Experiment No. 6

Title: Execution of graph base database

Batch:SYIT(B3) **Roll No.:16010423076** **Experiment No.:6**

Aim: To execute NOSQL database using Neo4j

Resources needed: Neo4j

Theory

A graph data structure consists of **nodes** (discrete objects) that can be connected by **relationships**. Below is the image of a graph with three nodes (the circles) and three relationships (the arrows).

Figure 1. Concept of a graph structure

The Neo4j property graph database model consists of:

- Nodes describe entities (discrete objects) of a domain.
- Nodes can have zero or more labels to define (classify) what kind of nodes they are.
- Relationships describe a connection between a source node and a target node.
- Relationships always have a direction (one direction).
- Relationships must have a type (one type) to define (classify) what type of relationship they are.
- Nodes and relationships can have properties (key-value pairs), which further describe them.

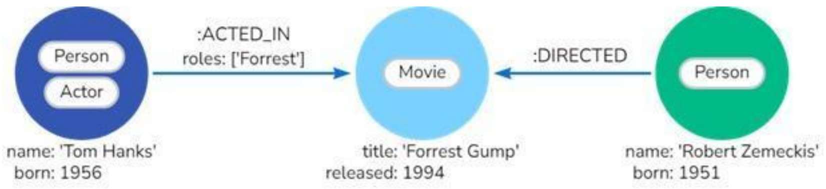


Figure 1. Example of graph

CREATE (:Person:Actor {name: 'Tom Hanks', born: 1956})-[:ACTED_IN {roles: ['Forrest']}]->(:Movie {title: 'Forrest Gump', released: 1994})<-[:DIRECTED]-(:Person {name: 'Robert Zemeckis', born: 1951})

Node

Nodes are used to represent entities (discrete objects) of a domain.

The simplest possible graph is a single node with no relationships. Consider the following graph, consisting of a single node.

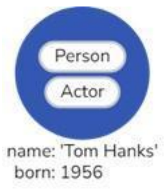


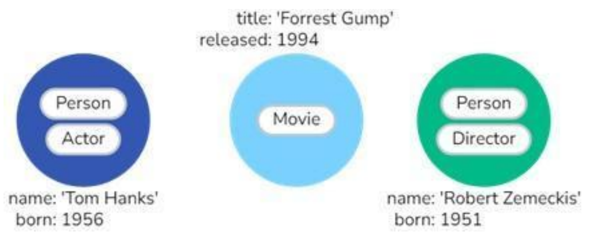
Figure 2. Node

The node labels are:	The properties are:
Person	name: Tom Hanks
Actor	born: 1956

The node can be created with Cypher using the query:

```
CREATE (:Person:Actor {name: 'Tom Hanks', born: 1956})
```

In the example graph, the node labels, Person, Actor, and Movie, are used to describe (classify) the nodes. More labels can be added to express different dimensions of the data.



A relationship:

- Connects a source node and a target node.
- Has a direction (one direction).
- Must have a type (one type) to define (classify) what type of relationship it is.
- Can have properties (key-value pairs), which further describe the relationship.

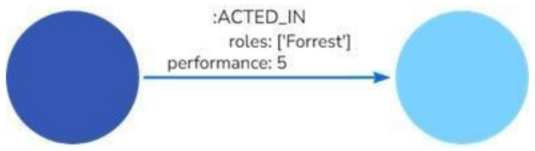


Figure 3. Relationship

The relationship type: ACTED_IN

The properties are:

roles: ['Forrest']
performance: 5

The roles property has an array value with a single item ('Forrest') in it.

The relationship can be created with Cypher using the query:

```
CREATE ()-[:ACTED_IN {roles: ['Forrest'], performance: 5}]->()
```

The value part of a property:

Can hold different data types, such as number, string, or boolean.
Can hold a homogeneous list (array) containing, for example, strings, numbers, or boolean values.
Example 1. Number

```
CREATE (:Example {a: 1, b: 3.14})
```

Example 2. String and boolean

```
CREATE (:Example {c: 'This is an example string', d: true, e: false})
```

Example 3. Lists
CREATE (:Example {f: [1, 2, 3], g: [2.71, 3.14], h: ['abc', 'example'], i: [true, true, false]})

Traversals and paths
Traversing a graph means visiting nodes by following relationships according to some rules. In most cases only a subset of the graph is visited.
Example 4. Path matching.
To find out which movies Tom Hanks acted in according to the tiny example database, the traversal would start from the Tom Hanks node, follow any ACTED_IN relationships connected to the node, and end up with the Movie node Forrest Gump as the result (see the black lines):

