2024-12-02

DATABASES 2

LAB 9 - GRAPH DATABASES WITH NEO4J

Paulina Czarnota C21365726

EXERCISE 1

INTRODUCTION

This exercise demonstrates the creation, management, and querying of a social network graph using Neo4j. The dataset consists of Person nodes connected by FRIEND_OF relationships. The lab includes creating nodes, establishing relationships, running queries, and analysing data using the Cypher Query Language (CQL).

PART 1: GRAPH CREATION (NODES AND RELATIONSHIPS)

1. Creating Person Nodes

```
CREATE
(e1:Person { name: "Mary", country: "Sweden", age: 29, sport: "Hockey" }),
(e2:Person { name: "Emily", country: "Ireland", age: 19, sport: "Football" }),
(e3:Person { name: "Mark", country: "Sweden", age: 23, sport: "Rugby" }),
(e4:Person { name: "Joe", country: "Sweden", age: 32, sport: "Hockey" }),
(e5:Person { name: "John", country: "Ireland", age: 31, sport: "Football" }),
(e6:Person { name: "Peter", country: "France", age: 23, sport: "Rugby" }),
(e7:Person { name: "Paul", country: "Sweden", age: 25, sport: "Hockey" }),
(e8:Person { name: "Kevin", country: "Ireland", age: 17, sport: "Football" }),
(e9:Person { name: "Patrick", country: "Sweden", age: 21, sport: "Rugby" }),
(e10:Person { name: "Sarah", country: "Ireland", age: 35, sport: "Football" }),
(e11:Person { name: "Julia", country: "Scotland", age: 28, sport: "Football" }),
(e12:Person { name: "Hilary", country: "France", age: 24, sport: "Rugby" }),
(e13:Person { name: "Francis", country: "France", age: 25, sport: "Football" }),
(e14:Person { name: "Lisa", country: "Scotland", age: 25, sport: "Football" }),
(e15:Person { name: "Bart", country: "Scotland", age: 25, sport: "Rugby" }),
(e16:Person { name: "Denis", country: "Scotland", age: 34, sport: "Football" });
```

Explanation:

- Nodes Definition: This query creates 16 nodes, each representing a person with four properties: name, country, age, and sport.

- Application: Each node will be linked using the FRIEND_OF relationship, forming the social network graph.

