Experiment report: Graph DataBases: Neo4j

Done by student: **Bylkova Kristina (伯汀娜), 1820249049**
Course: **Big Data Analysis Technology**
Date: **October 2024**

Summary

This report shows the MindMap and the use of the Neo4j DataBase, which is a native graph database designed specifically to store and manage data as a graph. Where it uses nodes and edges as complex, interconnected data.

MindMap

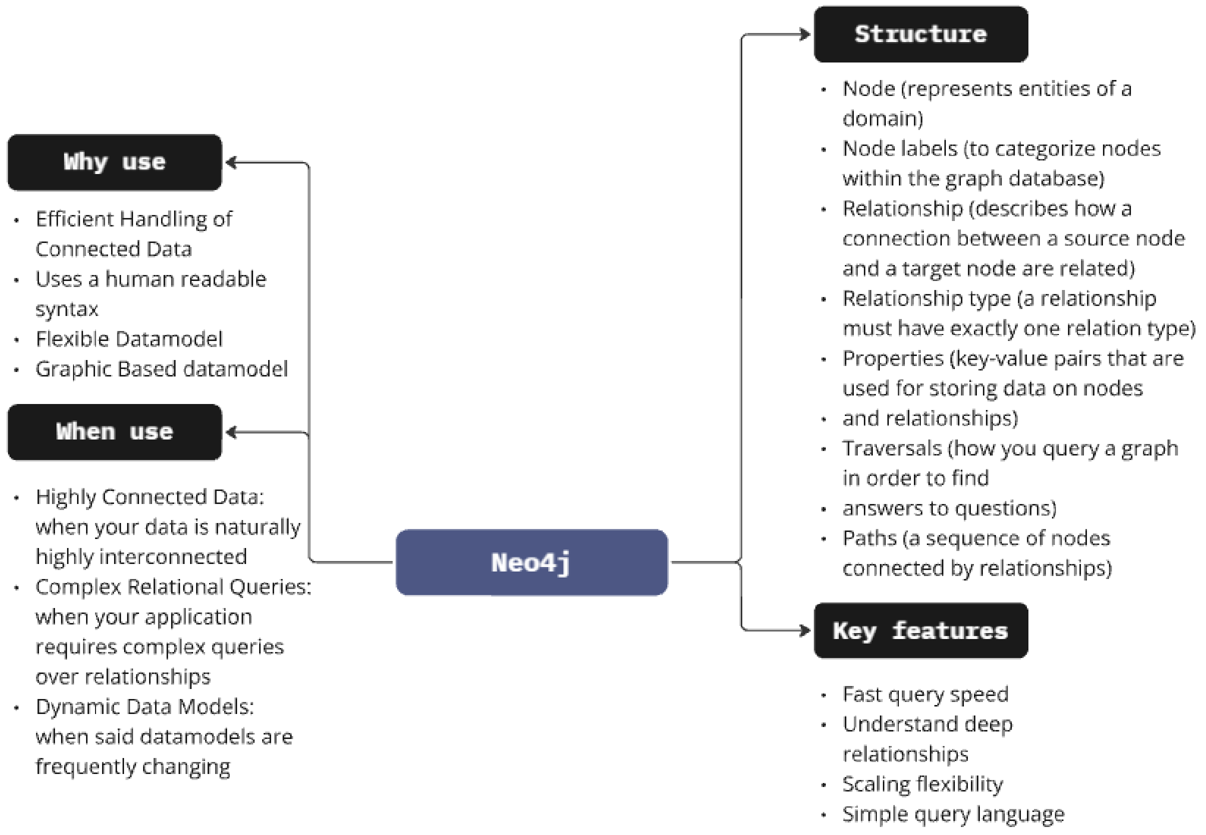

Structure
Node (represents entities of a domain)
Node labels (to categorize nodes within the graph database)
Relationship (describes how a connection between a source node and a target node are related)
Relationship type (a relationship must have exactly one relation type)
Properties (key-value pairs that are used for storing data on nodes
and relationships)
Traversals (how you query a graph in order to find
answers to questions)
Paths (a sequence of nodes connected by relationships)
Why use
Efficient Handling of Connected Data
Uses a human readable syntax
Flexible Datamodel
Graphic Based datamodel
When use
Highly Connected Data: when your data is naturally highly interconnected
Complex Relational Queries: when your application requires complex queries over relationships
Dynamic Data Models: when said datamodels are frequently changing
Neo4j
Key features
Fast query speed
Understand deep relationships
Scaling flexibility
Simple query language

Figure 1: Mindmap

Task

Explain native graph storage and non-native graph storage, compare the difference, and explain why the native graph storage store queries in the storage mechanism.

Answer:

In native graph storage, the database is specifically designed to store nodes, relationships, and properties efficiently. Both the physical storage structure and the query engine are optimized for graph traversal and manipulation. Graph elements (nodes, edges, properties) are stored directly in a graph-optimized format on disk.

In non-native graph storage, the graph database is built on top of an underlying storage system that was not designed with graphs in mind. Data is typically stored in a non-graph format, such as key-value pairs, documents, or relational tables. Nodes and edges may be stored in different tables, or as documents with pointers to each other.

Native graph storage is more efficient because it is designed specifically to handle graph data. Nodes and relationships are stored directly, allowing for faster navigation and simpler queries without needing complicated joins.

