# Graph Database for Esports Recommendation Engine using Neo4j

By

Mr. Rutik Kothwala,

Mr. Somil Saparia

Mr. Nagul Shaik

Guided By

FNU, Kaleemunnisa

Final Project in Scalable Database(CS673)

In

Seidenberg College of Computer Science and Information Science

Pace University

# TABLE OF CONTENT

# PEOBLEM DEFINITION

Representation and Analysis of E-Sports Data using a Graph Database

**Background:**

The e-sports industry has been experiencing rapid growth, with a vast amount of data generated from players, teams, games, genres, and their interrelationships. Analyzing and extracting insights from this complex data landscape is crucial for various stakeholders, including e-sports organizations, researchers, and enthusiasts. However, traditional relational databases struggle to efficiently represent and query the interconnected nature of e-sports data, hindering effective data analysis and decision-making processes.

**Problem Statement:**

The current challenge lies in representing and analyzing e-sports data in a comprehensive and efficient manner. The use of a graph database, such as Neo4j, presents a promising solution. However, there is a need to develop a systematic approach to construct and manage the graph database schema, load data from various sources, and establish relationships between different entities to enable meaningful analysis and exploration of e-sports data.

**Scope:**

The problem focuses on the development and implementation of a graph database schema using Neo4j to represent and analyze e-sports data.

**Benefits:**

By addressing the problem and achieving the objectives, the project will provide the following benefits:Enable efficient representation and analysis of e-sports data. Facilitate exploration of relationships between players, countries, games, genres, teams, and continents.Support data-driven decision making for e-sports organizations, researchers, and enthusiasts.Provide a foundation for further research and analysis in the e-sports .

## ABOUT GRAPH DATABASE

A graph database is a database that organizes data using nodes and relationships, representing complex relationships between entities. Key points about graph databases include:

1. Graph Structure: Data is organized as nodes and relationships, with nodes representing entities and relationships representing connections between entities.

2. Relationships as First-Class Citizens: Relationships are explicitly defined and can have properties, enabling efficient traversal of data.

3. Flexible Schema: Graph databases offer a flexible schema, allowing data relationships to evolve without predefined structures.

4. Efficient Relationship Queries: Graph databases excel at querying and traversing relationships, making them ideal for complex queries and pattern matching.

5. High Performance and Scalability: Graph databases are designed for high-performance data access, handling large datasets and scaling horizontally.

6. Use Cases: Graph databases find applications in social networks, recommendation systems, fraud detection, knowledge graphs, and more.

7. Graph Query Languages: Specialized query languages like Cypher and Gremlin enable intuitive querying of graph databases.

8. Integration Capabilities: Graph databases can integrate with other technologies and data sources, complementing existing databases.

In summary, graph databases provide a powerful and flexible approach to represent and analyze complex relationships in data, enabling efficient data exploration in various domains.

## ABOUT Neo4j

1. Neo4j is a highly scalable and performance-oriented graph database that allows you to efficiently store, retrieve, and analyze interconnected data.

2. With Neo4j, you can easily represent complex relationships between entities using nodes and relationships, enabling a flexible and intuitive data model.

3. Neo4j provides a powerful query language called Cypher, which allows you to express graph patterns and relationships in a concise and readable syntax, facilitating complex queries and data exploration.

4. Neo4j's graph algorithms and indexing techniques enable efficient traversal and analysis of relationships, making it an ideal choice for applications such as recommendation systems, social networks, fraud detection, and knowledge graphs.

# TRANSFORMATION OF DATA

Data transformation in Neo4j involves manipulating and reorganizing data stored in a graph database to derive new insights or prepare it for further analysis. Here are some common operations and techniques used for data transformation in Neo4j:

1. Creating Nodes and Relationships: Use the `CREATE` or `MERGE` statements to create nodes and relationships based on the data you have. Nodes represent entities, while relationships represent connections between entities.

2. Updating Node Properties: Use the `SET` statement to update properties of existing nodes or relationships. This allows you to add, modify, or delete property values.

3. Filtering Data: Use the `WHERE` clause to filter data based on specific conditions. This allows you to narrow down your data set to only include relevant entities or relationships.

4. Aggregating Data: Utilize aggregate functions such as `COUNT()`, `SUM()`, `AVG()`, `MIN()`, or `MAX()` to perform calculations on groups of nodes or relationships. This enables you to derive statistics or summarize data.

5. Joining Nodes and Relationships: Use the `MATCH` statement to find specific patterns in your graph and establish connections between related nodes or relationships. This allows you to combine data from different parts of your graph.

6. Sorting Data: Use the `ORDER BY` clause to sort your query results based on specific properties. This helps in organizing the output in a desired order, such as ascending or descending.

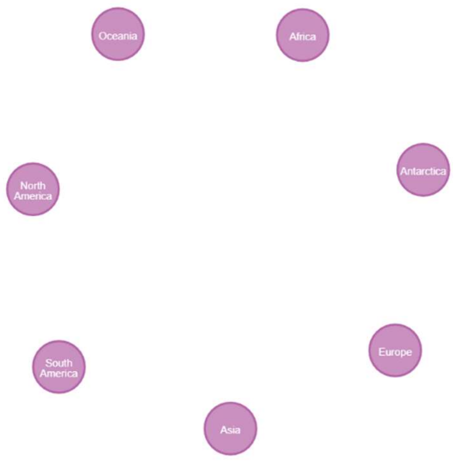
7. Removing Duplicates: Use the `DISTINCT` keyword to eliminate duplicate nodes or relationships from your query results. This ensures that each entity appears only once in the output.

8. Importing and Exporting Data: Neo4j provides tools and utilities to import data from external sources into the graph database, such as CSV files.

## MODELING

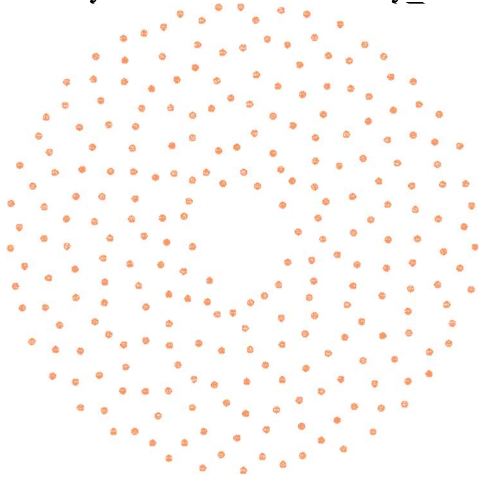
## Creating Continent nodes manually:

```
CREATE (node:continent {continentCode: 'AS', continentName: 'Asia'})
CREATE (node:continent {continentCode: 'EU', continentName: 'Europe'})
CREATE (node:continent {continentCode: 'AN', continentName: 'Antarctica'})
CREATE (node:continent {continentCode: 'AF', continentName: 'Africa'})
CREATE (node:continent {continentCode: 'OC', continentName: 'Oceania'})
CREATE (node:continent {continentCode: 'NA', continentName: 'North America'})
CREATE (node:continent {continentCode: 'SA', continentName: 'South America'})
```



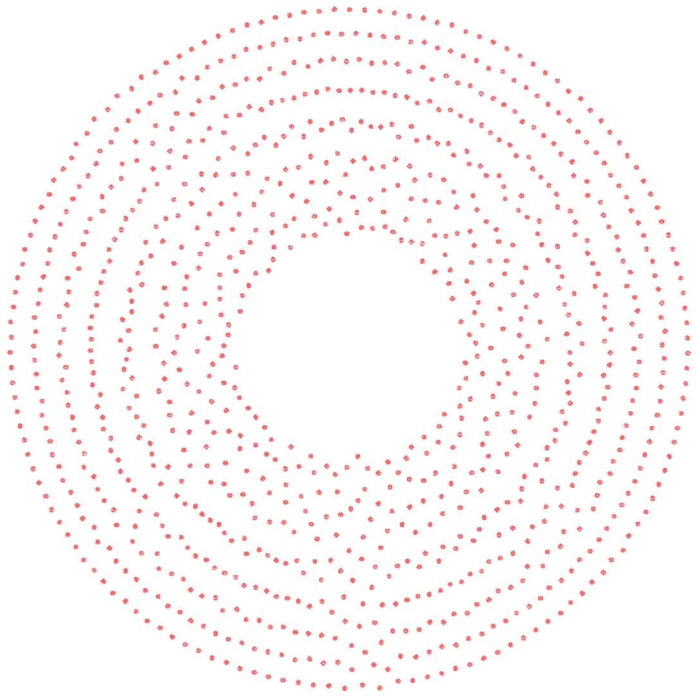
## Creating Country nodes using .CSV file

```
LOAD CSV WITH HEADERS FROM 'file:///country-and-continent-codes-list.csv' AS row
MERGE (c:country {countryName: row.Country_Name})
ON CREATE SET c.twoLetterCode = row.Two_Letter_Country_Code, c.threeLetterCode =
row.Three_Letter_Country_Code, c.countryNumber = row.Country_Number.
```