Step 2: Creating FRIEND_OF Relationships

```
// Relationships for Mary
MATCH (e1:Person {name: "Mary"}), (e2:Person {name: "Emily"})
CREATE (e1)-[:FRIEND_OF]->(e2);
MATCH (e1:Person {name: "Mary"}), (e3:Person {name: "Mark"})
CREATE (e1)-[:FRIEND_OF]->(e3);
MATCH (e1:Person {name: "Mary"}), (e4:Person {name: "Joe"})
CREATE (e1)-[:FRIEND_OF]->(e4);
MATCH (e1:Person {name: "Mary"}), (e5:Person {name: "John"})
CREATE (e1)-[:FRIEND_OF]->(e5);
MATCH (e1:Person {name: "Mary"}), (e7:Person {name: "Paul"})
CREATE (e1)-[:FRIEND_OF]->(e7);
MATCH (e1:Person {name: "Mary"}), (e6:Person {name: "Peter"})
CREATE (e1)-[:FRIEND_OF]->(e6);

// Relationships for Emily
MATCH (e2:Person {name: "Emily"}), (e13:Person {name: "Francis"})
CREATE (e2)-[:FRIEND_OF]->(e13);
MATCH (e2:Person {name: "Emily"}), (e3:Person {name: "Mark"})
CREATE (e2)-[:FRIEND_OF]->(e3);
MATCH (e2:Person {name: "Emily"}), (e14:Person {name: "Lisa"})
CREATE (e2)-[:FRIEND_OF]->(e14);
MATCH (e2:Person {name: "Emily"}), (e5:Person {name: "John"})
CREATE (e2)-[:FRIEND_OF]->(e5);
MATCH (e2:Person {name: "Emily"}), (e7:Person {name: "Paul"})
CREATE (e2)-[:FRIEND_OF]->(e7);
```

```
// Relationships for Mark
MATCH (e3:Person {name: "Mark"}), (e10:Person {name: "Sarah"})
CREATE (e3)-[:FRIEND_OF]->(e10);
MATCH (e3:Person {name: "Mark"}), (e4:Person {name: "Joe"})
CREATE (e3)-[:FRIEND_OF]->(e4);
MATCH (e3:Person {name: "Mark"}), (e14:Person {name: "Lisa"})
CREATE (e3)-[:FRIEND_OF]->(e14);
MATCH (e3:Person {name: "Mark"}), (e5:Person {name: "John"})
CREATE (e3)-[:FRIEND_OF]->(e5);

// Relationships for Joe
MATCH (e4:Person {name: "Joe"}), (e10:Person {name: "Sarah"})
CREATE (e4)-[:FRIEND_OF]->(e10);
MATCH (e4:Person {name: "Joe"}), (e3:Person {name: "Mark"})
CREATE (e4)-[:FRIEND_OF]->(e3);
MATCH (e4:Person {name: "Joe"}), (e11:Person {name: "Julia"})
CREATE (e4)-[:FRIEND_OF]->(e11);
MATCH (e4:Person {name: "Joe"}), (e5:Person {name: "John"})
CREATE (e4)-[:FRIEND_OF]->(e5);

// Relationships for John
MATCH (e5:Person {name: "John"}), (e13:Person {name: "Francis"})
CREATE (e5)-[:FRIEND_OF]->(e13);
MATCH (e5:Person {name: "John"}), (e3:Person {name: "Mark"})
CREATE (e5)-[:FRIEND_OF]->(e3);
MATCH (e5:Person {name: "John"}), (e14:Person {name: "Lisa"})
CREATE (e5)-[:FRIEND_OF]->(e14);
MATCH (e5:Person {name: "John"}), (e1:Person {name: "Mary"})
CREATE (e5)-[:FRIEND_OF]->(e1);
MATCH (e5:Person {name: "John"}), (e8:Person {name: "Kevin"})
```

```
CREATE (e5)-[:FRIEND_OF]->(e8);
MATCH (e5:Person {name: "John"}), (e12:Person {name: "Hilary"})
CREATE (e5)-[:FRIEND_OF]->(e12);
MATCH (e5:Person {name: "John"}), (e9:Person {name: "Patrick"})
CREATE (e5)-[:FRIEND_OF]->(e9);
MATCH (e5:Person {name: "John"}), (e10:Person {name: "Sarah"})
CREATE (e5)-[:FRIEND_OF]->(e10);

// Relationships for Paul
MATCH (e7:Person {name: "Paul"}), (e14:Person {name: "Lisa"})
CREATE (e7)-[:FRIEND_OF]->(e14);
MATCH (e7:Person {name: "Paul"}), (e1:Person {name: "Mary"})
CREATE (e7)-[:FRIEND_OF]->(e1);
MATCH (e7:Person {name: "Paul"}), (e15:Person {name: "Bart"})
CREATE (e7)-[:FRIEND_OF]->(e15);
MATCH (e7:Person {name: "Paul"}), (e6:Person {name: "Peter"})
CREATE (e7)-[:FRIEND_OF]->(e6);
MATCH (e7:Person {name: "Paul"}), (e16:Person {name: "Denis"})
CREATE (e7)-[:FRIEND_OF]->(e16);
MATCH (e7:Person {name: "Paul"}), (e4:Person {name: "Joe"})
CREATE (e7)-[:FRIEND_OF]->(e4);
```

Explanation:

- Relationship Definition: Each CREATE statement establishes a directed FRIEND_OF relationship between two nodes.
- Functionality: Relationships allow path traversal during queries, enabling deep network analysis.

Step 3: Adding New Persons

The following queries create two new persons, Tom and Bill, and establish relationships:

```
// Add Tom and Connect to Mary
CREATE (t:Person {name: "Tom", age: 28, country: "Spain", sport: "Football"});
```

```
MATCH (m:Person {name: "Mary"}), (t:Person {name: "Tom"})
CREATE (m)-[:FRIEND_OF]->(t);

// Add Bill and Connect to Mary and Denis
CREATE (b:Person {name: "Bill", age: 23, country: "Ireland"});
MATCH (m:Person {name: "Mary"}), (d:Person {name: "Denis"}), (b:Person {name: "Bill"})
CREATE (m)-[:FRIEND_OF]->(b), (d)-[:FRIEND_OF]->(b);
```

Explanation:

- New Nodes Added: Two additional persons Tom and Bill were created and connected to Mary and Denis.
- Application: This expands the social graph and increases query possibilities.