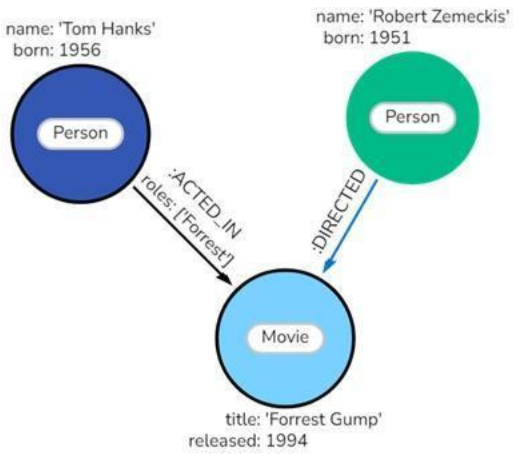


Figure 4. Path matching
The traversal result could be returned as a path with the length 1:

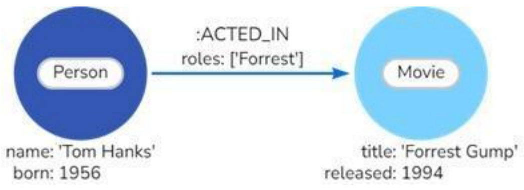


Figure 5. Relationship type

The shortest possible path has length zero. It contains a single node and no relationships.

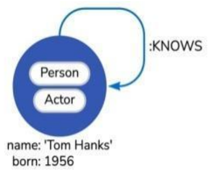


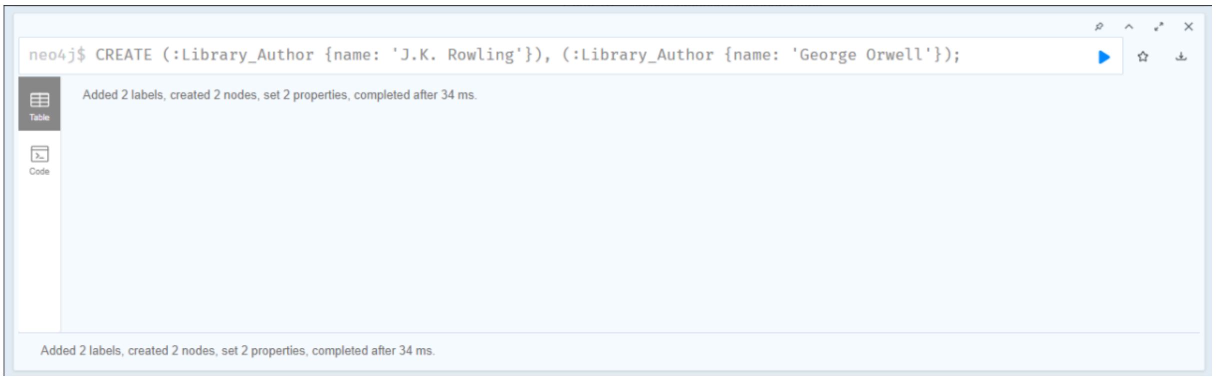
Figure 6. Path of length one

Results: (Program printout with output)

Step 1 : Create Nodes

Create Authors

```
CREATE (:Author {name: 'J.K. Rowling'}),
       (:Author {name: 'George Orwell'});
```



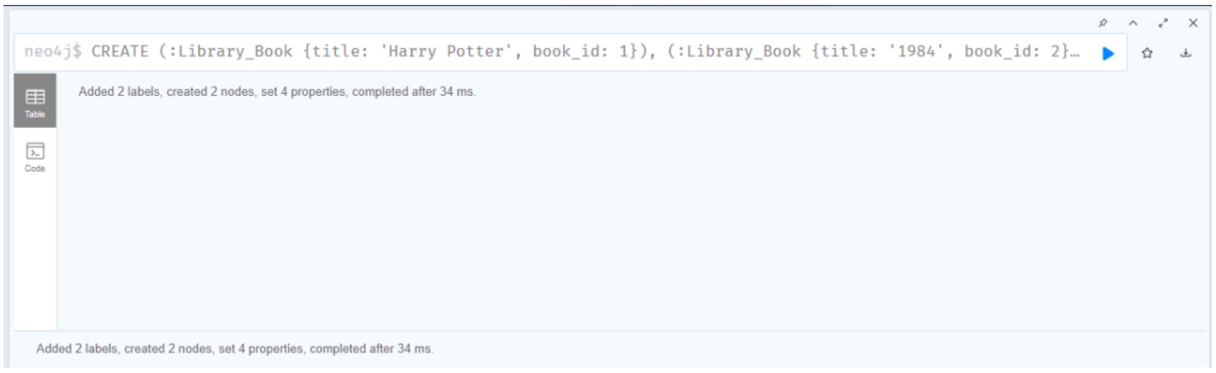
```
neo4j$ CREATE (:Library_Author {name: 'J.K. Rowling'}), (:Library_Author {name: 'George Orwell'});
```

```
Added 2 labels, created 2 nodes, set 2 properties, completed after 34 ms.
```

```
Added 2 labels, created 2 nodes, set 2 properties, completed after 34 ms.
```

