

Student name: Pothumulla Kankanamge Mewan Madhusha

NB! Grading system. Assignment contains 15 tasks. Each task completion brings you 1 point regardless of complexity. If task completed partially grade might be smaller, for example, 0.5 for half of task completed. Maximum number of points to earn for this assignment is 15.

Neo4J task.

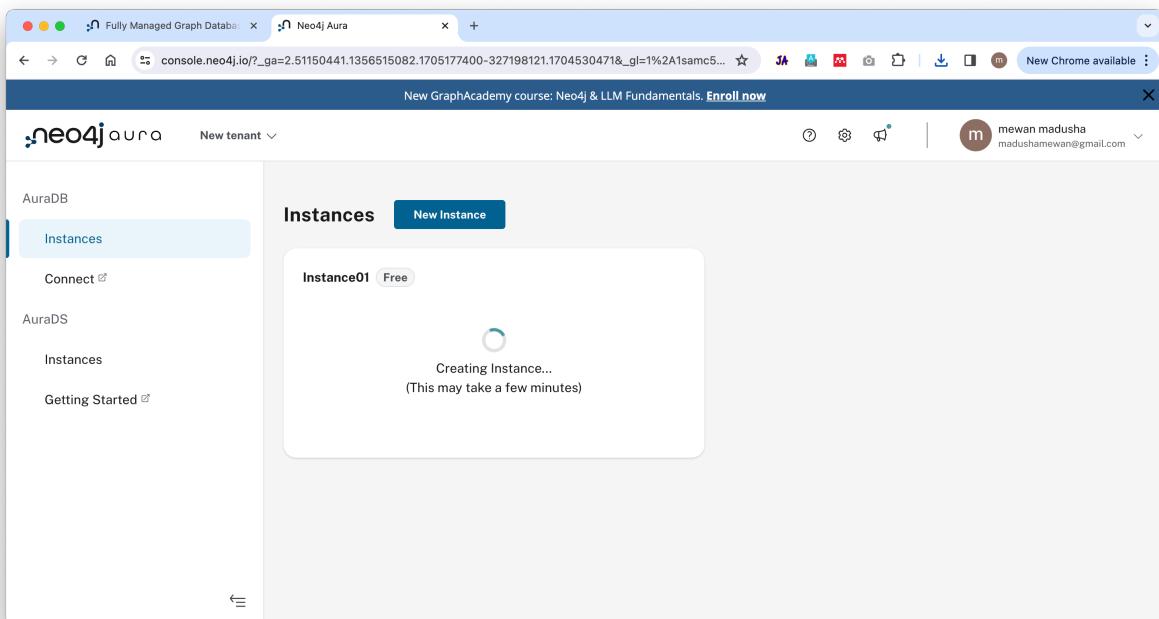
TASK 1.