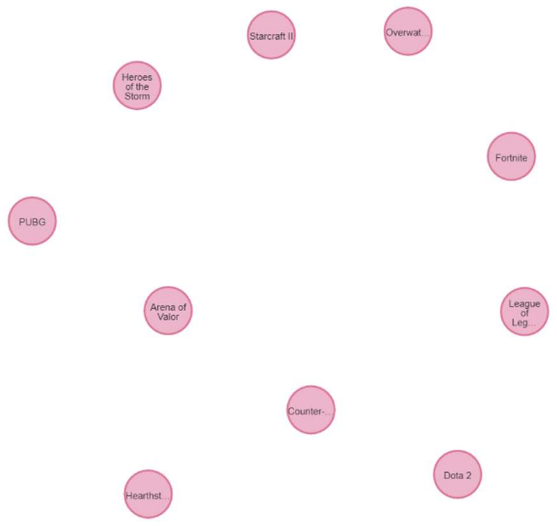
## Creating player nodes using .CSV file

```
LOAD CSV WITH HEADERS FROM 'file:///highest_earning_players.csv' AS row
MERGE (p:players {playerID: row.PlayerId})
ON CREATE SET p.firstName = row.NameFirst, p.lastName = row.NameLast, p.playerHandle =
row.CurrentHandle, p.totalPrizeWon = row.TotalUSDPrize
```



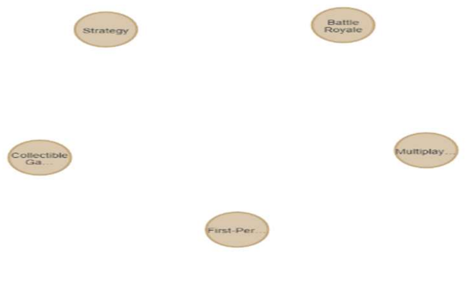
## Creating games nodes manually

```
MERGE (g:games {gameName: 'Counter-Strike: Global Offensive'})
MERGE (g:games {gameName: 'Dota 2'})
MERGE (g:games {gameName: 'League of Legends'})
MERGE (g:games {gameName: 'Fortnite'})
MERGE (g:games {gameName: 'Overwatch'})
MERGE (g:games {gameName: 'Starcraft II'})
MERGE (g:games {gameName: 'Heroes of the Storm'})
MERGE (g:games {gameName: 'PUBG'})
MERGE (g:games {gameName: 'Arena of Valor'})
MERGE (g:games {gameName: 'Hearthstone'})
```



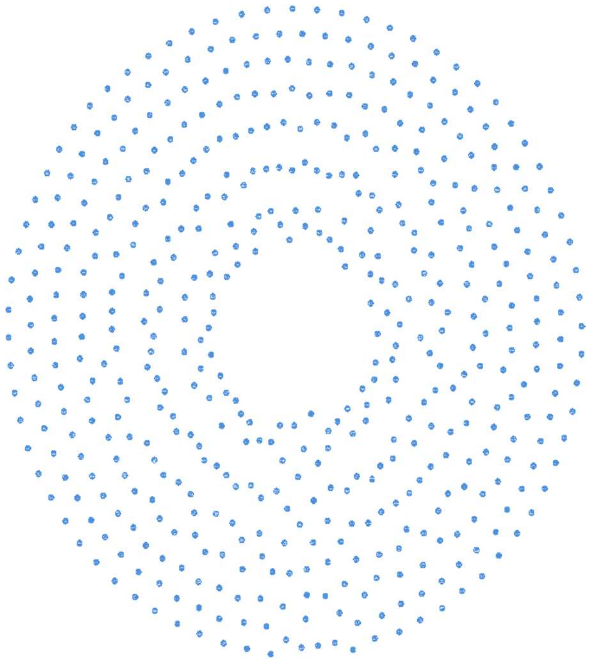
## Creating genre nodes manually

```
MERGE (g:genre {genreName: 'Multiplayer Online Battle Arena'})
MERGE (g:genre {genreName: 'Battle Royale'})
MERGE (g:genre {genreName: 'Strategy'})
MERGE (g:genre {genreName: 'Collectible Card Game'})
```



## Creating teams nodes using .CSV file

```
LOAD CSV WITH HEADERS FROM 'file:///highest_earning_teams.csv' AS row
MERGE (t:teams {teamID: row.TeamId})
ON CREATE SET t.teamName = row.TeamName, t.totalPrizeWon = row.TotalUSDPrize,
t.tournamentsPlayed = row.TotalTournaments
```

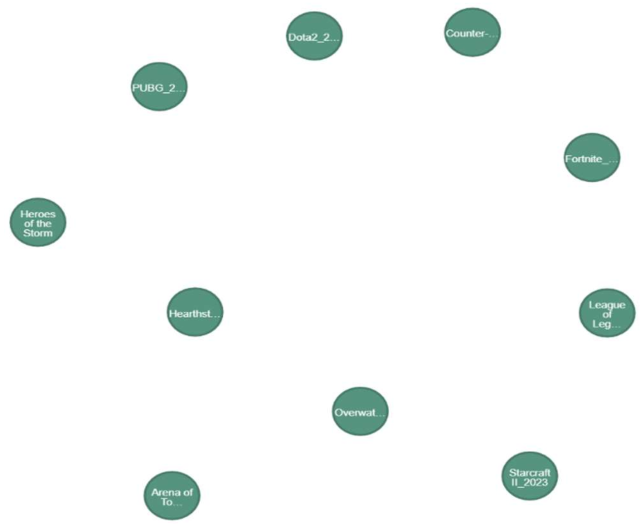


## Creating platform nodes manually

```
MERGE (p1:platform {platformName: 'PC'})
MERGE (p2:platform {platformName: 'Mobile'})
MERGE (p3:platform {platformName: 'Console'})
```

## Creating tournaments node manually

```
MERGE (t1:tournaments {tournamentName: 'Overwatch_2023'})
ON CREATE SET t1.startDate = date('2023-05-01'), t1.endDate = date('2023-05-07')
```

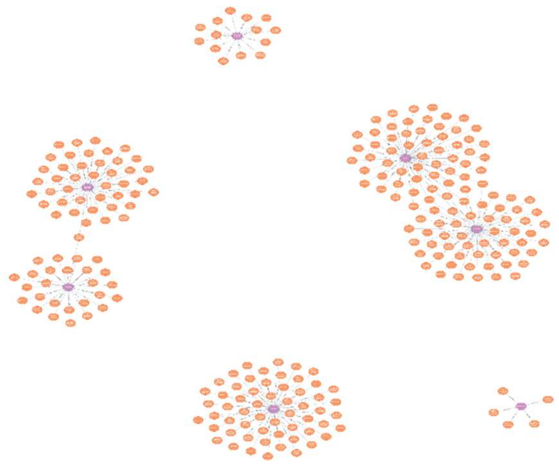


## Relations:

1)Relations between country and continent
Relation is labeled as "PART_OF

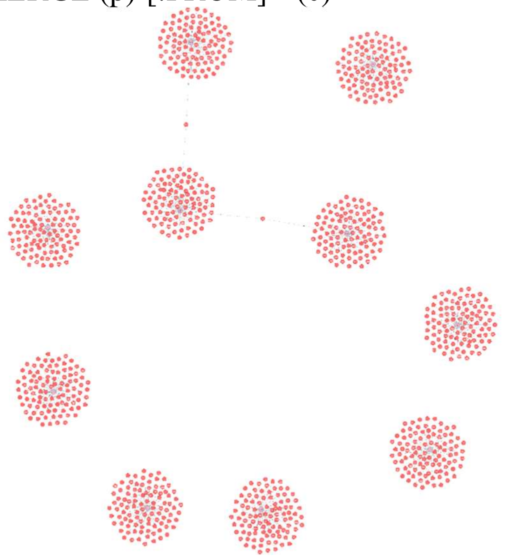
Code of the relation:

```
MATCH (con:continent {continentName: row.Continent_Name})
MATCH (cou:country {countryName: row.Country_Name})
CREATE (cou)-[:PART_OF]->(con)
```