PART 2: QUERIES AND ANALYSIS

Query 1: Age of Denis and His Friends

```
MATCH (d:Person {name: "Denis"})-[:FRIEND_OF]->(f)
RETURN d.name AS Name, d.age AS Age, f.name AS Friend, f.age AS Friend_Age;
```

Output:

List of Denis's friends and their ages.

	Name	Age	Friend	Friend_Age
1	"Denis"	34	"Bill"	23

Functionality:

This query retrieves Denis's direct friends using the FRIEND_OF relationship.

Query 2: Persons from Scotland

```
MATCH (p:Person)
WHERE p.country = "Scotland"
RETURN p.name AS Name, p.age AS Age, p.sport AS Sport;
```

Output:

All persons from Scotland along with their age and sport.

	Name	Age	Sport
1	"Julia"	28	"Football"
2	"Lisa"	25	"Football"
3	"Bart"	25	"Rugby"
4	"Denis"	34	"Football"

Functionality:

Filters nodes by the country property, returning only Scottish residents.

Query 3: Persons Aged ≤ 20 from Ireland

```
MATCH (p:Person)
WHERE p.country = "Ireland" AND p.age <= 20
RETURN p.name AS Name, p.age AS Age, p.sport AS Sport;
```

Output:

List of Irish persons aged 20 or younger.

	Name	Age	Sport
1	"Emily"	19	"Football"
2	"Kevin"	17	"Football"

Functionality:

Combines two conditions using AND to filter young Irish residents.

Query 4: Persons Aged ≤ 30 Playing Football

```
MATCH (p:Person)
WHERE p.age <= 30 AND p.sport = "Football"
```

RETURN p.name AS Name, p.age AS Age, p.country AS Country;

Output:

Persons aged ≤ 30 playing Football.

	Name	Age	Country
1	"Emily"	19	"Ireland"
2	"Kevin"	17	"Ireland"
3	"Julia"	28	"Scotland"
4	"Francis"	25	"France"
5	"Lisa"	25	"Scotland"
6	"Tom"	28	"Spain"

Functionality:

Filters based on age and sport properties.

Query 5: Count Persons by Country

MATCH (p:Person)

RETURN p.country AS Country, COUNT(p) AS Person_Count;

Output:

Count of persons grouped by country.

	Country	Person_Count
1	"Sweden"	5
2	"Ireland"	5
3	"France"	3
4	"Scotland"	4
5	"Spain"	1

Functionality:

Aggregates persons by their country and counts the number of nodes.

Query 6: Average Age by Sport

```
MATCH (p:Person)
RETURN p.sport AS Sport, AVG(p.age) AS Avg_Age;
```

Output:

Average age of persons grouped by sport.

	Sport	Avg_Age
1	"Hockey"	28.666666666666668
2	"Football"	26.888888888888886
3	"Rugby"	23.2
4	*null*	23.0

Functionality:

Computes the average using AVG() and groups results by sport.

Query 7: Direct Friends of Mary

```
MATCH (m:Person {name: "Mary"})-[:FRIEND_OF]->(f)
RETURN f.name AS Friend;
```

Output:

Names of Mary’s direct friends.

	Friend
1	"Emily"
2	"Mark"
3	"Joe"
4	"John"
5	"Peter"
6	"Paul"
7	"Tom"
8	"Bill"

Functionality:

Traverses outward from Mary using the FRIEND_OF relationship.

Query 8: Friends of Paul (Max 5 Steps)

```
MATCH (p:Person {name: "Paul"})-[:FRIEND_OF*1..5]->(f)
RETURN DISTINCT f.name AS Friend;
```

Output:

Friends of Paul up to 5 connections away.

	Friend
1	"Mary"
2	"Joe"
3	"Peter"
4	"Lisa"
5	"Bart"

Functionality:

Expands the search depth using the *1..5 range operator.

Query 9: Count Friends of Paul by Nationality (Max 5 Steps)

```
MATCH (p:Person {name: "Paul"})-[:FRIEND_OF*1..5]->(f)
RETURN f.country AS Country, COUNT(DISTINCT f) AS Count;
```

Output:

A list of countries and the number of Paul's friends from each country.