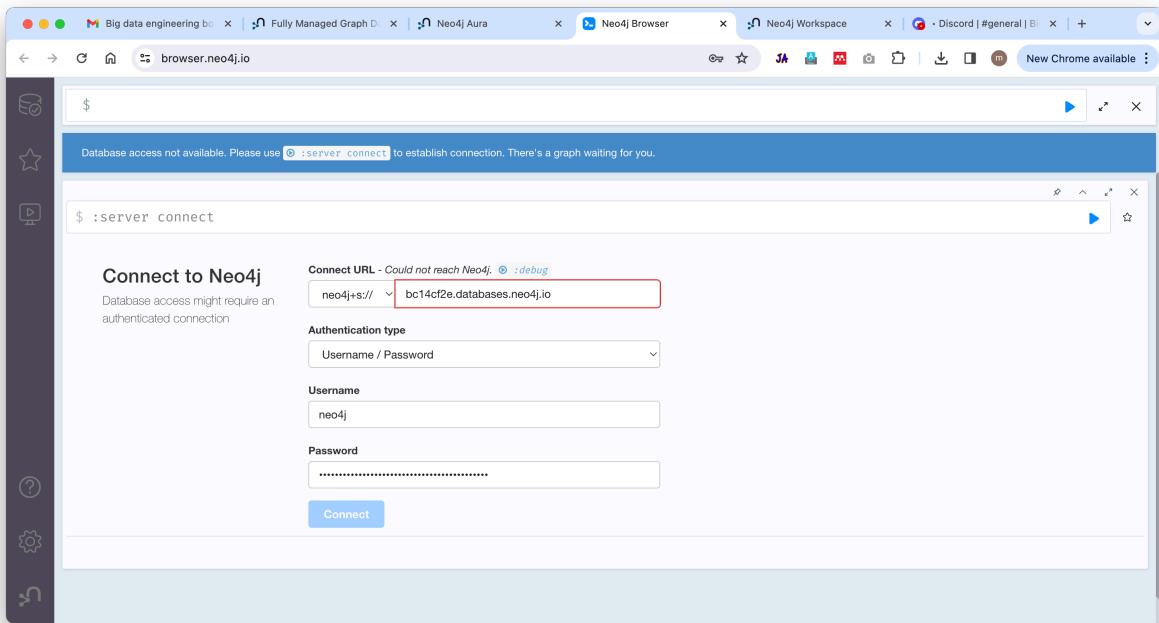
1. Go to neo4j.io
2. Set up Neo4j free account to use Aura DB.
3. Create new AuraDB instance with Movies dataset.

Alternative: Download and install Desktop version, then download free Movies dataset.

Insert screenshot of completed assignment here:

Creating Free AuraDB account





Create Movie dataset

Movie Graph Guide

Create
Create the movie graph

Use the following code block to create the movie graph. It contains a single Cypher query statement composed of multiple `CREATE` clauses.

NOTE: This guide assumes that you use an empty graph. If it contains data, see page 9 on how to clean it up.

- Click this code block and bring it into the Editor:

```
released:1999, tagline:'Welcome to the Real World'}
    CREATE (Keanu:Person {name:'Keanu Reeves', born:1964})
    CREATE (Carrie:Person {name:'Carrie-Anne Moss', born:1967})
    CREATE (Laurence:Person {name:'Laurence Fishburne', born:1961})
    CREATE (Hugo:Person {name:'Hugo Weaving', born:1960})
    CREATE (Lilly:Person {name:'Lilly Wachowski', born:1967})
```

- Run the Cypher code by clicking the **Run** button.
- Wait for the operation to finish.

neo4j\$

To help make Neo4j Browser better we collect information on product usage. Review your [settings](#) at any time.

`neo4j$ CREATE (TheMatrix:Movie {title:'The Matrix', released:1999, tagline:'Welcome to the Real World'})`

Graph

Overview

Node labels
* (17) Person (9) Movie (8)

Relationship types
* (18) DIRECTED (10) ACTED_IN (8)

Displaying 17 nodes, 0 relationships.

(points: 1)

TASK 2.

Refresh your impression of Neo4j with https://youtu.be/_D19h5s73Co

We'll be taking advantage of built-in learning materials. Type these commands in console and read information there:

\$:play intro

The screenshot shows the Neo4j Browser interface with the following details:

- Database Information:** Shows the database is set to "neo4j".
- Node labels:** Shows three selected: "Movie" (orange), "Person" (purple), and "ACTED_IN" (grey).
- Relationship types:** Shows six selected: "ACTED_IN", "DIRECTED", "FOLLOWS", "PRODUCED", "REVIEWED", and "WROTE".
- Property keys:** Shows several selected: "born", "data", "id", "name", "nodes", "rating", "relationships", "released", "roles", "style", "summary", "tagline", "title", and "visualisation".
- Connected as:** Shows a "Graph" icon.
- Neo4j\$ Shell:**
 - Query: `\$:play intro`
 - Result: "Introduction" section with text about the Neo4j Browser User Interface.
 - Result: "Cypher" section with text about Neo4j's Cypher language.
 - Result: "Graph" visualization showing nodes for "Neo4j", "Movie", and "Person" with relationships.