Create Books

```
CREATE (:Book {title: 'Harry Potter', book_id: 1}),
       (:Book {title: '1984', book_id: 2});
```



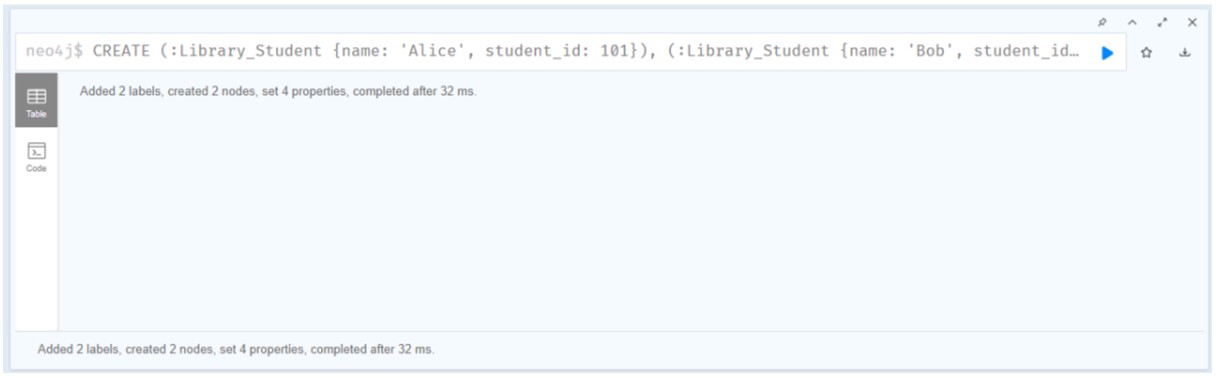
```
neo4j$ CREATE (:Library_Book {title: 'Harry Potter', book_id: 1}), (:Library_Book {title: '1984', book_id: 2}…
```

```
Added 2 labels, created 2 nodes, set 4 properties, completed after 34 ms.
```

```
Added 2 labels, created 2 nodes, set 4 properties, completed after 34 ms.
```

Create Students

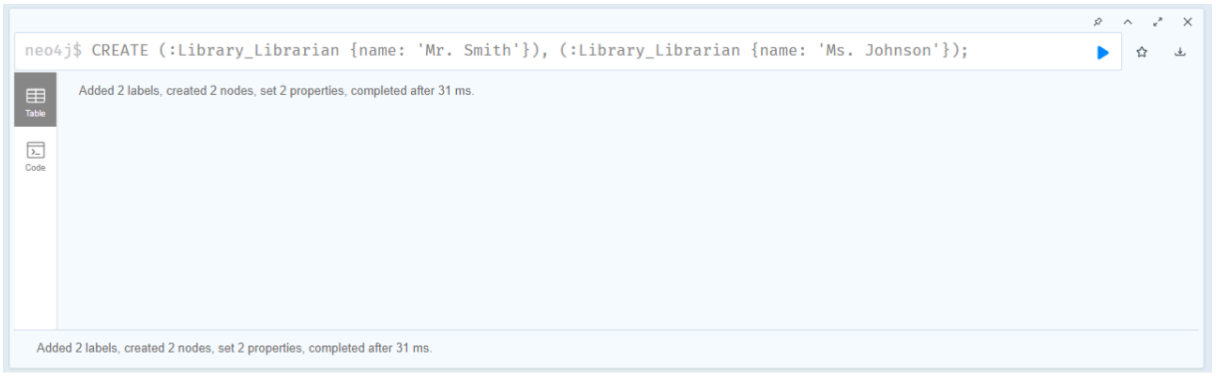
```
CREATE (:Student {name: 'Alice', student_id: 101}),
       (:Student {name: 'Bob', student_id: 102});
```



```
neo4j$ CREATE (:Library_Student {name: 'Alice', student_id: 101}), (:Library_Student {name: 'Bob', student_id…
Added 2 labels, created 2 nodes, set 4 properties, completed after 32 ms.
Added 2 labels, created 2 nodes, set 4 properties, completed after 32 ms.
```