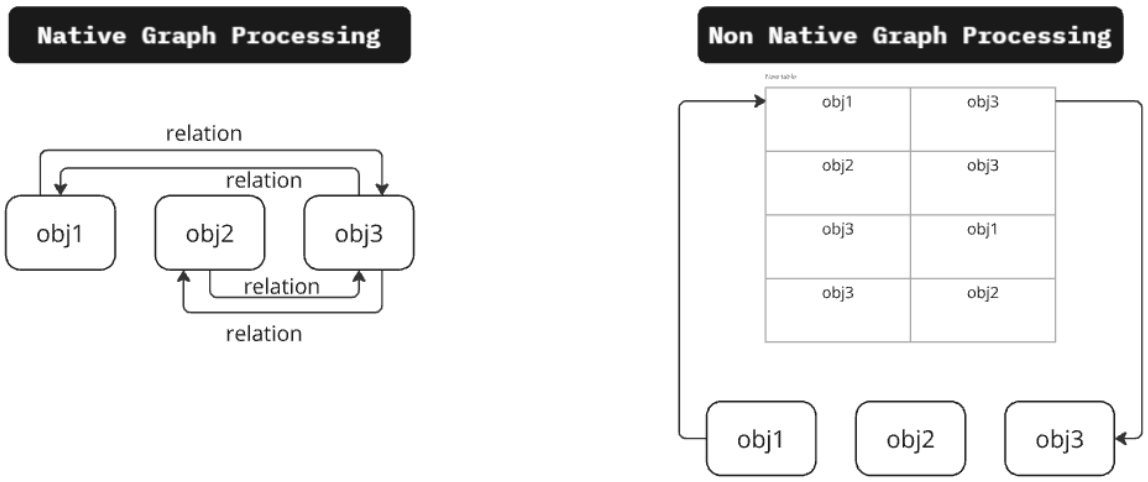
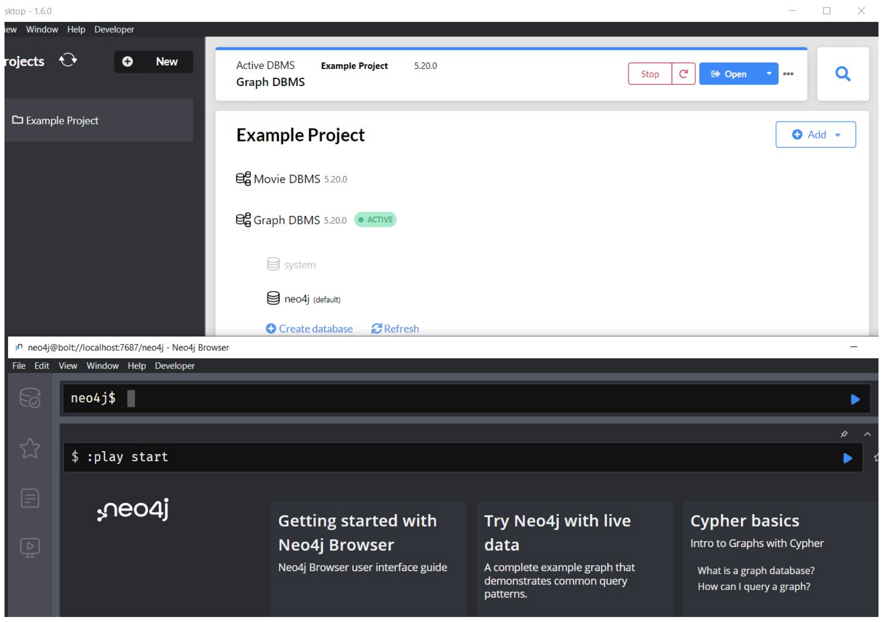


Figure 2: Native and Non Native

Installation

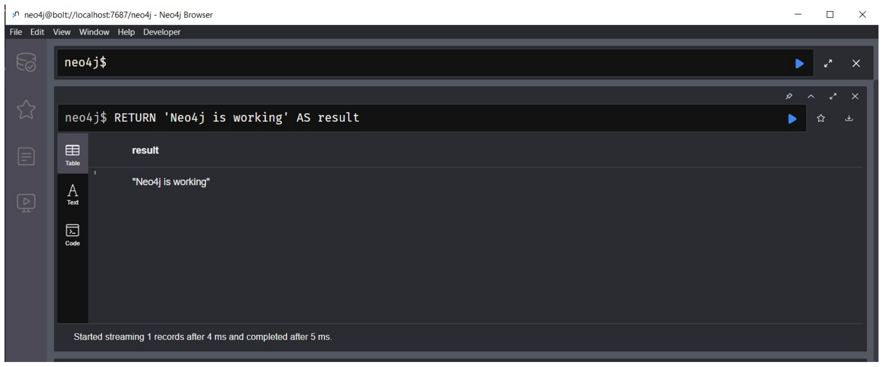
I installed and opened Neo4j Desktop, created a new DataBase and started it.


sktop - 1.6.0
iew Window Help Developer
rojects
New
Example Project
Active DBMS
Example Project
5.20.0
Graph DBMS
Stop
Open
Example Project
Add
Movie DBMS 5.20.0
Graph DBMS 5.20.0 ACTIVE
system
neo4j (default)
Create database
Refresh
neo4j@bolt://localhost:7687/neo4j - Neo4j Browser
File Edit View Window Help Developer
neo4j
Getting started with Neo4j Browser
Neo4j Browser user interface guide
Try Neo4j with live data
A complete example graph that demonstrates common query patterns.
Cypher basics
Intro to Graphs with Cypher
What is a graph database?
How can I query a graph?

```
neo4j$
$ :play start
```

Figure 3: Neo4j Desktop

Then I ran Cypher query to check if the database is working.


neo4j@bolt://localhost:7687/neo4j - Neo4j Browser
File Edit View Window Help Developer
Table
Text
Code

```
neo4j$
neo4j$ RETURN 'Neo4j is working' AS result
result
"Neo4j is working"
Started streaming 1 records after 4 ms and completed after 5 ms.
```

Figure 4: Check if working

Basic usage

I used basic commands in Neo4j shell here:

1. Create

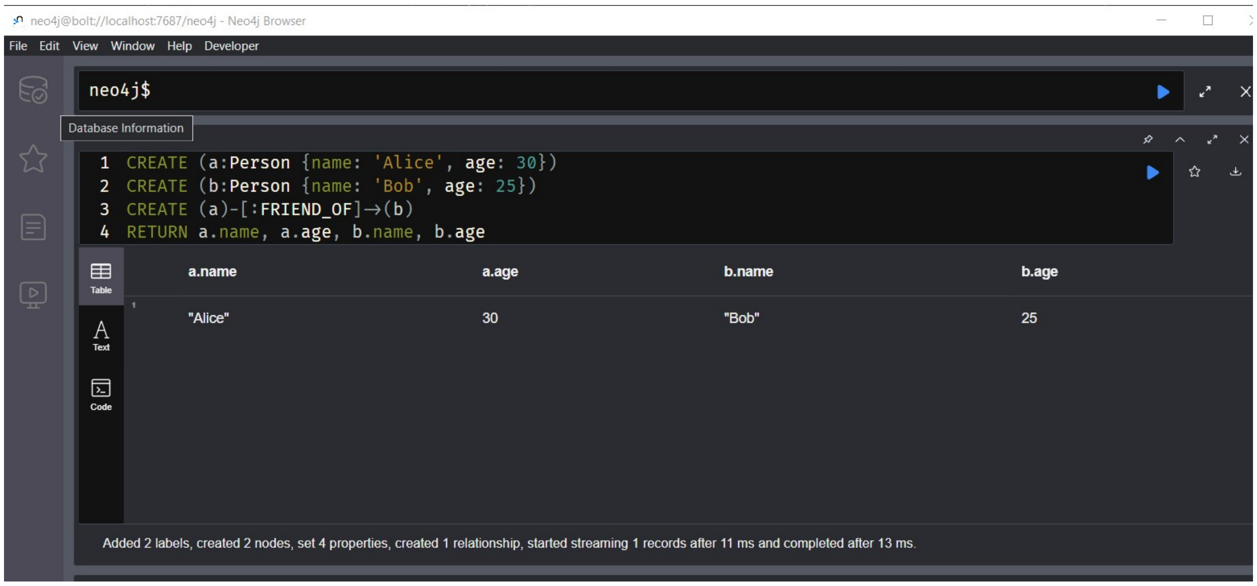
```
CREATE (a:Person {name: 'Alice', age: 30})
CREATE (b:Person {name: 'Bob', age: 25})
```