	Country	Count
1	"Sweden"	5
2	"France"	3
3	"Scotland"	4
4	"Ireland"	5
5	"Spain"	1

Functionality:

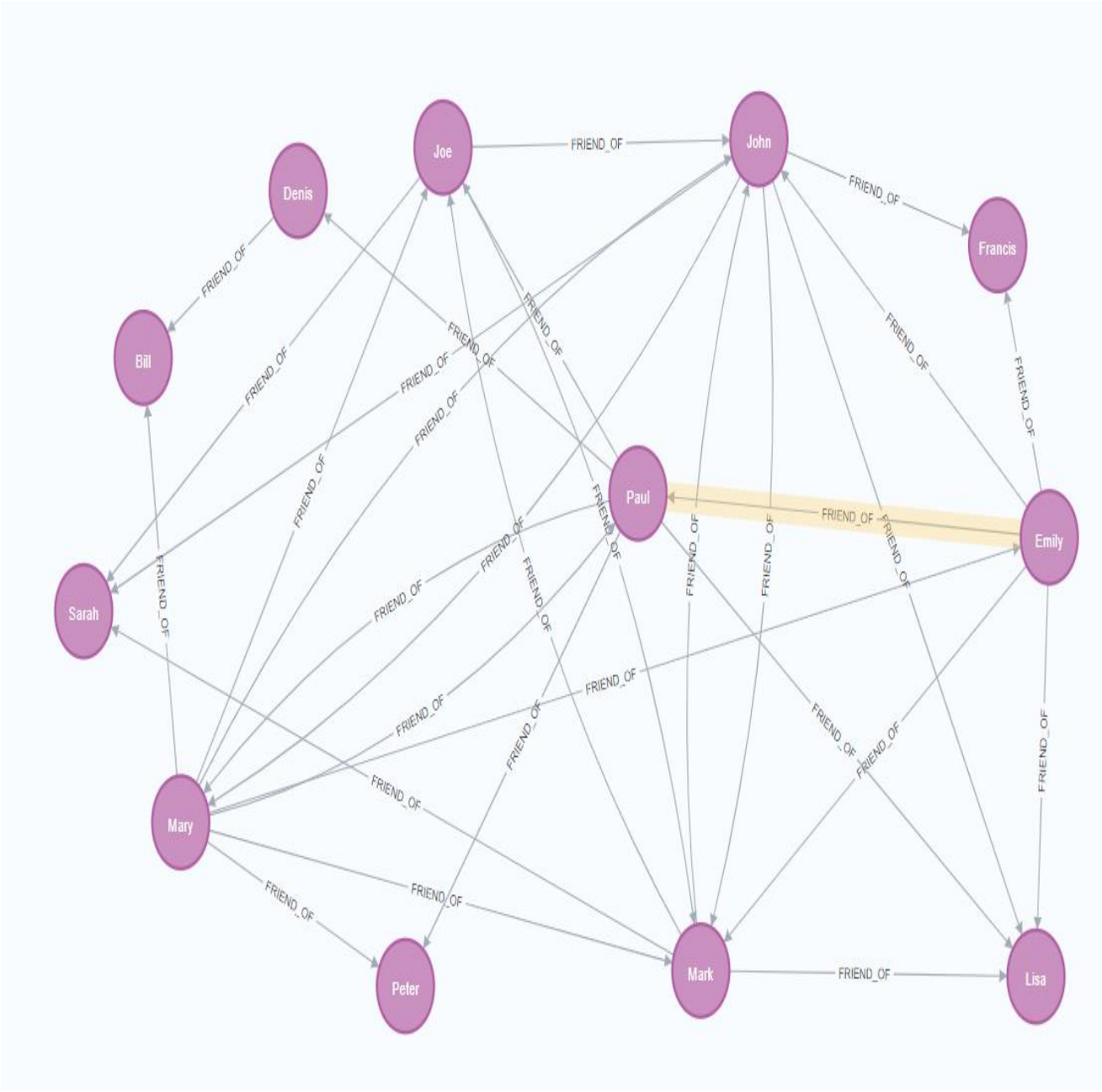
Finds all unique friends of Paul within five steps and counts them by nationality.