Create Librarians

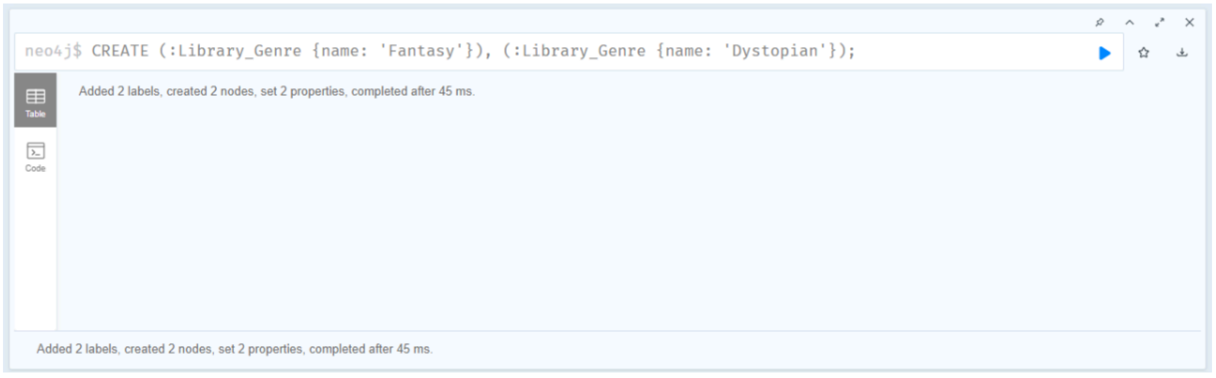
```
CREATE (:Librarian {name: 'Mr. Smith'}),
       (:Librarian {name: 'Ms. Johnson'});
```



```
neo4j$ CREATE (:Library_Librarian {name: 'Mr. Smith'}), (:Library_Librarian {name: 'Ms. Johnson'});
Added 2 labels, created 2 nodes, set 2 properties, completed after 31 ms.
Added 2 labels, created 2 nodes, set 2 properties, completed after 31 ms.
```

Create Genres

```
CREATE (:Genre {name: 'Fantasy'}),
       (:Genre {name: 'Dystopian'});
```



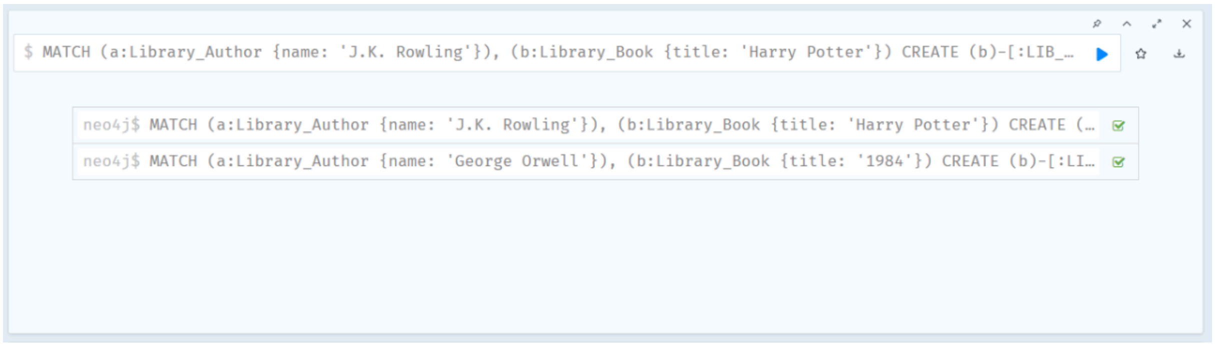
```
neo4j$ CREATE (:Library_Genre {name: 'Fantasy'}), (:Library_Genre {name: 'Dystopian'});
Added 2 labels, created 2 nodes, set 2 properties, completed after 45 ms.
Added 2 labels, created 2 nodes, set 2 properties, completed after 45 ms.
```