2. Create a relationship

```
CREATE (a)-[:FRIEND_OF]->(b)
```

3. Create a graph

```
RETURN a.name, a.age, b.name, b.age
```



```
CREATE (a:Person {name: 'Alice', age: 30})
CREATE (b:Person {name: 'Bob', age: 25})
CREATE (a)-[:FRIEND_OF]→(b)
RETURN a.name, a.age, b.name, b.age
```

	a.name	a.age	b.name	b.age
1	"Alice"	30	"Bob"	25

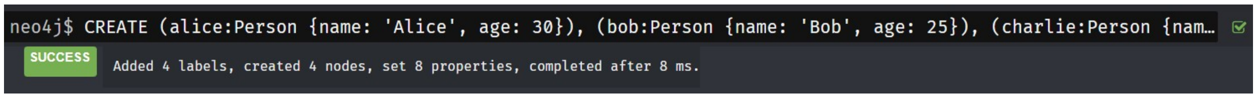
Added 2 labels, created 2 nodes, set 4 properties, created 1 relationship, started streaming 1 records after 11 ms and completed after 13 ms.

Figure 5: Result for basic operations

Simple experiment

Step 1: Create People Nodes

```
CREATE (alice:Person {name: 'Alice', age: 30}),
       (bob:Person {name: 'Bob', age: 25}),
       (charlie:Person {name: 'Charlie', age: 35}),
       (david:Person {name: 'David', age: 28});
```

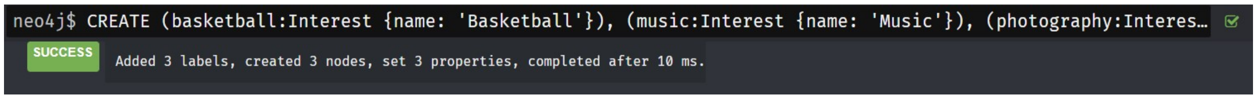


```
neo4j$ CREATE (alice:Person {name: 'Alice', age: 30}), (bob:Person {name: 'Bob', age: 25}), (charlie:Person {nam…
SUCCESS  Added 4 labels, created 4 nodes, set 8 properties, completed after 8 ms.
```

Figure 6: Simple experiment: step 1

Step 2: Create Interests Nodes

```
CREATE (basketball:Interest {name: 'Basketball'}),
       (music:Interest {name: 'Music'}),
       (photography:Interest {name: 'Photography'});
```

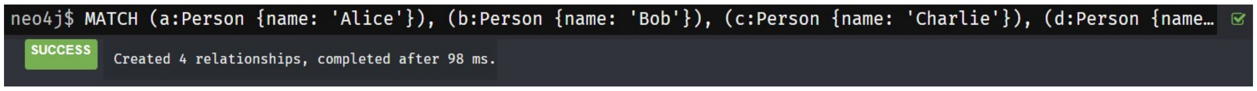


```
neo4j$ CREATE (basketball:Interest {name: 'Basketball'}), (music:Interest {name: 'Music'}), (photography:Interes…
SUCCESS  Added 3 labels, created 3 nodes, set 3 properties, completed after 10 ms.
```

Figure 7: Simple experiment: step 2

Step 3: Create Friendships between People

```
MATCH (a:Person {name: 'Alice'}), (b:Person {name: 'Bob'}),
      (c:Person {name: 'Charlie'}), (d:Person {name: 'David'})
CREATE (a)-[:FRIEND_OF]->(b),
       (b)-[:FRIEND_OF]->(c),
       (a)-[:FRIEND_OF]->(d),
       (c)-[:FRIEND_OF]->(d);
```

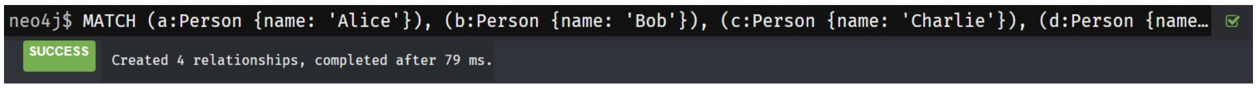


```
neo4j$ MATCH (a:Person {name: 'Alice'}), (b:Person {name: 'Bob'}), (c:Person {name: 'Charlie'}), (d:Person {name…
SUCCESS  Created 4 relationships, completed after 98 ms.
```

Figure 8: Simple experiment: step 3

Step 4: Connect People with their Interests

```
MATCH (a:Person {name: 'Alice'}), (b:Person {name: 'Bob'}),
      (c:Person {name: 'Charlie'}), (d:Person {name: 'David'}),
      (basketball:Interest {name: 'Basketball'}), (music:Interest {name: 'Music'}), (photography:
    Interest {name: 'Photography'})
CREATE (a)-[:INTERESTED_IN]->(basketball),
       (b)-[:INTERESTED_IN]->(music),
       (c)-[:INTERESTED_IN]->(photography),
       (d)-[:INTERESTED_IN]->(music);
```

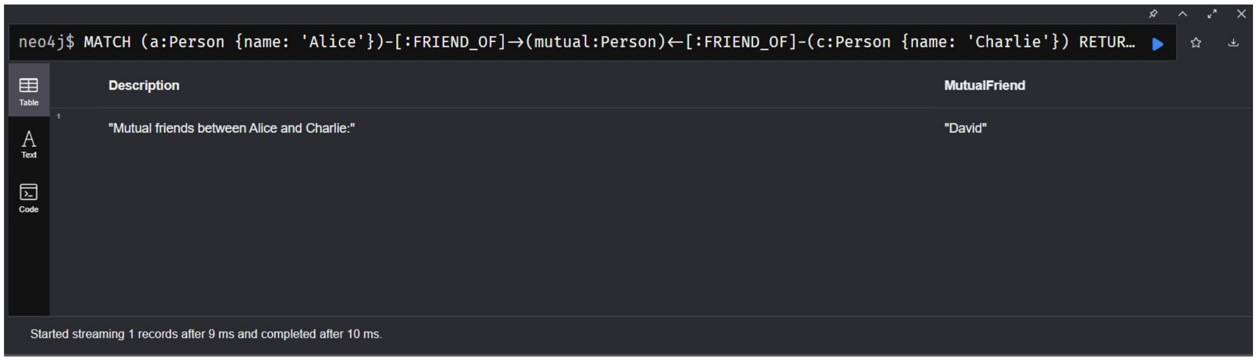


```
neo4j$ MATCH (a:Person {name: 'Alice'}), (b:Person {name: 'Bob'}), (c:Person {name: 'Charlie'}), (d:Person {name…
SUCCESS  Created 4 relationships, completed after 79 ms.
```