Query 10: Paths Between Paul and Lisa with Length

```
MATCH path = (p:Person {name: "Paul"})-[:FRIEND_OF*1..5]-(l:Person {name: "Lisa"})
RETURN path, LENGTH(path) AS Path_Length;
```

Output:

All paths from Paul to Lisa with path lengths.



Functionality:

Retrieves paths between Paul and Lisa within five steps.

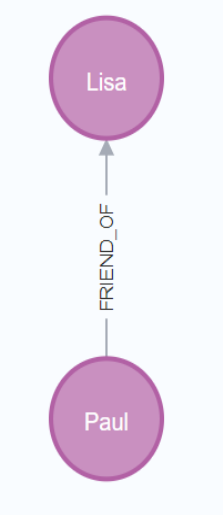
Query 11: Shortest Path Between Paul and Lisa

MATCH path = shortestPath((p:Person {name: "Paul"})-[:FRIEND_OF*]-(l:Person {name: "Lisa"}))

RETURN path;

Output:

The shortest path between Paul and Lisa.



Functionality:

Finds the shortest connection between Paul and Lisa using the FRIEND_OF relationship in both directions.

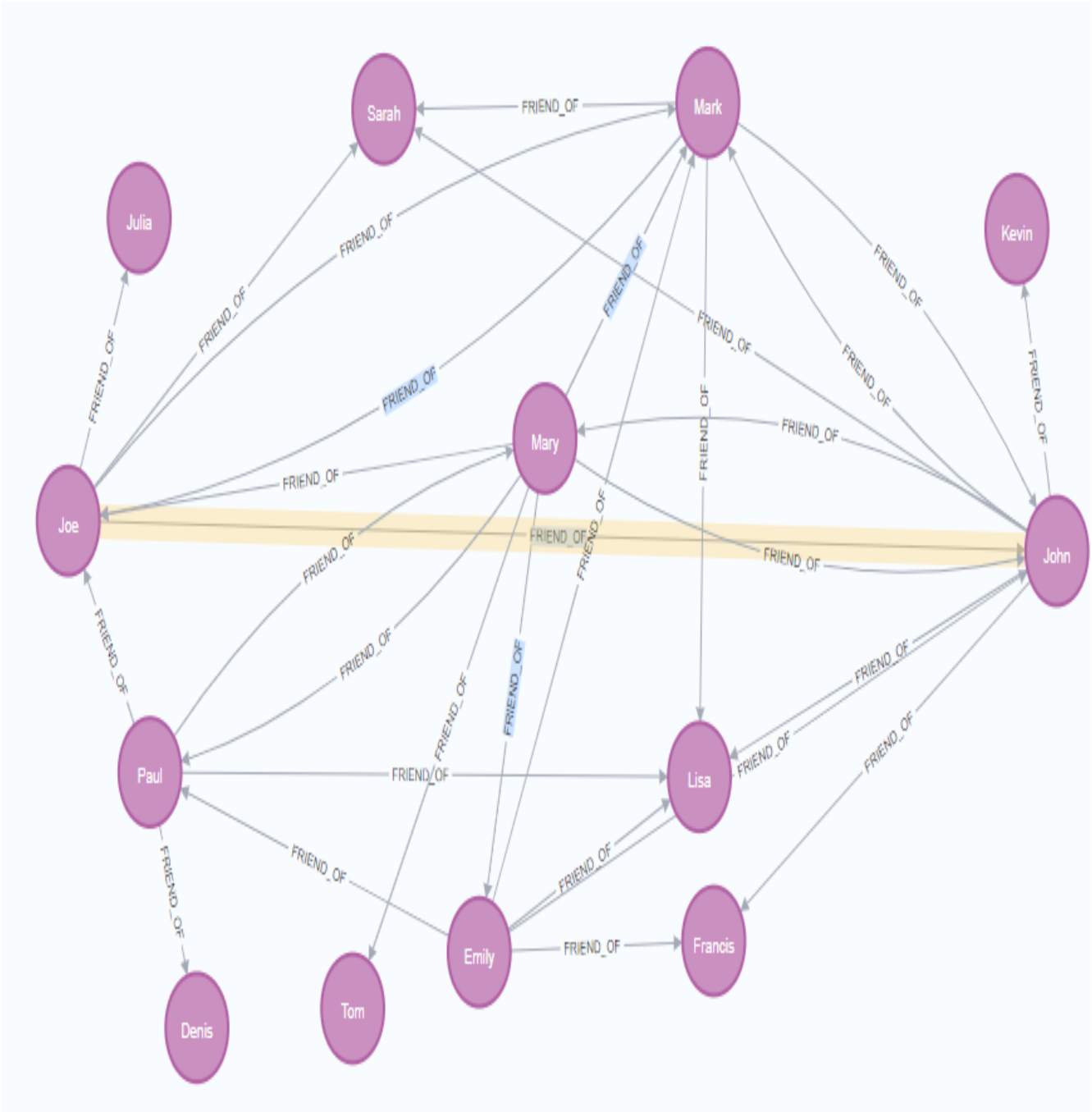
Query 12: Mary's Friends Playing Football