Step 2 : Create Relationships

Books Written by Authors

```
MATCH (a:Author {name: 'J.K. Rowling'}), (b:Book {title:
'Harry Potter'})
CREATE (b)-[:WRITTEN_BY]->(a);
```

```
MATCH (a:Author {name: 'George Orwell'}), (b:Book {title:
'1984'})
CREATE (b)-[:WRITTEN_BY]->(a);
```

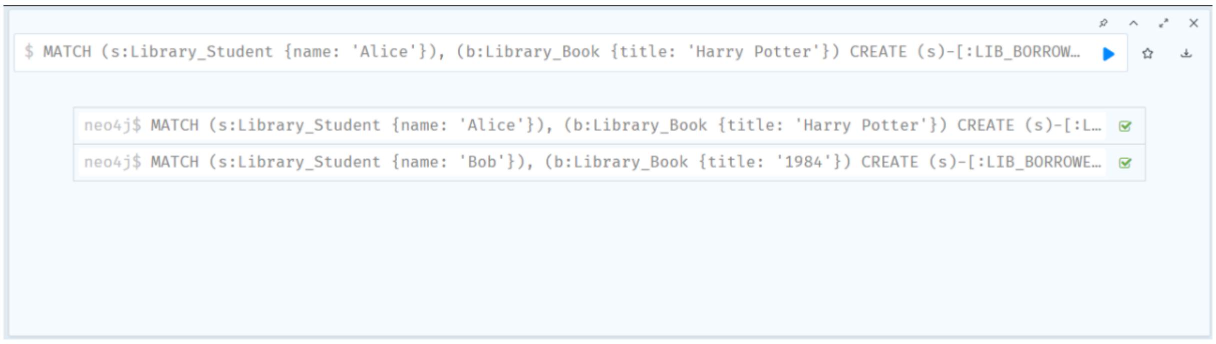


```
$ MATCH (a:Library_Author {name: 'J.K. Rowling'}), (b:Library_Book {title: 'Harry Potter'}) CREATE (b)-[:LIB_…
neo4j$ MATCH (a:Library_Author {name: 'J.K. Rowling'}), (b:Library_Book {title: 'Harry Potter'}) CREATE (…
neo4j$ MATCH (a:Library_Author {name: 'George Orwell'}), (b:Library_Book {title: '1984'}) CREATE (b)-[:LI…
```

Students Borrow Books

```
MATCH (s:Student {name: 'Alice'}), (b:Book {title: 'Harry
Potter'})
CREATE (s)-[:BORROWED_BY]->(b);

MATCH (s:Student {name: 'Bob'}), (b:Book {title: '1984'})
CREATE (s)-[:BORROWED_BY]->(b);
```

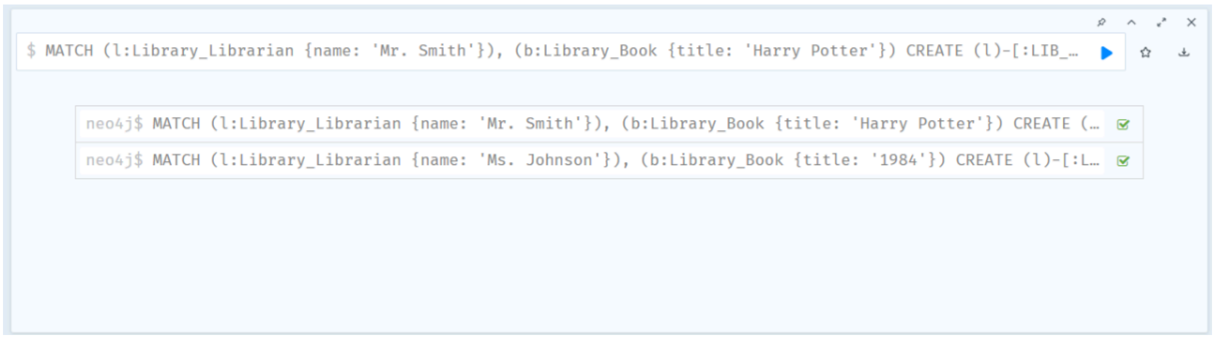


```
$ MATCH (s:Library_Student {name: 'Alice'}), (b:Library_Book {title: 'Harry Potter'}) CREATE (s)-[:LIB_BORROW…
neo4j$ MATCH (s:Library_Student {name: 'Alice'}), (b:Library_Book {title: 'Harry Potter'}) CREATE (s)-[:L…
neo4j$ MATCH (s:Library_Student {name: 'Bob'}), (b:Library_Book {title: '1984'}) CREATE (s)-[:LIB_BORROWE…
```