Figure 9: Simple experiment: step 4

Step 5: Query for Mutual Friends between Alice and Charlie

```
MATCH (a:Person {name: 'Alice'})-[:FRIEND_OF]->(mutual:Person)<-[:FRIEND_OF]-(c:Person {name: 'Charlie'
    })
RETURN 'Mutual friends between Alice and Charlie:' AS Description, mutual.name AS MutualFriend;
```



```
neo4j$ MATCH (a:Person {name: 'Alice'})-[:FRIEND_OF]→(mutual:Person)←[:FRIEND_OF]-(c:Person {name: 'Charlie'}) RETUR…
```

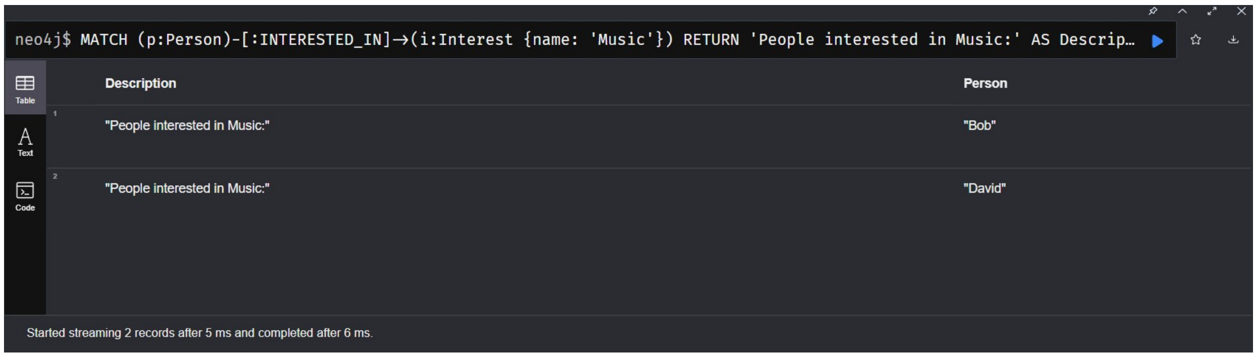
	Description	MutualFriend
1	"Mutual friends between Alice and Charlie:"	"David"

Started streaming 1 records after 9 ms and completed after 10 ms.

Figure 10: Simple experiment: step 5

Step 6: Find People Interested in Music

```
MATCH (p:Person)-[:INTERESTED_IN]->(i:Interest {name: 'Music'})
RETURN 'People interested in Music:' AS Description, p.name AS Person;
```



```
neo4j$ MATCH (p:Person)-[:INTERESTED_IN]→(i:Interest {name: 'Music'}) RETURN 'People interested in Music:' AS Descrip…
```

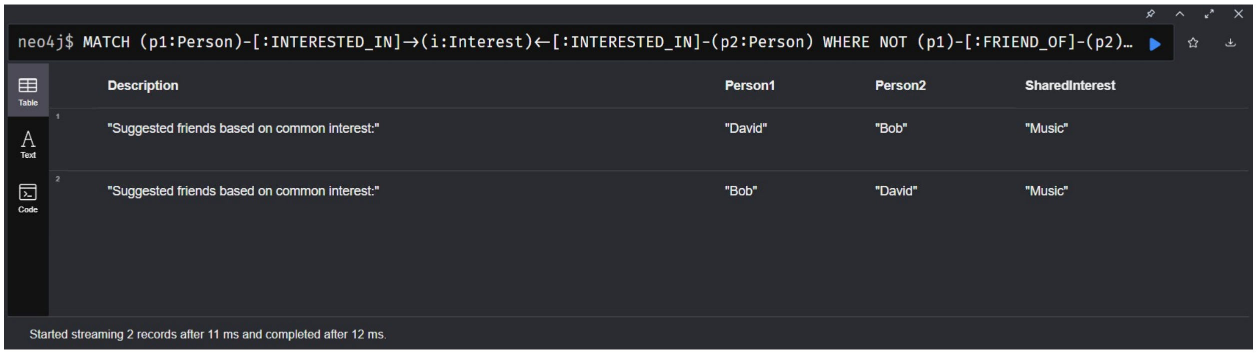
	Description	Person
1	"People interested in Music:"	"Bob"
2	"People interested in Music:"	"David"

Started streaming 2 records after 5 ms and completed after 6 ms.

Figure 11: Simple experiment: step 6

Step 7: Suggest Friends based on Shared Interests

```
MATCH (p1:Person)-[:INTERESTED_IN]->(i:Interest)<-[:INTERESTED_IN]-(p2:Person)
WHERE NOT (p1)-[:FRIEND_OF]-(p2)
RETURN 'Suggested friends based on common interest:' AS Description, p1.name AS Person1, p2.name AS
    Person2, i.name AS SharedInterest;
```



```
neo4j$ MATCH (p1:Person)-[:INTERESTED_IN]→(i:Interest)←[:INTERESTED_IN]-(p2:Person) WHERE NOT (p1)-[:FRIEND_OF]-(p2)…
```

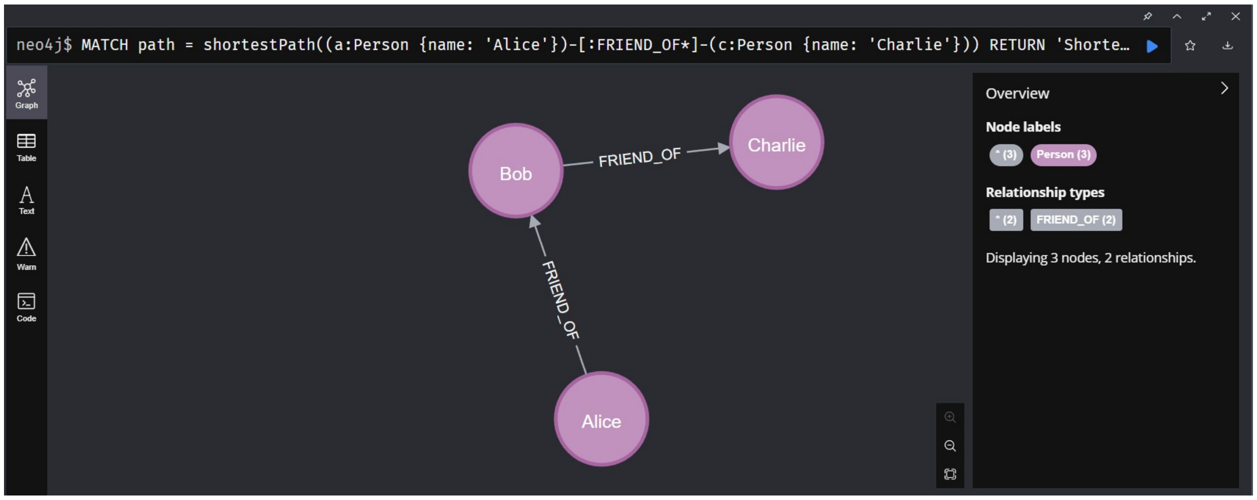
	Description	Person1	Person2	SharedInterest
1	"Suggested friends based on common interest:"	"David"	"Bob"	"Music"
2	"Suggested friends based on common interest:"	"Bob"	"David"	"Music"

Started streaming 2 records after 11 ms and completed after 12 ms.