MATCH path = (m:Person {name: "Mary"})-[:FRIEND_OF*1..5]->(f:Person {sport: "Football"})

RETURN DISTINCT path;

Output:

Paths connecting Mary to friends who play football.



Functionality:

Retrieves all paths from Mary to friends playing football within five steps.

CONCLUSION

This exercise demonstrates the creation, querying, and analysis of a graph database using Neo4j. The dataset includes persons from various countries and sports, interconnected by friendships. By using Cypher, I successfully created a social network graph, established relationships, and performed complex queries as required.

EXERCISE 2

INTRODUCTION

This exercise involves creating and querying an airport graph database using Neo4j. The database represents airports as nodes and flight connections as relationships, with properties for time (flight duration in minutes) and price (flight cost in euros). The objective is to demonstrate graph creation, query execution, and analysis using Cypher Query Language.

PART 1: GRAPH CREATION

Step 1: Create Airport Nodes

The following query creates 13 airport nodes, each with properties for city, country, and code.

```
CREATE
(dub:Airport {city: "Dublin", country: "Ireland", code: "DUB"}),
(ork:Airport {city: "Cork", country: "Ireland", code: "ORK"}),
(lhr:Airport {city: "London", country: "UK", code: "LHR"}),
(fco:Airport {city: "Rome", country: "Italy", code: "FCO"}),
(dme:Airport {city: "Moscow", country: "Russia", code: "DME"}),
(hkg:Airport {city: "Hongkong", country: "China", code: "HKG"}),
(ams:Airport {city: "Amsterdam", country: "Holland", code: "AMS"}),
(txl:Airport {city: "Berlin", country: "Germany", code: "TXL"}),
(cdg:Airport {city: "Paris", country: "France", code: "CDG"}),
(jfk:Airport {city: "New York", country: "USA", code: "JFK"}),
(ord:Airport {city: "Chicago", country: "USA", code: "ORD"}),
(gru:Airport {city: "Sao Paulo", country: "Brazil", code: "GRU"}),
(gig:Airport {city: "Rio", country: "Brazil", code: "GIG"});
```

Explanation:

- Each node represents an airport, and its properties (city, country, and code) uniquely identify it.
- This is a foundational step for creating the airport graph database.

Functionality:

Nodes are the entities in the graph. They will be linked by flight connections in the next step.

Step 2: Create Flight Connections

The following queries establish CONNECTED_TO relationships between airports. Each relationship has properties time (flight duration in minutes) and price (cost in euros).

```
MATCH (a1:Airport {code: "LHR"}), (a2:Airport {code: "DUB"})
CREATE (a1)-[:CONNECTED_TO {time: 45, price: 150}]->(a2);
MATCH (a1:Airport {code: "FCO"}), (a2:Airport {code: "LHR"})
CREATE (a1)-[:CONNECTED_TO {time: 150, price: 400}]->(a2);
MATCH (a1:Airport {code: "FCO"}), (a2:Airport {code: "CDG"})
CREATE (a1)-[:CONNECTED_TO {time: 120, price: 500}]->(a2);
MATCH (a1:Airport {code: "DUB"}), (a2:Airport {code: "JFK"})
CREATE (a1)-[:CONNECTED_TO {time: 360, price: 800}]->(a2);
MATCH (a1:Airport {code: "DUB"}), (a2:Airport {code: "ORK"})
CREATE (a1)-[:CONNECTED_TO {time: 50, price: 50}]->(a2);
MATCH (a1:Airport {code: "AMS"}), (a2:Airport {code: "HKG"})
CREATE (a1)-[:CONNECTED_TO {time: 660, price: 750}]->(a2);
MATCH (a1:Airport {code: "ORD"}), (a2:Airport {code: "JFK"})
CREATE (a1)-[:CONNECTED_TO {time: 240, price: 430}]->(a2);
MATCH (a1:Airport {code: "GRU"}), (a2:Airport {code: "JFK"})
CREATE (a1)-[:CONNECTED_TO {time: 840, price: 650}]->(a2);
MATCH (a1:Airport {code: "DME"}), (a2:Airport {code: "GIG"})
CREATE (a1)-[:CONNECTED_TO {time: 1200, price: 1100}]->(a2);
```