Librarians Manage Books

```
MATCH (l:Librarian {name: 'Mr. Smith'}), (b:Book {title:
'Harry Potter'})
CREATE (l)-[:MANAGED_BY]->(b);

MATCH (l:Librarian {name: 'Ms. Johnson'}), (b:Book {title:
'1984'})
CREATE (l)-[:MANAGED_BY]->(b);
```

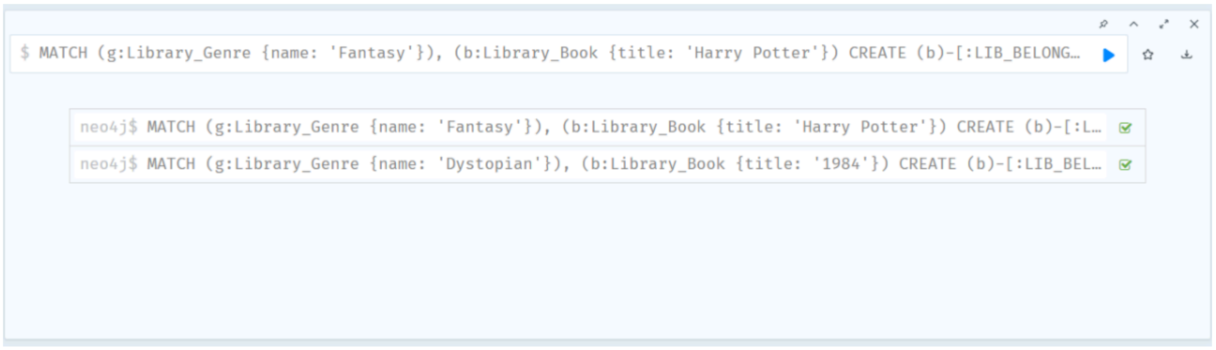


```
$ MATCH (l:Library_Librarian {name: 'Mr. Smith'}), (b:Library_Book {title: 'Harry Potter'}) CREATE (l)-[:LIB_…
neo4j$ MATCH (l:Library_Librarian {name: 'Mr. Smith'}), (b:Library_Book {title: 'Harry Potter'}) CREATE (…
neo4j$ MATCH (l:Library_Librarian {name: 'Ms. Johnson'}), (b:Library_Book {title: '1984'}) CREATE (l)-[:L…
```

Books Belong to Genres

```
MATCH (g:Genre {name: 'Fantasy'}), (b:Book {title: 'Harry
Potter'})
CREATE (b)-[:BELONGS_TO]->(g);

MATCH (g:Genre {name: 'Dystopian'}), (b:Book {title: '1984'})
CREATE (b)-[:BELONGS_TO]->(g);
```



```
$ MATCH (g:Library_Genre {name: 'Fantasy'}), (b:Library_Book {title: 'Harry Potter'}) CREATE (b)-[:LIB_BELONG…

neo4j$ MATCH (g:Library_Genre {name: 'Fantasy'}), (b:Library_Book {title: 'Harry Potter'}) CREATE (b)-[:L…
neo4j$ MATCH (g:Library_Genre {name: 'Dystopian'}), (b:Library_Book {title: '1984'}) CREATE (b)-[:LIB_BEL…
```

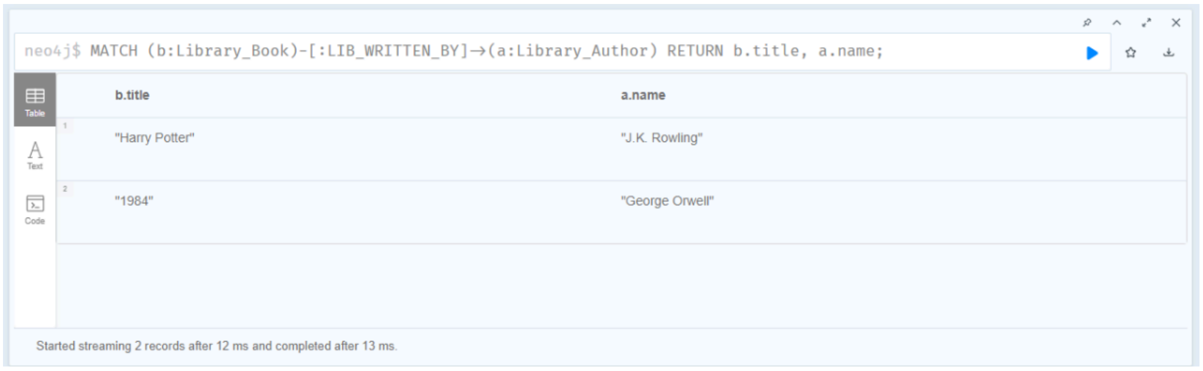
Step 3 : Query the Graph for Visualization