Figure 12: Simple experiment: step 7

Step 8: Find Shortest Path between Alice and Charlie

```
MATCH path = shortestPath((a:Person {name: 'Alice'})-[:FRIEND_OF*]-(c:Person {name: 'Charlie'}))
RETURN 'Shortest path between Alice and Charlie:' AS Description, path;
```

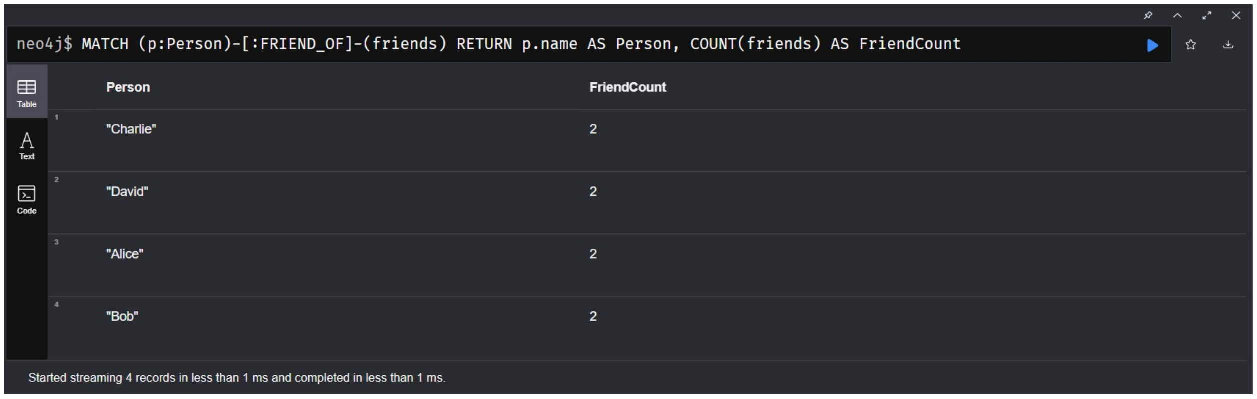


```
neo4j$ MATCH path = shortestPath((a:Person {name: 'Alice'})-[:FRIEND_OF*]-(c:Person {name: 'Charlie'})) RETURN 'Shorte…
```

Figure 13: Simple experiment: step 8

Step 9: Find Degree of Connection (Number of Friends)

```
MATCH (p:Person)-[:FRIEND_OF]-(friends)
RETURN p.name AS Person, COUNT(friends) AS FriendCount;
```



```
neo4j$ MATCH (p:Person)-[:FRIEND_OF]-(friends) RETURN p.name AS Person, COUNT(friends) AS FriendCount
```

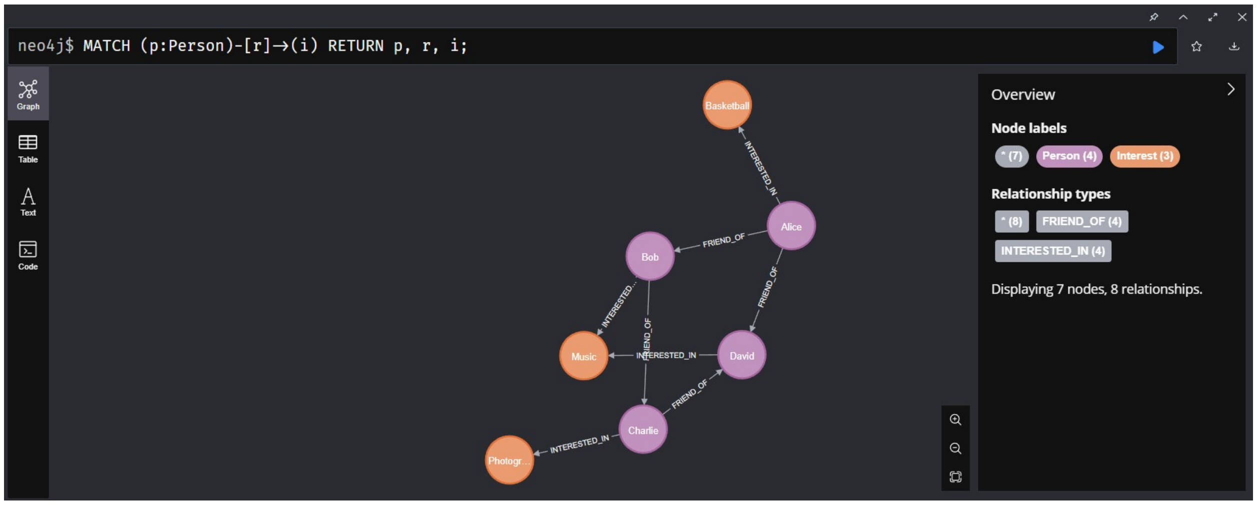
	Person	FriendCount
1	"Charlie"	2
2	"David"	2
3	"Alice"	2
4	"Bob"	2

Started streaming 4 records in less than 1 ms and completed in less than 1 ms.

Figure 14: Simple experiment: step 9

Step 10: Return the Entire Graph Structure

```
MATCH (p:Person)-[r]->(i)
RETURN p, r, i;
```



```
neo4j$ MATCH (p:Person)-[r]→(i) RETURN p, r, i;
```

Overview

Node labels

* (7) Person (4) Interest (3)

Relationship types

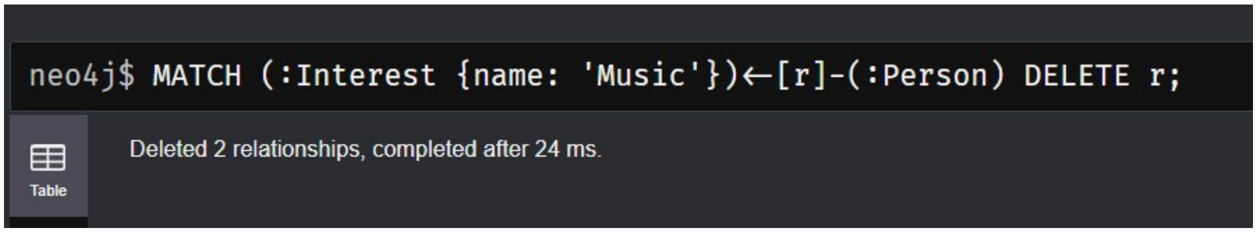
* (8) FRIEND_OF (4) INTERESTED_IN (4)

Displaying 7 nodes, 8 relationships.