2. Relation between players and country
   - ▶ The label use for the relationship between the players and the country is "FROM"
   - ▶ CODE OF THE RELATION:

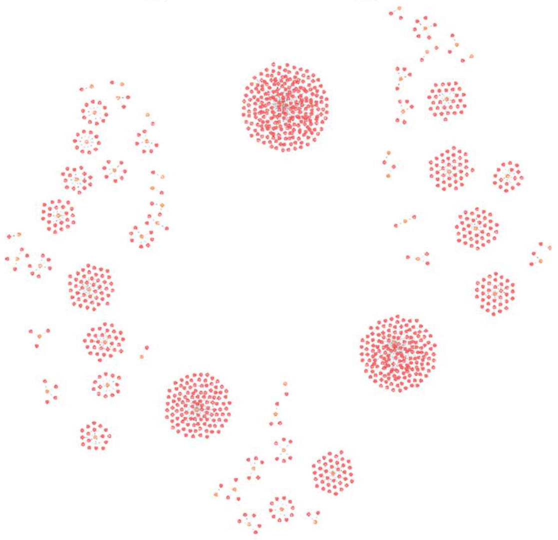
```
MATCH (c:country {twoLetterCode: TOUPPER(row.CountryCode)})
MATCH (p:players {playerID: row.PlayerId})
MERGE (p)-[:FROM]->(c)
```



3.Relation between players and games:

- ▶ The label use for the relationship between players and games is “PLAYS”.
- ▶ CODE OF THE RELATION:

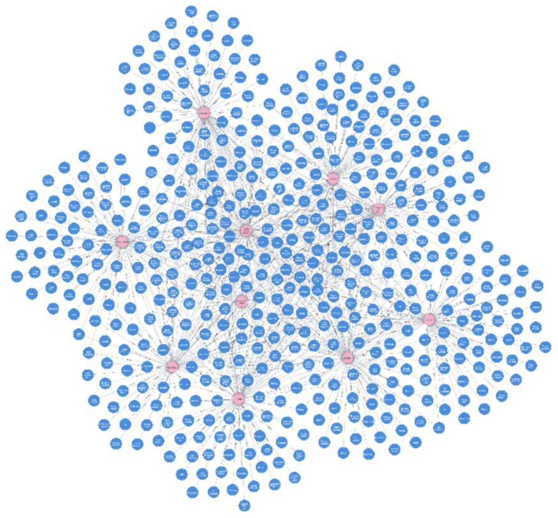
```
MATCH (p:players {playerID: row.PlayerId})
MATCH (g:games {gameName}
CREATE (p)-[:PLAYS]->(g)
```



4. Relationship between Players and genre:

- ▶ The label use for the relationship between players and genre is “PLAYS_IN”
- ▶ The code of the relationship is:

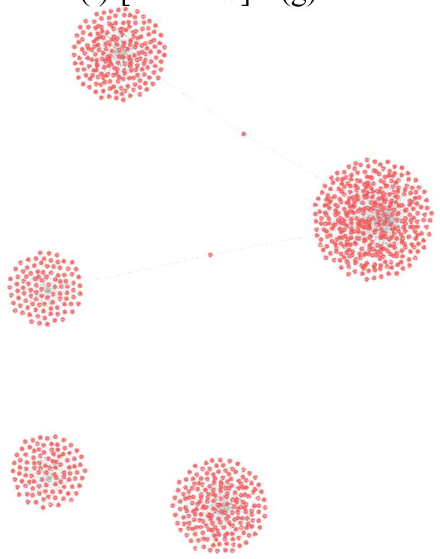
```
MATCH (p:players {playerID: row.PlayerId})
MATCH (gen:genre {genreName: row.Genre})
CREATE (p)-[:PLAYS_IN]->(gen)
```



5. Relationship between the teams and games:

- ▶ The label use for the relationship between teams and games is "PLAYS"
- ▶ The code of the relationship is :

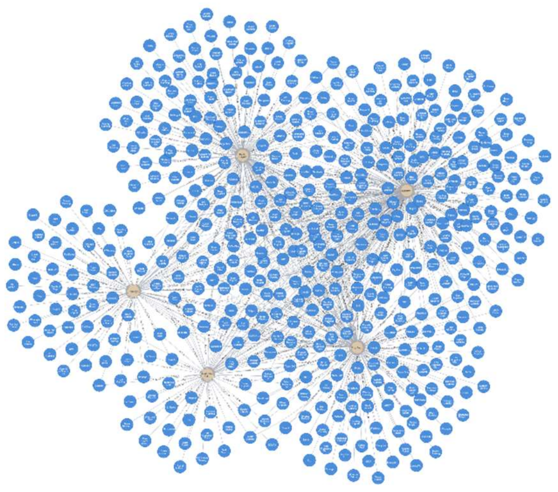
```
MATCH (t:teams {teamID: row.TeamId})
MATCH (g:games {gameName: row.Game})
CREATE (t)-[:PLAYS]->(g)
```



6. Relationship between Teams and genre:

- ▶ The label use for the relationship between teams and genre "PLAYS_IN"
- ▶ The code of the relationship is :

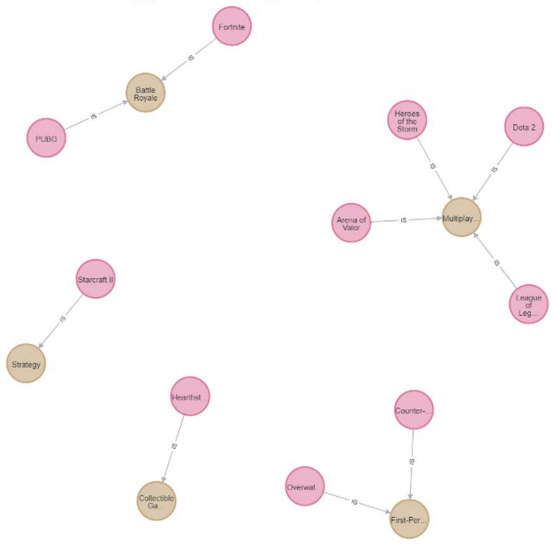
```
MATCH (t:teams {teamID: row.TeamId})
MATCH (gen:genre {genreName: row.Genre})
CREATE (t)-[:PLAYS_IN]->(gen)
```



7. Relationship between games and genre:

- ▶ The label use for the relationship between games and genre: "IS"
- ▶ The code of the relationship is :

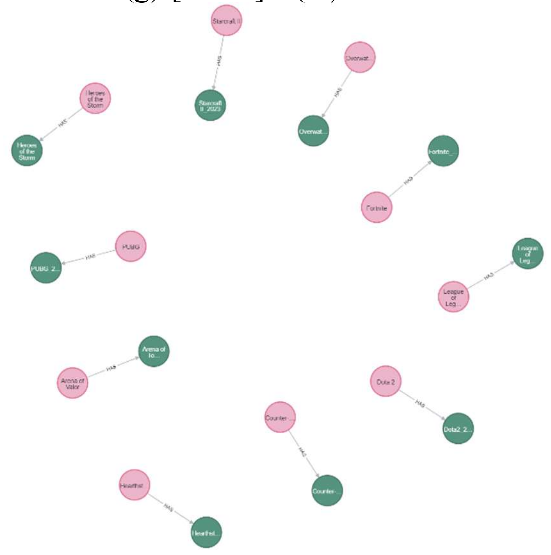
```
MATCH (g:games {gameName: row.Game})
MATCH (gen:genre {genreName: row.Genre})
MERGE (g)-[:IS]->(gen)
```