Explanation:

- Each CONNECTED_TO relationship represents a flight connection between two airports.
- The properties time and price provide additional details about each flight connection.

Functionality:

Relationships in Neo4j allow traversing between nodes and querying based on specific criteria.

PART 2: QUERY EXECUTION

Query 1: Total Flight Time from Moscow to Rio

```
MATCH path = (dme:Airport {code: "DME"})-[:CONNECTED_TO*]->(gig:Airport {code: "GIG"})

RETURN path, REDUCE(totalTime = 0, r IN relationships(path) | totalTime + r.time) AS Total_Flight_Time;
```

Explanation:

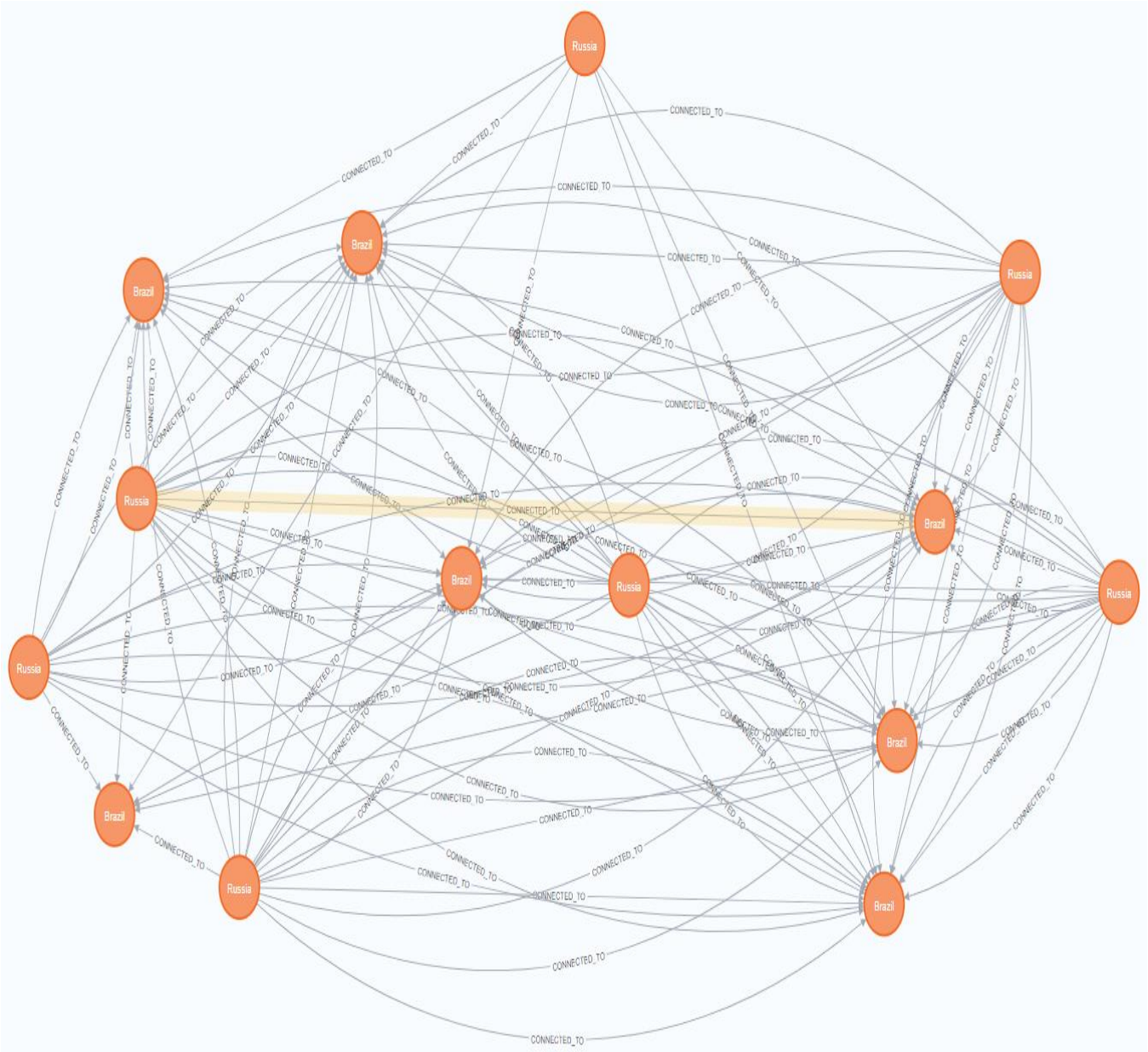
- This query uses MATCH to find all paths between Moscow (DME) and Rio (GIG).
- The REDUCE function accumulates the total flight time along each path.