Figure 15: Simple experiment: step 10

Step 11: Delete all Relations connected to Music

```
MATCH (:Interest {name: 'Music'})<-[r]-(:Person)
DELETE r;
```



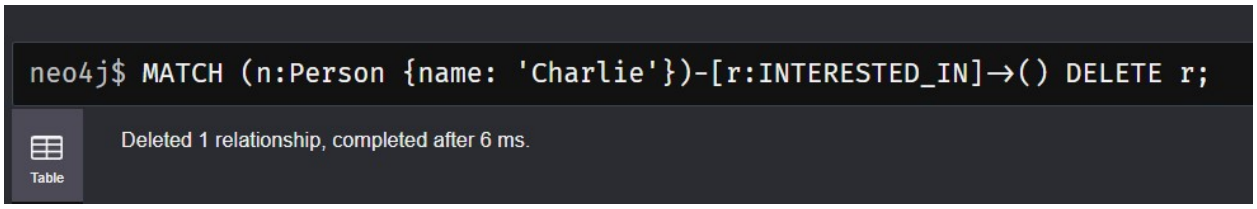
```
neo4j$ MATCH (:Interest {name: 'Music'})←[r]-(:Person) DELETE r;
```

```
Deleted 2 relationships, completed after 24 ms.
```

Figure 16: Simple experiment: step 11

Step 12: Delete relationship between Charlie and Photography

```
MATCH (n:Person {name: 'Charlie'})-[r:INTERESTED_IN]->()
DELETE r;
```



```
neo4j$ MATCH (n:Person {name: 'Charlie'})-[r:INTERESTED_IN]→() DELETE r;
```

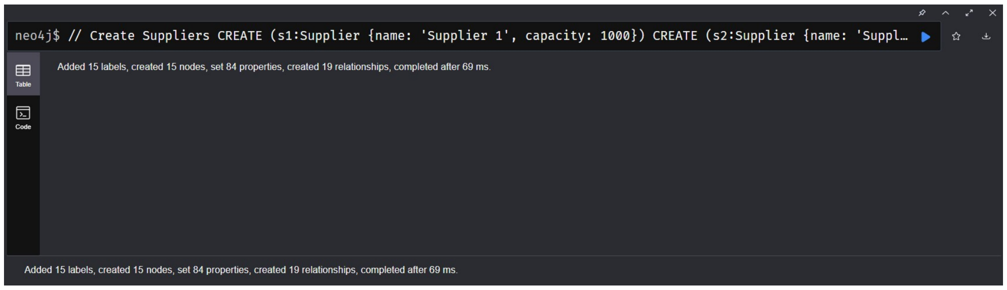
```
Deleted 1 relationship, completed after 6 ms.
```

Figure 17: Simple experiment: step 12

Complex experiment

Step 1: Write down all the nodes and edges of the database

```
// Create Suppliers
CREATE (s1:Supplier {name: 'Supplier 1', capacity: 1000})
CREATE (s2:Supplier {name: 'Supplier 2', capacity: 1200})
CREATE (s3:Supplier {name: 'Supplier 3', capacity: 800})

// Create Manufacturers
CREATE (m1:Manufacturer {name: 'Manufacturer 1', productionRate: 500})
CREATE (m2:Manufacturer {name: 'Manufacturer 2', productionRate: 300})
CREATE (m3:Manufacturer {name: 'Manufacturer 3', productionRate: 400})

// Create Warehouses/Distribution Centers
CREATE (w1:Warehouse {name: 'Warehouse 1', stock: 700})
CREATE (w2:Warehouse {name: 'Warehouse 2', stock: 500})
CREATE (w3:Warehouse {name: 'Warehouse 3', stock: 600})

// Create Retailers
CREATE (r1:Retailer {name: 'Retailer 1', demand: 350})
CREATE (r2:Retailer {name: 'Retailer 2', demand: 250})
CREATE (r3:Retailer {name: 'Retailer 3', demand: 400})

// Create Transportation Hubs
CREATE (t1:TransportHub {name: 'Transport Hub 1'})
CREATE (t2:TransportHub {name: 'Transport Hub 2'})
CREATE (t3:TransportHub {name: 'Transport Hub 3'})

// Create Routes (Relationships between nodes with varying transport modes, costs, and times)
CREATE (s1)-[:SUPPLIES {mode: 'road', cost: 20, time: 5}]->(m1)
CREATE (s2)-[:SUPPLIES {mode: 'sea', cost: 50, time: 10}]->(m1)
CREATE (s3)-[:SUPPLIES {mode: 'air', cost: 70, time: 3}]->(m2)
CREATE (m1)-[:PRODUCES {mode: 'road', cost: 10, time: 2}]->(w1)
CREATE (m2)-[:PRODUCES {mode: 'sea', cost: 40, time: 8}]->(w2)
CREATE (m3)-[:PRODUCES {mode: 'air', cost: 30, time: 4}]->(w3)
CREATE (w1)-[:DISTRIBUTES {mode: 'road', cost: 15, time: 3}]->(r1)
CREATE (w2)-[:DISTRIBUTES {mode: 'sea', cost: 25, time: 7}]->(r2)
CREATE (w3)-[:DISTRIBUTES {mode: 'air', cost: 35, time: 2}]->(r3)

// Additional Routes to increase complexity
CREATE (m1)-[:TRANSPORTS_VIA {mode: 'road', cost: 5, time: 1}]->(t1)
CREATE (m2)-[:TRANSPORTS_VIA {mode: 'sea', cost: 15, time: 6}]->(t2)
CREATE (m3)-[:TRANSPORTS_VIA {mode: 'air', cost: 25, time: 3}]->(t3)
CREATE (t1)-[:CONNECTS_TO {mode: 'road', cost: 8, time: 2}]->(w2)
CREATE (t2)-[:CONNECTS_TO {mode: 'sea', cost: 18, time: 5}]->(w3)
CREATE (t3)-[:CONNECTS_TO {mode: 'air', cost: 22, time: 1}]->(w1)

// Cross-supply chain routes
CREATE (s1)-[:SUPPLIES {mode: 'road', cost: 22, time: 6}]->(m3)
CREATE (w1)-[:DISTRIBUTES {mode: 'sea', cost: 20, time: 4}]->(r3)
CREATE (w2)-[:DISTRIBUTES {mode: 'road', cost: 10, time: 3}]->(r1)
CREATE (w3)-[:DISTRIBUTES {mode: 'air', cost: 30, time: 2}]->(r2)
```



```
neo4j$ // Create Suppliers CREATE (s1:Supplier {name: 'Supplier 1', capacity: 1000}) CREATE (s2:Supplier {name: 'Suppl…
```