8. Relationship between games and tournaments:

- ▶ The label use for the relationship between games and tournments is "HAS"
- ▶ The code for the relationship is:

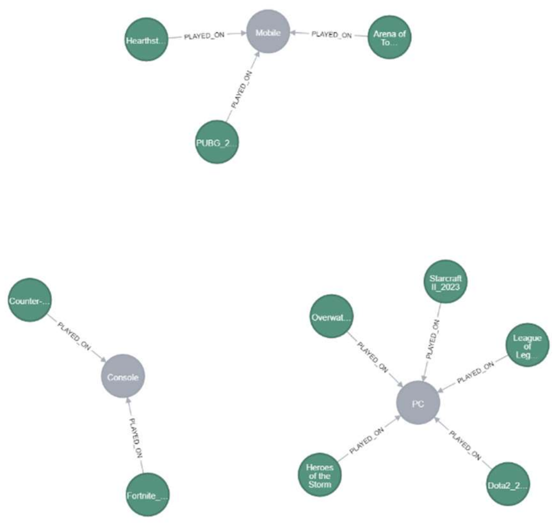
```
MERGE (t1:tournaments {tournamentName: 'Overwatch_2023'})
ON CREATE SET t1.startDate = date('2023-05-01'), t1.endDate = date('2023-05-07')
WITH t1
MATCH (g:games {gameName: 'Overwatch'})
MERGE (g)-[:HAS]->(t1)
```



9. Relationship between tournaments and platforms:

- The label use for the relationship between tournments and platform is " PLAYED_ON"
- The code of the relationship is:

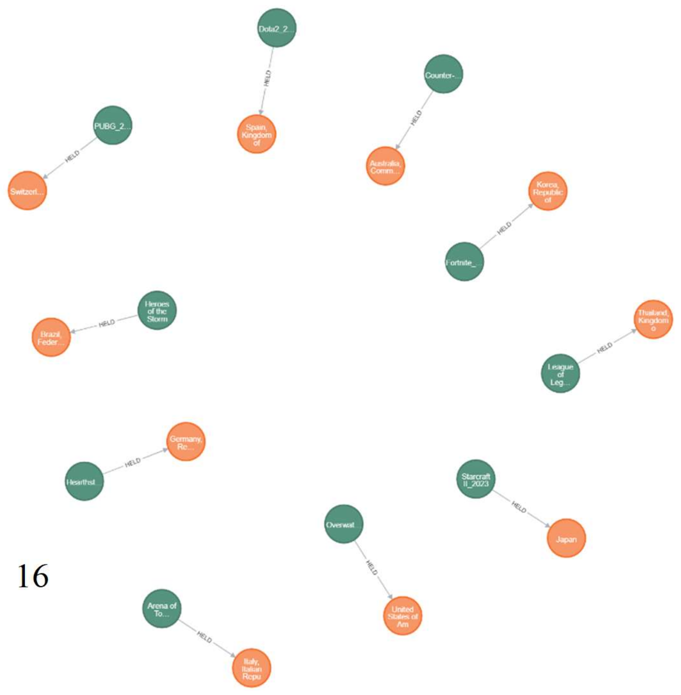
```
MATCH (t:tournaments {tournamentName: 'Overwatch_2023'}), (p:platform {platformName: 'PC'})
MERGE (t)-[:PLAYED_ON]->(p)
```



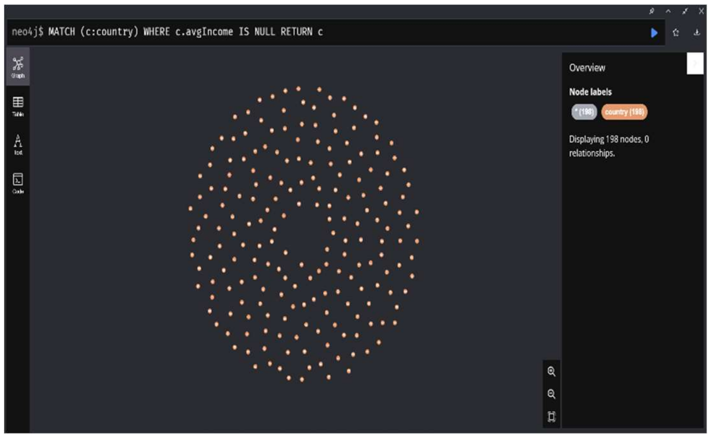
10. Relationship between Tournaments and countries:

- The label use for the relationship between tournament and countries is "HELD"
- The code of the relationship tournament and countries:

```
MERGE (t1:tournaments {tournamentName: 'Overwatch_2023'})
ON CREATE SET t1.startDate = date('2023-05-01'), t1.endDate = date('2023-05-07')
WITH t1
MATCH (c:country {countryName: 'United States of America'})
MERGE (t1)-[:HELD]->(c)
```



## NULL NODE COMPARISION:



```
neo4j$ MATCH (c:country) WHERE c.avgIncome IS NULL RETURN c
```

Overview

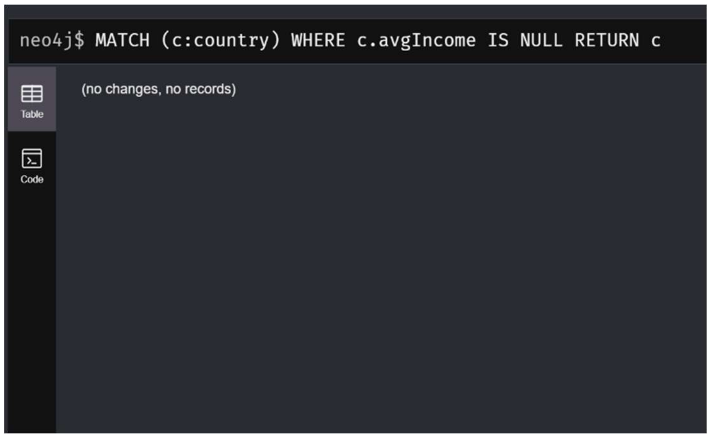
Node labels

* (198) country (198)

Displaying 198 nodes, 0 relationships.

VALUES IN AVERAGE INCOME OF COUNTRIES, BEFORE

AFTER

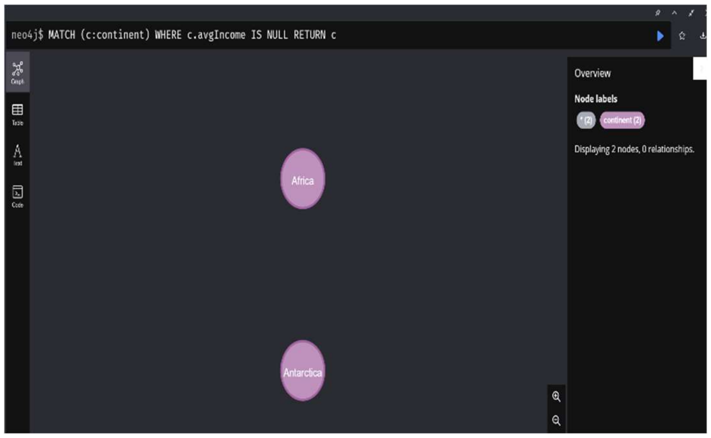


```
neo4j$ MATCH (c:country) WHERE c.avgIncome IS NULL RETURN c
```

(no changes, no records)

## NULL VALUES IN AVERAGE INCOME OF CONTINENTS

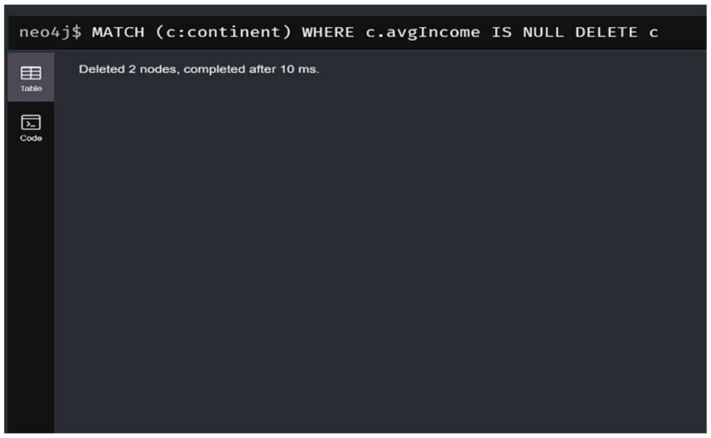
BEFORE



```
neo4j$ MATCH (c:continent) WHERE c.avgIncome IS NULL RETURN c
```

Displaying 2 nodes, 0 relationships.