Functionality:

Demonstrates path traversal and aggregation using Neo4j's Cypher query language.

Output:

- Path: Displays all connections from Moscow to Rio.
- Total Flight Time: 1200 minutes.
- Price: 1100



Query 2: Flights from Dublin Sorted by Price

```
MATCH (dub:Airport {code: "DUB"})-[r:CONNECTED_TO]->(dest:Airport)
RETURN DISTINCT dest.city AS Destination, dest.code AS Code, dest.country AS Country,
    r.time AS Flight_Time, r.price AS Price
ORDER BY r.price DESC;
```

Explanation:

- Retrieves flights originating from Dublin (DUB).
- The ORDER BY clause sorts the results by price in descending order.
- DISTINCT ensures there are no duplicate destinations in the output.

Functionality:

Retrieves detailed information about flights and ensures uniqueness and order.

Output:

	Destination	Code	Country	Flight_Time	Price
1	"Chicago"	"ORD"	"USA"	480	890
2	"New York"	"JFK"	"USA"	360	800
3	"Sao Paulo"	"GRU"	"Brazil"	900	800
4	"Rome"	"FCO"	"Italy"	150	70
5	"Amsterdam"	"AMS"	"Holland"	90	60
6	"Cork"	"ORK"	"Ireland"	50	50

Query 3: Destinations from Chicago in 1-2 Steps

MATCH path = (ord:Airport {code: "ORD"})-[:CONNECTED_TO*1..2]->(dest:Airport)

RETURN DISTINCT dest.city AS Destination, dest.code AS Code, dest.country AS Country;

Explanation:

- Retrieves destinations reachable from Chicago (ORD) in 1 or 2 steps.
- The *1..2 specifies a path length between 1 and 2 relationships.

Functionality:

Demonstrates how to query multi-step relationships in a graph database.

Output:

	Destination	Code	Country
1	"New York"	"JFK"	"USA"
2	"Chicago"	"ORD"	"USA"

Query 4: Flights from London Under 240 Minutes

```
MATCH (lhr:Airport {code: "LHR"})-[r:CONNECTED_TO]->(dest:Airport)

WHERE r.time < 240

RETURN DISTINCT dest.city AS Destination, dest.code AS Code, dest.country AS Country, r.time AS Flight_Time;
```

Explanation:

- Retrieves flights from London (LHR) with flight times under 240 minutes.
- The WHERE clause filters based on the time property.

Functionality:

Illustrates filtering results using relationship properties.

Output:

	Destination	Code	Country	Flight_Time
1	"Dublin"	"DUB"	"Ireland"	45

CONCLUSION

This exercise demonstrates:

- The successful creation of a graph database with airport nodes and flight connections.
- Execution of Cypher queries to retrieve travel-related information.
- Correct use of graph database features such as relationship traversal, aggregation, and filtering.

The exercise validates Neo4j's capabilities in managing and querying graph-structured data effectively.