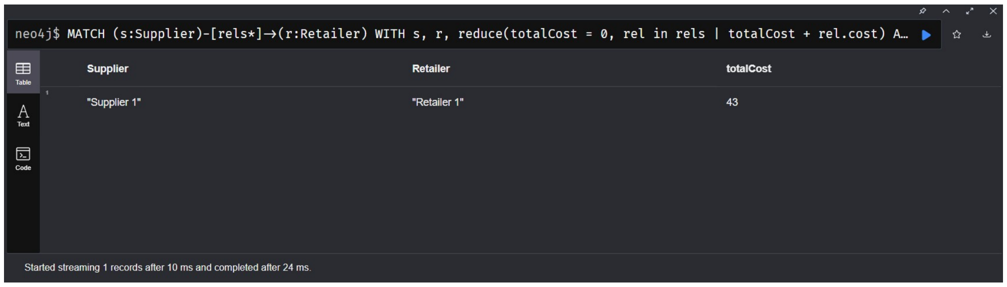
```
Added 15 labels, created 15 nodes, set 84 properties, created 19 relationships, completed after 69 ms.
```

```
Added 15 labels, created 15 nodes, set 84 properties, created 19 relationships, completed after 69 ms.
```

Figure 18: Complex experiment: step 1

Step 2: Find the shortest path by cost from Supplier to Retailer

```
MATCH (s:Supplier)-[rels*]->(r:Retailer)
WITH s, r, reduce(totalCost = 0, rel in rels | totalCost + rel.cost) AS totalCost
RETURN s.name AS Supplier, r.name AS Retailer, totalCost
ORDER BY totalCost ASC
LIMIT 1;
```



```
neo4j$ MATCH (s:Supplier)-[rels*]→(r:Retailer) WITH s, r, reduce(totalCost = 0, rel in rels | totalCost + rel.cost) A…
```

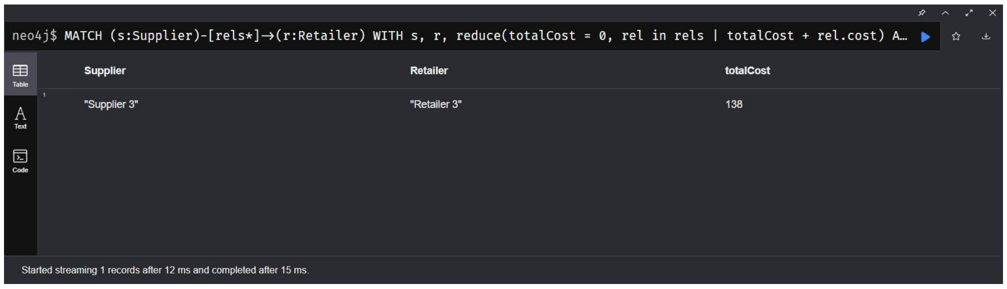
Supplier	Retailer	totalCost
"Supplier 1"	"Retailer 1"	43

```
Started streaming 1 records after 10 ms and completed after 24 ms.
```

Figure 19: Complex experiment: step 2

Step 3: Find the longest path by cost from Supplier to Retailer

```
MATCH (s:Supplier)-[rels*]->(r:Retailer)
WITH s, r, reduce(totalCost = 0, rel in rels | totalCost + rel.cost) AS totalCost
RETURN s.name AS Supplier, r.name AS Retailer, totalCost
ORDER BY totalCost DESC
LIMIT 1;
```



```
neo4j$ MATCH (s:Supplier)-[rels*]→(r:Retailer) WITH s, r, reduce(totalCost = 0, rel in rels | totalCost + rel.cost) A…
```

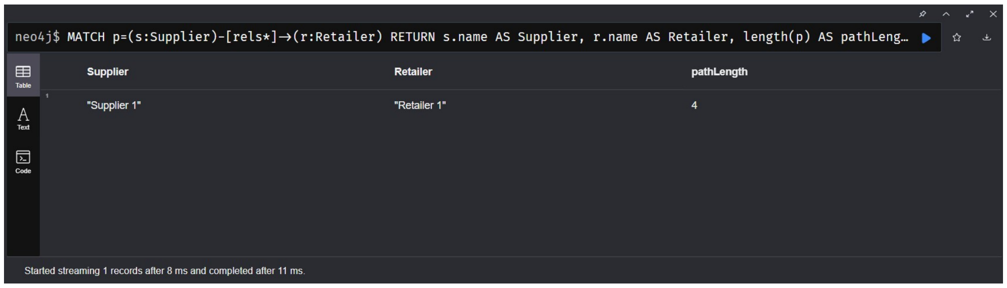
Supplier	Retailer	totalCost
"Supplier 3"	"Retailer 3"	138

Started streaming 1 records after 12 ms and completed after 15 ms.

Figure 20: Complex experiment: step 3

Step 4: Find the longest physical path (most hops) from Supplier to Retailer

```
MATCH p=(s:Supplier)-[rels*]->(r:Retailer)
RETURN s.name AS Supplier, r.name AS Retailer, length(p) AS pathLength
ORDER BY pathLength DESC
LIMIT 1;
```



```
neo4j$ MATCH p=(s:Supplier)-[rels*]→(r:Retailer) RETURN s.name AS Supplier, r.name AS Retailer, length(p) AS pathLeng…
```

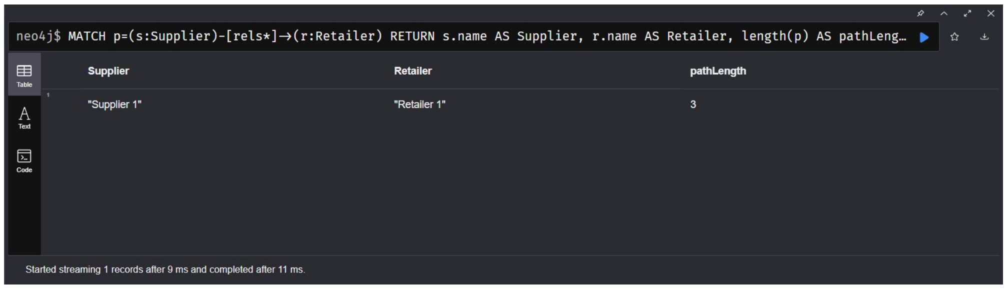
Supplier	Retailer	pathLength
"Supplier 1"	"Retailer 1"	4

Started streaming 1 records after 8 ms and completed after 11 ms.