AFTER



```
neo4j$ MATCH (c:continent) WHERE c.avgIncome IS NULL DELETE c
```

Deleted 2 nodes, completed after 10 ms.

## Queries performed to remove NULL value nodes are:

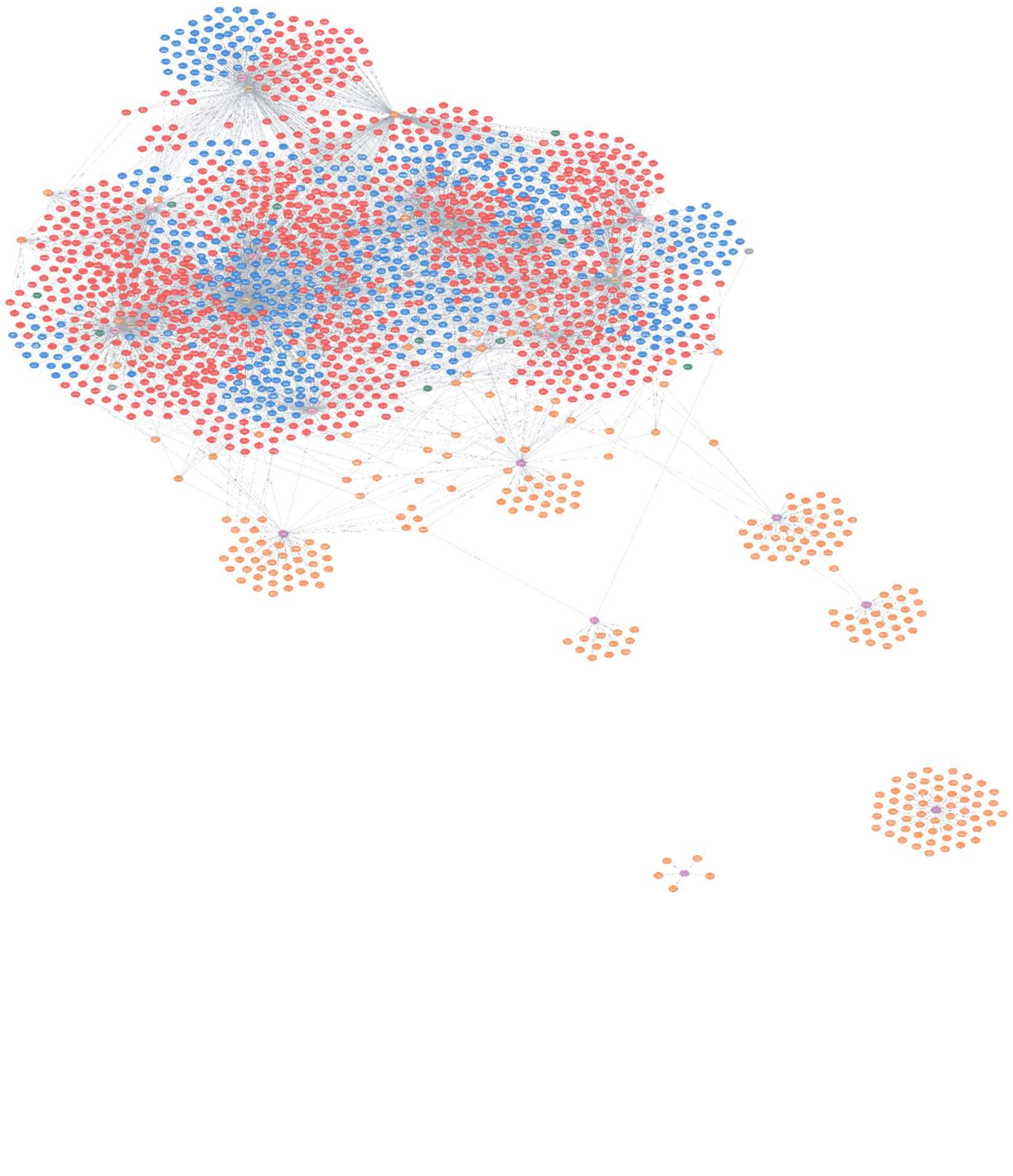
- ▶ Query 1 (to remove countries):

```
MATCH (c:country)-[r:PART_OF]->(ct:continent)
WHERE c.avgIncome IS NULL
DELETE r, c
```

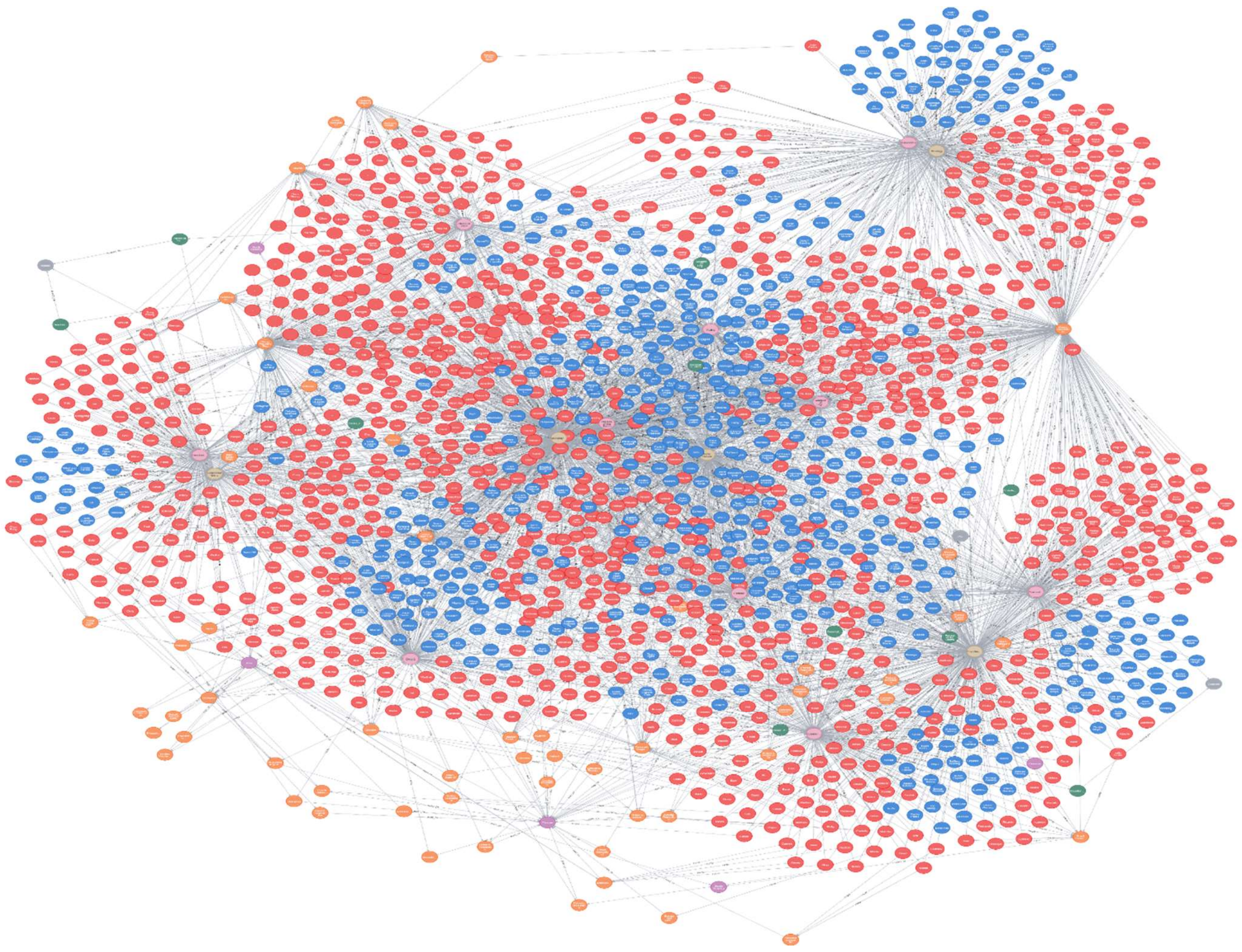
- ▶ Query 2 (to remove continents):

```
MATCH (c:continent)
WHERE c.avgIncome IS NULL
DELETE c
```