\$:play cypher

The screenshot shows the Neo4j Browser interface with the following details:

- Database Information:** Shows the database is set to "neo4j".
- Node labels:** Shows three selected: "Movie" (orange), "Person" (purple), and "ACTED_IN" (grey).
- Relationship types:** Shows six selected: "ACTED_IN", "DIRECTED", "FOLLOWS", "PRODUCED", "REVIEWED", and "WROTE".
- Property keys:** Shows several selected: "born", "data", "id", "name", "nodes", "rating", "relationships", "released", "roles", "style", "summary", "tagline", "title", and "visualisation".
- Connected as:** Shows a "Graph" icon.
- Neo4j\$ Shell:**
 - Query: `\$:play cypher`
 - Result: "Cypher" section with text about Neo4j's Cypher language.
 - Result: "Introduction" section with text about the Neo4j Browser User Interface.

Now time for practice - navigate to the “Movie Graph” section:

\$:play movie-graph

The screenshot shows the Neo4j Browser interface with the URL `browser.neo4j.io`. On the left, there's a sidebar titled "Database Information" with sections for "Use database" (set to "neo4j"), "Node labels" (Movie, Person), "Relationship types" (ACTED_IN, DIRECTED, FOLLOWS, PRODUCED, REVIEWED, WROTE), and "Property keys" (born, data, id, name, nodes, rating, relationships, released, roles, style, summary, tagline, title, visualisation). Below these are "Connected as" options. The main area has a title "neo4j\$" and a command line interface with two tabs: "\$:play movie-graph" and "\$:play cypher". The "movie-graph" tab contains a guide for the "Movie Graph" application, which is described as a mini graph application containing actors and directors related through movies they've collaborated on. It lists tasks: Create (insert movie data), Find (retrieve individual movies and actors), Query (discover related actors and directors), and Solve (the Bacon Path). A warning message states: "WARNING: This guide will modify the data in the currently active database." The "cypher" tab contains a brief introduction to Cypher, stating it's Neo4j's graph query language that uses patterns to describe graph data and familiar SQL-like clauses.

We'll be working through the examples provided in this section for working with a graph of data relations between actors, directors and movies, with some additional tasks.

What is Cypher?

Cypher is a graph query language that is used to query the Neo4j Database. Just like you use SQL to query a MySQL database, you would use Cypher to query the Neo4j Database.

A simple cypher query can look something like this:

`Match (m:Movie) where m.released > 2000 RETURN m limit 5`

Please look at this video:

<https://www.youtube.com/watch?v=3sgNbgRjZSU>

Cypher tutorial starts automatically when you connect to Neo4j instance and choose to use Movies dataset.

Finish Cypher tutorial.

Try

1. Write a query to retrieve all the movies released after the year 2005.

```
① Match (m:Movie) where m.released > 2005 RETURN m
```