Figure 21: Complex experiment: step 4

Step 5: Find the shortest physical path (fewest hops) from Supplier to Retailer

```
MATCH p=(s:Supplier)-[rels*]->(r:Retailer)
RETURN s.name AS Supplier, r.name AS Retailer, length(p) AS pathLength
ORDER BY pathLength ASC
LIMIT 1;
```



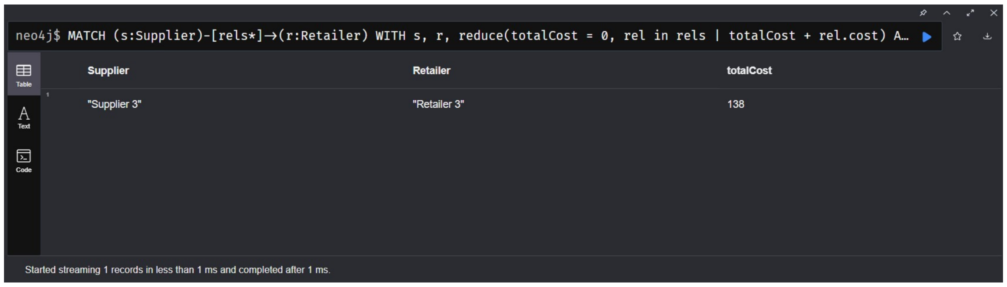
```
neo4j$ MATCH p=(s:Supplier)-[rels*]→(r:Retailer) RETURN s.name AS Supplier, r.name AS Retailer, length(p) AS pathLeng…
```

```
  Supplier        Retailer        pathLength
1 "Supplier 1"    "Retailer 1"    3

Started streaming 1 records after 9 ms and completed after 11 ms.
```

Figure 22: Complex experiment: step 5

Step 6: Find the farthest path by time from Supplier to Retailer

```
// Find the longest path by cost from Supplier to Retailer
MATCH (s:Supplier)-[rels*]->(r:Retailer)
WITH s, r, reduce(totalCost = 0, rel in rels | totalCost + rel.cost) AS totalCost
RETURN s.name AS Supplier, r.name AS Retailer, totalCost
ORDER BY totalCost DESC
LIMIT 1;
```



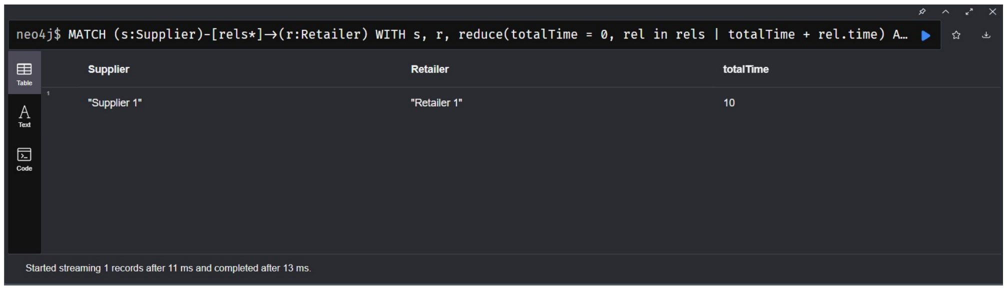
```
neo4j$ MATCH (s:Supplier)-[rels*]→(r:Retailer) WITH s, r, reduce(totalCost = 0, rel in rels | totalCost + rel.cost) A…
```

```
  Supplier        Retailer        totalCost
1 "Supplier 3"    "Retailer 3"    138

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

Figure 23: Complex experiment: step 6

Step 7: Find the fastest route by time from Supplier to Retailer

```
MATCH (s:Supplier)-[rels*]->(r:Retailer)
WITH s, r, reduce(totalTime = 0, rel in rels | totalTime + rel.time) AS totalTime
RETURN s.name AS Supplier, r.name AS Retailer, totalTime
ORDER BY totalTime ASC
LIMIT 1;
```



```
neo4j$ MATCH (s:Supplier)-[rels*]→(r:Retailer) WITH s, r, reduce(totalTime = 0, rel in rels | totalTime + rel.time) A…
```

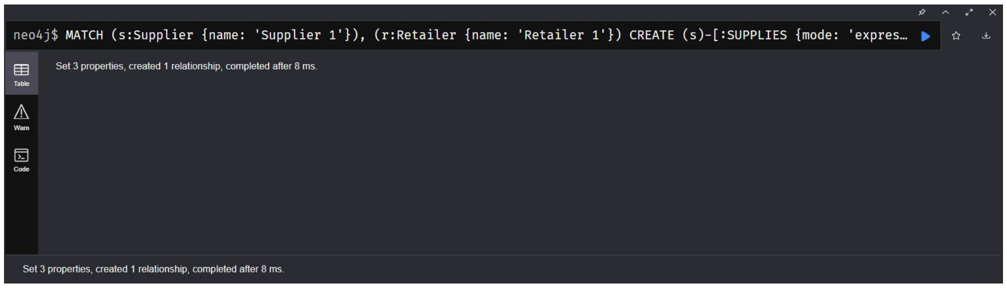
```
Supplier        Retailer        totalTime
"Supplier 1"    "Retailer 1"    10
```

```
Started streaming 1 records after 11 ms and completed after 13 ms.
```

Figure 24: Complex experiment: step 7

Step 8: Create a new faster path relation

```
MATCH (s:Supplier {name: 'Supplier 1'}), (r:Retailer {name: 'Retailer 1'})
CREATE (s)-[:SUPPLIES {mode: 'express', cost: 15, time: 1}]->(r);
```



```
neo4j$ MATCH (s:Supplier {name: 'Supplier 1'}), (r:Retailer {name: 'Retailer 1'}) CREATE (s)-[:SUPPLIES {mode: 'expres…
```

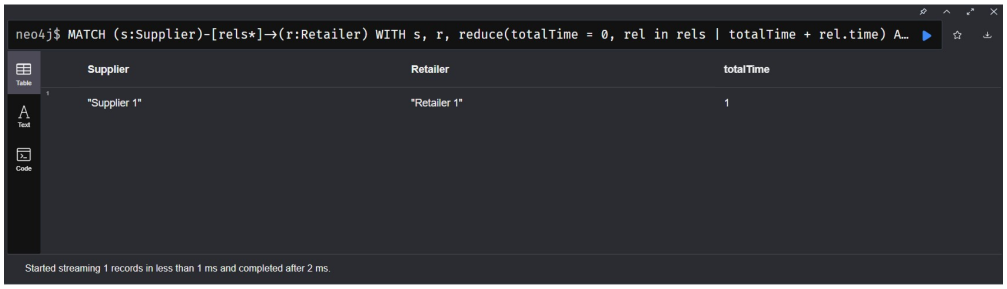
```
Set 3 properties, created 1 relationship, completed after 8 ms.
```

```
Set 3 properties, created 1 relationship, completed after 8 ms.
```

Figure 25: Complex experiment: step 8

Step 9: Find the fastest path by time from Supplier to Retailer

```
MATCH (s:Supplier)-[rels*]->(r:Retailer)
WITH s, r, reduce(totalTime = 0, rel in rels | totalTime + rel.time) AS totalTime
RETURN s.name AS Supplier, r.name AS Retailer, totalTime
ORDER BY totalTime ASC
LIMIT 1;
```



```
neo4j$ MATCH (s:Supplier)-[rels*]→(r:Retailer) WITH s, r, reduce(totalTime = 0, rel in rels | totalTime + rel.time) A…
```

```
Supplier        Retailer        totalTime
"Supplier 1"    "Retailer 1"    1
```

```
Started streaming 1 records in less than 1 ms and completed after 2 ms.
```

Figure 26: Complex experiment: step 9