## FINAL GRAPH BEFORE REMOVING NULL VALUES



## FINAL GRAPH AFTER REMOVING NULL VALUES

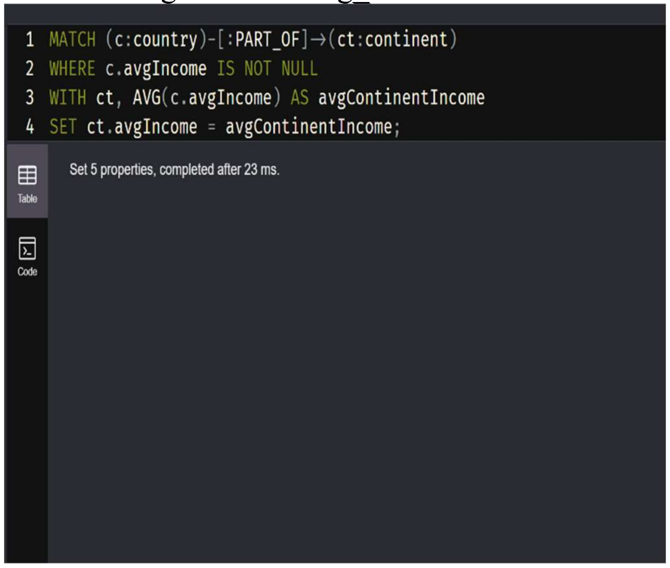


## AGGRIGATION

1. Aggrigation of average income of a players of a country

- ▶ Use “AVG” to find the average.
- ▶ Code for the aggregation:

```
MATCH (p:players)-[: FROM]→ (c: country)
WITH C, AVG(toInteger(p.totalPrizeWon)) AS avg_income
SET c.avgIncome = avg_income
```



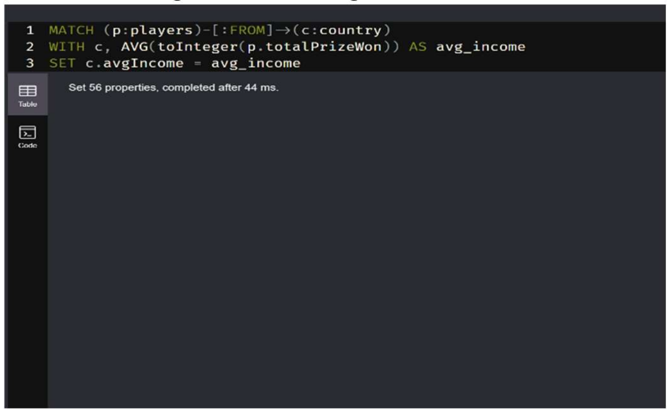
```
MATCH (c:country)-[:PART_OF]→(ct:continent)
WHERE c.avgIncome IS NOT NULL
WITH ct, AVG(c.avgIncome) AS avgContinentIncome
SET ct.avgIncome = avgContinentIncome;
```

Set 5 properties, completed after 23 ms.

2. Aggrigation of average income of a country and continent

- ▶ Use “AVG” to find the average
- ▶ Code for the aggregation:

```
MATCH (c:country)-[:PART_OF]→(ct: continent)
WHERE c.avgIncome IS NOT NULL
WITH ct, AVG(c.avgIncome) AS avgContinentIncome
SET ct.avgIncome = avgContinent Income;
```



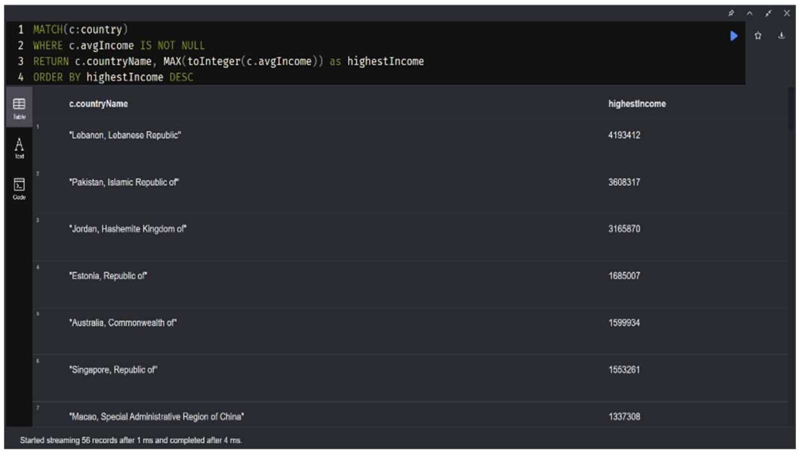
```
MATCH (p:players)-[:FROM]→(c:country)
WITH c, AVG(toInteger(p.totalPrizeWon)) AS avg_income
SET c.avgIncome = avg_income
```

Set 56 properties, completed after 44 ms.

3. Arranging the continents based on their incomes using the “MIN()” function in ascending order:

- Code of aggrigaion:

```
MATCH (ct:continent)
WHERE ct.avgIncome IS NOT NULL
RETURN ct.continentName, MIN(toInteger (ct. avgIncome)) as                    lowest Income
ORDER BY LowestIncome ASC
```



```
MATCH(c:country)
WHERE c.avgIncome IS NOT NULL
RETURN c.countryName, MAX(toInteger(c.avgIncome)) as highestIncome
ORDER BY highestIncome DESC
```

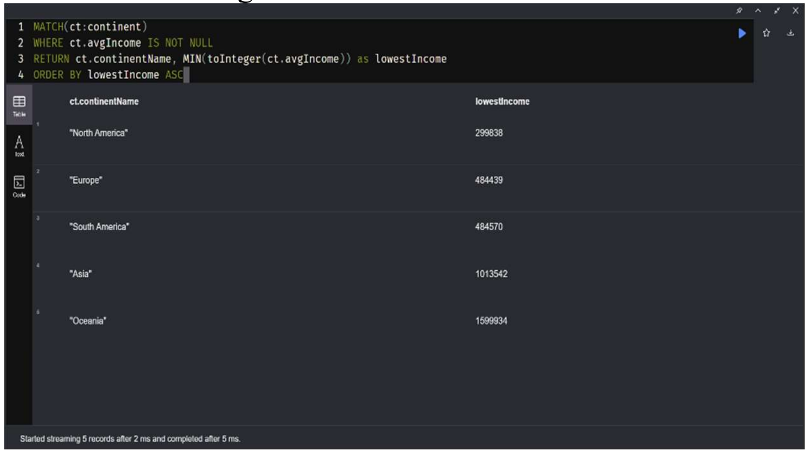
| | c.countryName | highestIncome |
|---|---|---|
| 1 | "Lebanon, Lebanese Republic" | 4193412 |
| 2 | "Pakistan, Islamic Republic of" | 3608317 |
| 3 | "Jordan, Hashemite Kingdom of" | 3165870 |
| 4 | "Estonia, Republic of" | 1685007 |
| 5 | "Australia, Commonwealth of" | 1599934 |
| 6 | "Singapore, Republic of" | 1553261 |
| 7 | "Macao, Special Administrative Region of China" | 1337308 |

Started streaming 56 records after 1 ms and completed after 4 ms.