View the Entire Graph

```
MATCH (n) RETURN n;
```

Find All Books and Their Authors

```
MATCH (b:Book)-[:WRITTEN_BY]->(a:Author)
RETURN b.title, a.name;
```



```
neo4j$ MATCH (b:Library_Book)-[:LIB_WRITTEN_BY]→(a:Library_Author) RETURN b.title, a.name;
```

	b.title	a.name
1	"Harry Potter"	"J.K. Rowling"
2	"1984"	"George Orwell"

Started streaming 2 records after 12 ms and completed after 13 ms.

Find Who Borrowed Each Book

```
MATCH (s:Student)-[:BORROWED_BY]->(b:Book)
RETURN s.name AS Student, b.title AS Book;
```

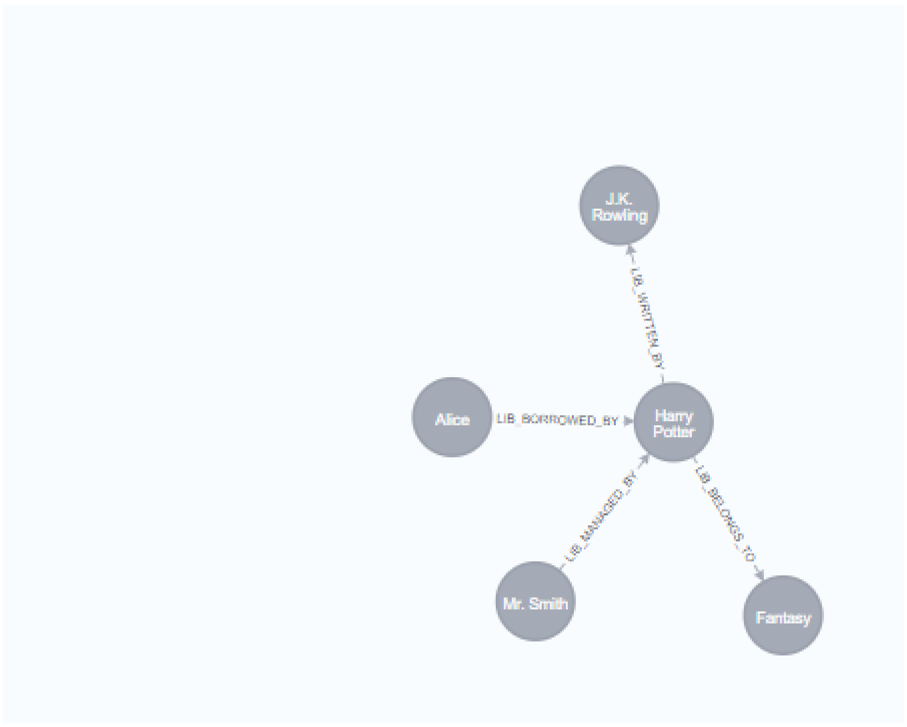
Find Who Manages Which Book

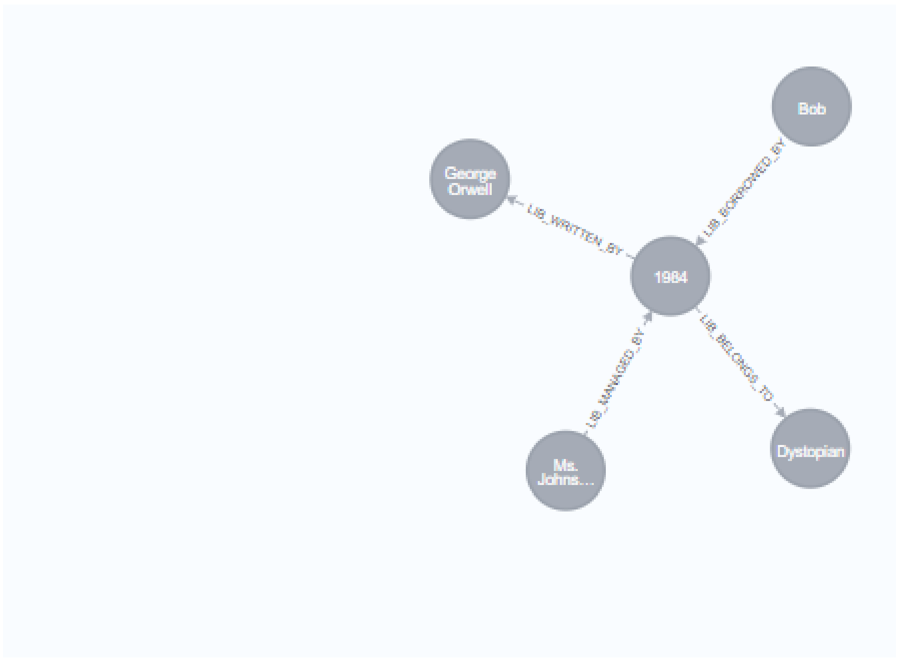
```
MATCH (l:Librarian)-[:MANAGED_BY]->(b:Book)
RETURN l.name AS Librarian, b.title AS Book;
```

Find Books That Belong to a Genre

```
MATCH (b:Book)-[:BELONGS_TO]->(g:Genre)
```

```
RETURN b.title, g.name;
```





Questions:

1. Why does Neo4j enforce directionality in relationships? Can an undirected graph be represented in Neo4j?

Neo4j uses directed relationships because it makes queries faster and more efficient. Instead of checking connections both ways, it knows exactly where to go. If you need an undirected graph, you can create relationships in both directions or use Cypher queries (`MATCH (a)-[:FRIENDS]-(b)`) that don't care about direction.

2. What are graph traversals, and how does Neo4j optimize them for performance?

Graph traversal is like following a treasure map—starting from one node and moving through connections to find what you need. Neo4j optimizes this by using index-free adjacency, meaning each node directly knows its neighbors, so searches are super fast compared to traditional databases that scan huge tables.

3. Compare and contrast graph databases like Neo4j with relational databases. When would you prefer one over the other?
Neo4j (Graph DB): Great for highly connected data, like social networks, recommendation engines, or fraud detection. It handles relationships naturally.
Relational DBs (SQL): Best for structured, tabular data, like banking records or inventory systems. They use tables and predefined schemas.
