• **Flexible Schema**: Dynamic schema changes without predefined alterations.[1]
• **Traversal and Path Analysis**: Excel at traversing relationships and analysing paths.
• **Performance in Connected Data Queries**: Superior performance in querying connected data.[2]
• **Scalability for Relationship-Heavy Data**: Graceful scaling with complex relationships.
• **Semantic Queries and Pattern Matching**: Support expressive query languages for sophisticated pattern matching.

Summary:

Graph databases are capable where the application is dependent upon the relationships and connections between data points offering a much more efficient way to query and manipulate interconnected data. Simplifies the complexity of tackling an immense amount of interconnected data or networked data as compared with relational databases. Graph databases provide a tremendous speed-up on performance, scalability and maintainability where the structure of your data is deeply related to each other, and many queries are asked about those relations in most applications that can have graph-like structures. But on an application with stable schemas and orientation towards CRUD (Create Read Update Delete) operations over simple fairly flat data models, traditional RDBMS will serve this purpose.

Practical Application using Graph Data Science:

1. **Graph Data Science for Fraud Detection in Financial Transactions:**

Graph Data Science (GDS) applies graph algorithms, machine learning, and data analysis to extract insights from graph-structured data. In fraud detection, GDS offers a holistic approach by modelling financial transactions as a graph, enabling the identification of sophisticated fraud schemes.

How GDS Works:

- **Graph Representation**: Model transactions as a graph, with nodes representing entities and edges representing relationships.[3]
- **Fraud Network Identification**: Uncover hidden relationships and organized fraud networks using graph algorithms.[4]
- **Link Analysis**: Identify influential nodes and critical pathways within the fraud network.
- **Anomaly Detection**: Identify suspicious patterns using graph-based anomaly detection algorithms.[5]
- **Behavioural Analysis**: Analyse behavioural patterns and transaction histories to identify deviations from normal behaviour.
- **Risk Scoring and Predictive Analytics**: Assign risk scores and anticipate future fraud incidents using graph-based risk scoring models and predictive analytics.

Benefits:

- Enhance fraud detection capabilities.
- Improve risk assessment accuracy.
- Protect customers and assets from fraudulent activities.
- Proactive and comprehensive approach to fraud detection.

By leveraging GDS(Graph Data Science), financial institutions can uncover hidden patterns and insights in financial transactions data, enabling more effective fraud detection and prevention.

2. **Recommendation System in E-commerce and Social Media:**

Recommendation systems are essential in every e-commerce or any social media platforms. They also improve the user experience of an app by recommending products, content or connections that are most likely to be relevant to a given person and behaviour history within the network. We can utilize GDS to create advanced recommendation engines as below:

1. Data Model:

- Nodes represent entities such as users, movies, articles, products etc.
- Edges represent interaction between nodes such as purchase,view,follow or like.

2. Use case:

a. **Recommendation engine**: Recommending products to a user in an e-commerce site, where the recommended product is sourced from the past purchase history of users and behaviour similarity across users.
b. **Social media:** Suggestions about possible articles or videos, similar news or any king of suggestion in base to what a similar user have seen.
c. **Connection Recommendations**: In social media platforms, GDS suggests new friends by analysing mutual connections, interests etc.

3. Graph Algorithms:

a. **Graph based collaborative filtering:** Here, we connect users and items as nodes and edges will be created between user item if connected. The edges will indicate purchases or ratings. To recommend items, algorithms such as personalized PageRank can follow equivalent approaches to those outlined above by finding similar users or directly related items
b. **Path Finding Algorithms**: Used to find out the shortest path or could be various ones taken by a user in any navigation application/game, displaying relevant content, offers etc. along the way.

4. Example:

a. **Build a Graph**: Construct a graph where users and products become the nodes. The relationship are purchases, views and cart additions.
b. **Node Similarity**: Calculate the similarity score of two users based upon their interaction with products.
c. **Predict Links**: By using machine learning models, predict new links between users and products that are not yet present but seem meanly given the similarity scores of users and product attributes.
d. **Generate Recommendations**: Recommend products to user based on the links that are predicted, now in this case it will be only known data of product available and not new upcoming product. These suggestions are inventively refreshed as a client shows more product.

References:

[1] Chang, F., et al. (2008). Bigtable: A Distributed Storage System for Structured Data. ACM Transactions on Computer Systems (TOCS), 26(2), 1-26.

[2] Angles, R., & Gutierrez, C. (2008). Survey of Graph Database Models. ACM Computing Surveys (CSUR), 40(1), 1-39.

[3] Akoglu, L., et al. (2015). Graph-based Fraud Detection in Financial Transactions. In Proceedings of the 2015 IEEE International Conference on Data Mining (ICDM) (pp. 1-10).

[4] Sathe, S., et al. (2019). Graph Representation Learning for Fraud Detection. In Proceedings of the 2019 IEEE International Conference on Big Data (BigData) (pp. 1-8).

[5] Chakrabarti, D., et al. (2014). Anomaly Detection in Graphs. In Proceedings of the 2014 IEEE International Conference on Data Mining (ICDM) (pp. 1-10).

[6] Used Chatgpt for understanding how to create different csv files for nodes and relationships using primary keys.

[7] Lecture Notes-8 for definition of graph database.