4. Arranging the countries based on their incomes using the “Max()” function in descending order:

- Code of aggrigaion:

```
MATCH (c:country)
WHERE c.avgIncome IS NOT NULL
RETURN c.countryName, Max(toInteger (ct. avgIncome)) as              highest Income
ORDER BY highestincome DESC
```



```
MATCH(ct:continent)
WHERE ct.avgIncome IS NOT NULL
RETURN ct.continentName, MIN(toInteger(ct.avgIncome)) as lowestIncome
ORDER BY lowestIncome ASC
```

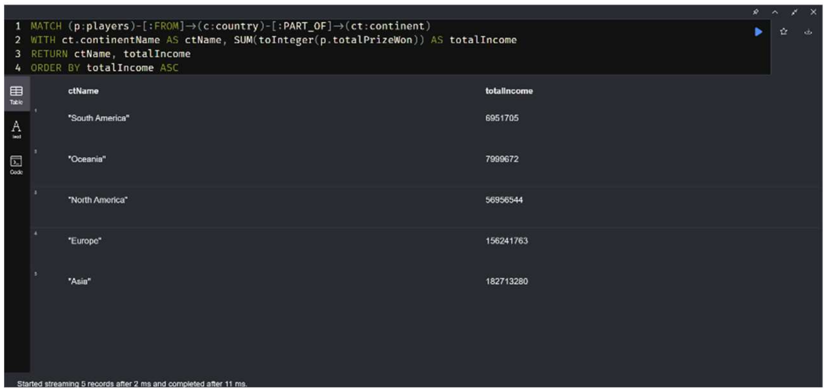
| | ct.continentName | lowestIncome |
|---|---|---|
| 1 | "North America" | 299838 |
| 2 | "Europe" | 484439 |
| 3 | "South America" | 484570 |
| 4 | "Asia" | 1013542 |
| 5 | "Oceania" | 1599934 |

Started streaming 5 records after 2 ms and completed after 5 ms.

5. Arranging the continents based on their total prizes won in ascending order:

- Code of aggregation:

```
MATCH (p:players)-[:FROM](c: country)-[:PART_OF](ct:continent)
WITH ct.continentName AS ctName, SUM(toInteger (p.totalPrizeWon)) AS    totalIncome
RETURN ctName, totalIncome
ORDER BY totalIncome ASC
```



```
MATCH (p:players)-[:FROM]→(c:country)-[:PART_OF]→(ct:continent)
WITH ct.continentName AS ctName, SUM(toInteger(p.totalPrizeWon)) AS totalIncome
RETURN ctName, totalIncome
ORDER BY totalIncome ASC
```

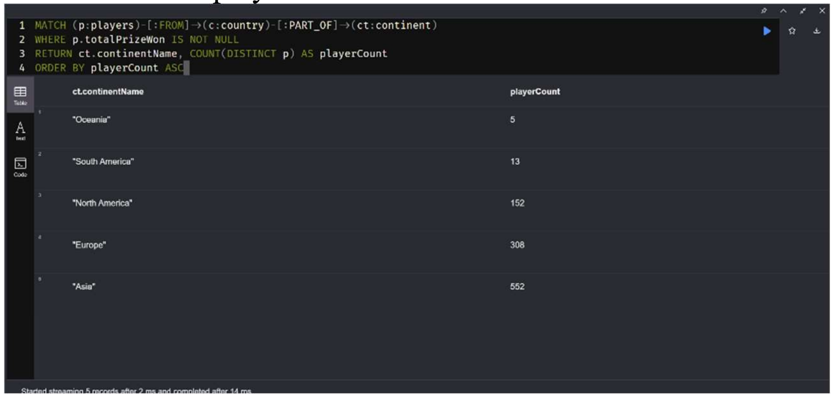
| ctName | totalIncome |
|---|---|
| "South America" | 6951705 |
| "Oceania" | 7999672 |
| "North America" | 56956544 |
| "Europe" | 156241763 |
| "Asia" | 182713280 |

Started streaming 5 records after 2 ms and completed after 11 ms.

6. How many players for a continent:

- Code of aggregation:

```
MATCH (p:players)-[:FROM](c: country)-[:PART_OF](ct:continent)
WHERE p.totalPrizeWon IS NOT NULL
RETURN ct.continentName, COUNT(DISTINCT p) AS playerCount
ORDER BY playerCount ASC
```



```
MATCH (p:players)-[:FROM]→(c:country)-[:PART_OF]→(ct:continent)
WHERE p.totalPrizeWon IS NOT NULL
RETURN ct.continentName, COUNT(DISTINCT p) AS playerCount
ORDER BY playerCount ASC
```

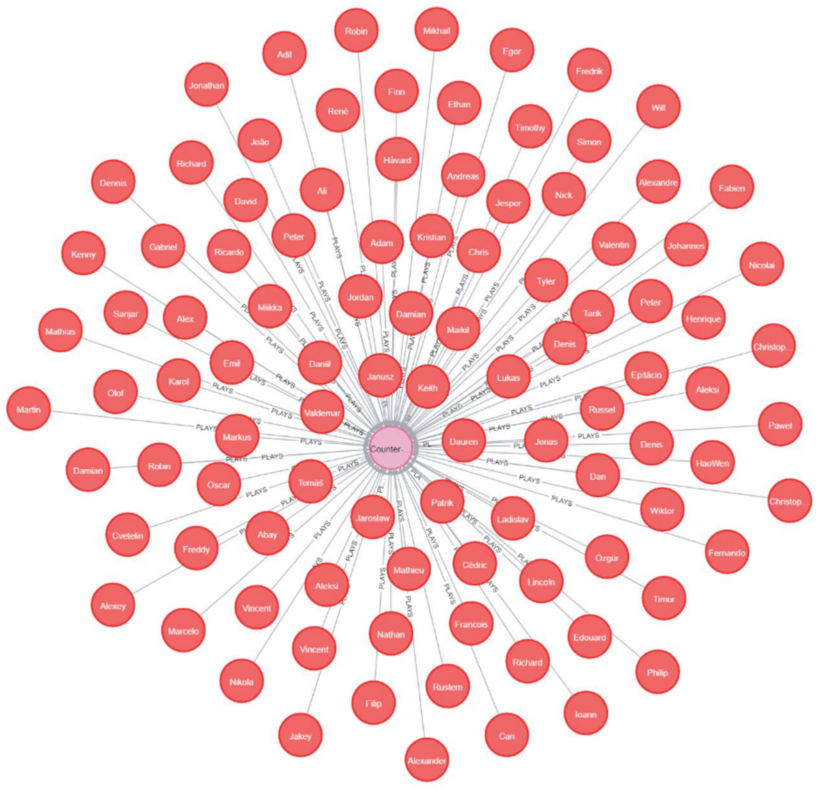
| ct.continentName | playerCount |
|---|---|
| "Oceania" | 5 |
| "South America" | 13 |
| "North America" | 152 |
| "Europe" | 308 |
| "Asia" | 552 |

Started streaming 5 records after 2 ms and completed after 14 ms.

## QUERYING THE DATABASE

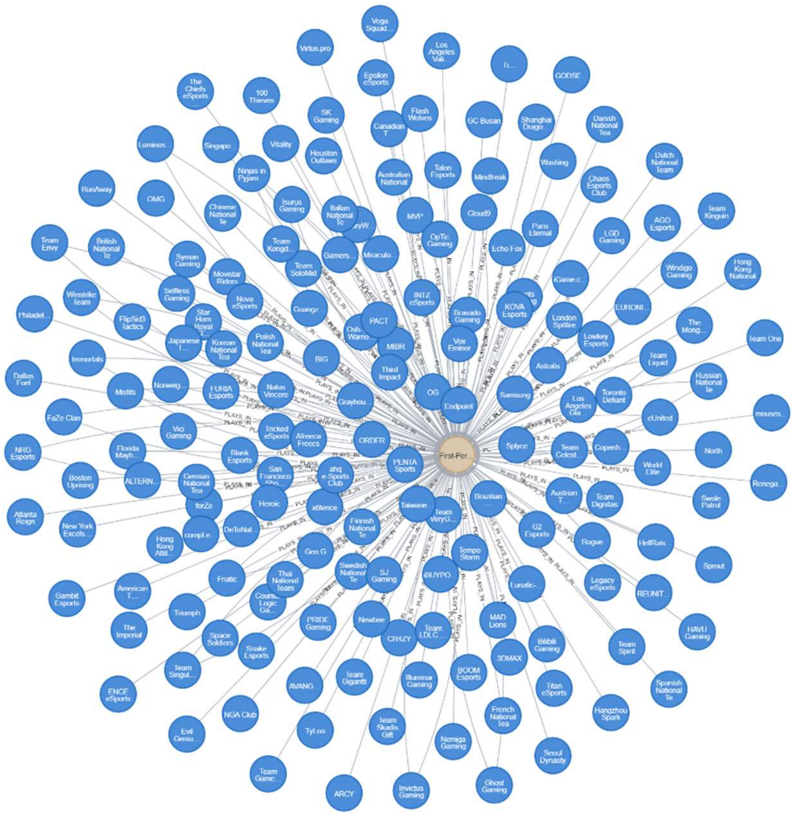
1)Get all the players of a specific game called “Counter-Strike: Global Offensive”