If your problem is all about relationships, go with Neo4j. If it's about well-structured data, stick with SQL.

4. What is the significance of labels in Neo4j, and how do they help in query performance?
Labels are like tags for nodes. Instead of searching the entire database, you can filter by labels (e.g., `MATCH (p:Person) WHERE p.name = "Alice"`), making queries faster and more manageable. They also help with indexing, which boosts speed.

5. Explain how indexing works in Neo4j. How does it improve query efficiency?
Indexes in Neo4j act like speed boosters for searches. Instead of scanning every node, an index helps the database jump straight to relevant data. For example, if you index `name`, searching for `"Alice"` is much faster. Neo4j automatically creates indexes for primary keys and lets you add more for better performance.

Outcomes:
CO2: Design advanced database systems using In-memory,Spatial and NOSQL databases and its implementation.

Conclusion: (Conclusion to be based on outcomes achieved)
From Experiment Number 6, I learned how to work with Neo4j, a NoSQL graph database that efficiently handles relationships between data. By creating nodes, defining relationships, and executing Cypher queries, I explored how Neo4j optimizes graph traversals and indexing to improve query performance. Understanding the importance of labels, relationships, and indexing gave me insights into why graph databases excel at managing connected data. This hands-on experience reinforced how graph databases differ from relational databases and when to choose one over the other. Overall, this experiment deepened my understanding of NoSQL databases and their real-world applications.

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of faculty in-charge with date

References:

1. Elmasri and Navathe, “Fundamentals of Database Systems”, Pearson Education

2. Raghu Ramakrishnan and Johannes Gehrke, “Database Management Systems” 3rd Edition, McGraw Hill,2002

3. Korth, Silberchatz, Sudarshan, “Database System Concepts” McGraw Hill

4. http://www.bostongis.com/PrinterFriendly.aspx?content_name=postgis_tut01

5. https://neo4j.com/download/