- **Query:**
- MATCH (p:players)-[:PLAYS]->(g:games {gameName: 'Counter-Strike: Global Offensive'})
- RETURN p, g



2) Get all the teams of a specific genre called “First-Person Shooter”

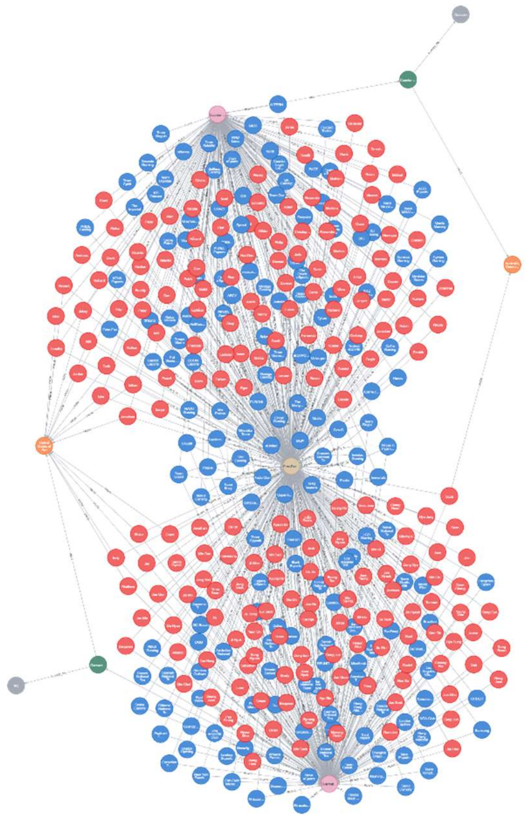
- **Query:**
- MATCH (t:teams)-[:PLAYS_IN]->(g:genre {genreName: 'First-Person Shooter'})
- RETURN t, g



3) Get all the players, teams, games, tournaments, tournament_platform and tournament_location of a specific genre called “First-Person Shooter”

**Query:**

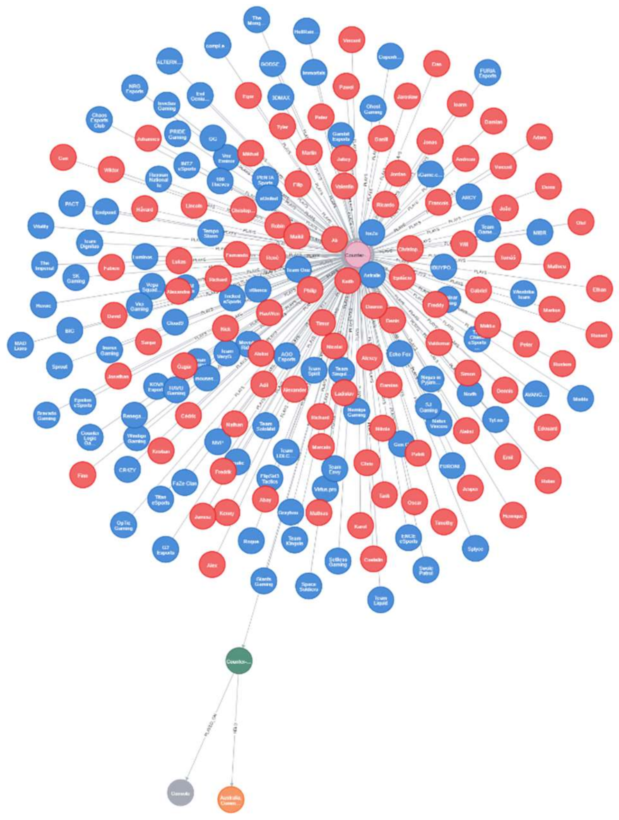
```
MATCH (p:players)-[:PLAYS_IN]->(gen:genre {genreName: 'First-Person Shooter'})<-[:IS]-(g:games)-
[:HAS]->(tour:tournaments)-[:PLAYED_ON]->(plat:platform)
MATCH (tour)-[:HELD]->(c:country)
MATCH (g)<-[:PLAYS]-(t:teams)-[:PLAYS_IN]->(gen)
RETURN p, gen, g, tour, plat, c, t
```



4)Get all the players, teams, tournaments, tournament_platform and tournament_location for a specific game called “Counter-Strike: Global Offensive”

▶ **Query:**

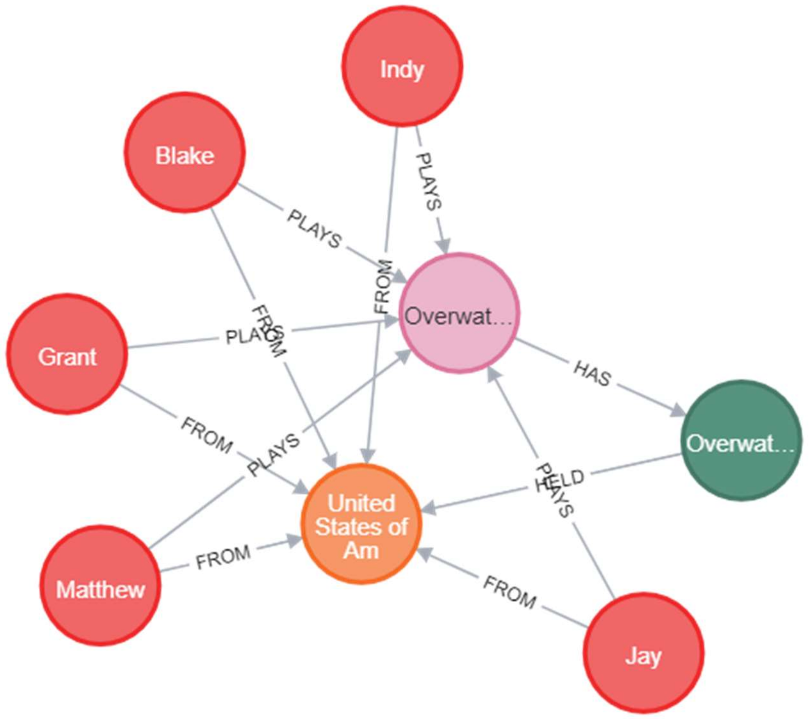
MATCH (p:players)-[:PLAYS]->(g:games {gameName: 'Counter-Strike: Global Offensive'})-[:HAS]->(tour:tournaments)-[PLAYED_ON]->(plat:platform)
MATCH (tour)-[:HELD]->(c:country)
MATCH (t:teams)-[:PLAYS]->(g)
RETURN p, g, tour, plat, c, t



5) Get all the players who play games whose tournaments are going to held in a specific country called "United States of America"

▶ **Query:**

```
MATCH (c:country {countryName: 'United States of America'})<-[:FROM]-(p:players)-[:PLAYS]->(g:games)
MATCH (c)<-[:HELD]-(t:tournaments)<-[:HAS]-(g)
RETURN c, p, t, g
```



## Conclusion

- In conclusion, our project successfully addressed the challenges of representing and analyzing e-sports data by leveraging a graph database, specifically Neo4j. We developed a systematic approach to construct and manage the graph database schema, effectively loaded data from diverse sources, and established meaningful relationships between entities. This enabled us to gain valuable insights and facilitate exploration of the interconnected nature of e-sports data. The use of a graph database proved to be a powerful and efficient solution for extracting insights from the complex e-sports data landscape, benefiting various organizations , stakeholders in the industry, also normal person who are fascinated about Esports.

**Thank You**