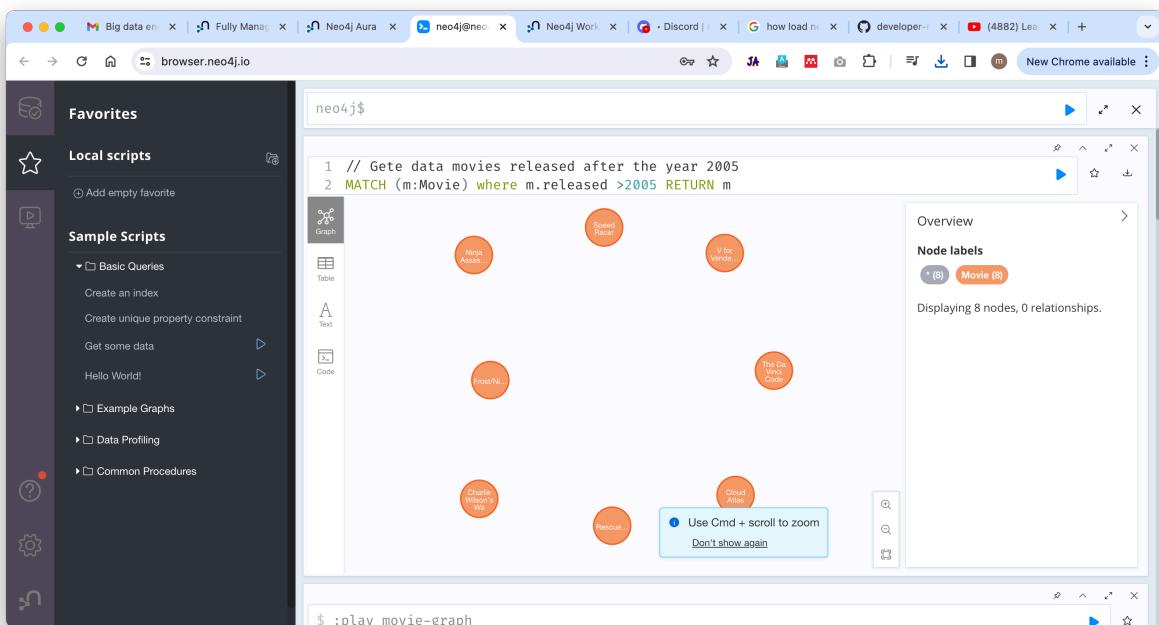
2. Write a query to return the count of movies released after the year 2005. (Hint: you can use the `count(m)` function to return the count)

```
① Match (m:Movie) where m.released > 2005 RETURN count(m)
```

Connected to Neo4j

Nice to meet you.

QUERY 01



QUERY 02

The screenshot shows the Neo4j Browser interface. On the left is a sidebar with 'Favorites', 'Local scripts', 'Sample Scripts' (including 'Basic Queries', 'Create an index', 'Create unique property constraint', 'Get some data', 'Hello World!', 'Example Graphs', 'Data Profiling', and 'Common Procedures'), and help/sync icons.

The main area has two tabs:

- neo4j\$**: Shows a table with one row labeled 'count(m)' containing the value '8'. Below it, a message says "Started streaming 1 records after 24 ms and completed after 28 ms."
- neo4j\$ // Gete data movies released after the year 2005 MATCH (m:Movie) where m.release...**: Shows a graph visualization with three nodes: 'Ninja Assassin' (orange), 'Speed Racer' (orange), and 'V for Vendetta' (orange). To the right, an 'Overview' panel shows 'Node labels' with '(8)' for both 'Ninja Assassin' and 'Movie (8)'. It also states "Displaying 8 nodes, 0 relationships."

What are advantages and disadvantages of Cypher

Advantages	Disadvantages
<p>Pattern matching - Easy to write queries that match specific structures within the graph.</p> <p>Readability</p> <p>Traversal and path finding - Cypher provides efficient syntax for traversing relationships and finding paths within the graph</p>	<p>Learning of Cypher – It take much time to deep dive into advanced query.</p> <p>Less support and difference in outside the graph database when considering neo4j and other systems.</p> <p>Performance consideration when writing queries (Different from each other)</p>

TASK 3.

What differences did you notice between CYPHER and SQL?

Paste your answer here: I have identify several differences in CYPHER and SQL in different criteria.

Data model	Cypher Designed for querying graph databases that use the property graph model but in SQL Designed for querying relational databases that use tables with rows and columns. It is well-suited for working with structured data
Syntax	Cypher syntax more similar to ASCII art-like syntax to represent patterns in the graph but in SQL it based on keywords and Sequence of using keywords
Usecase	Primarily used for querying graph databases where relationships are as important as the entities themselves. Commonly used in scenarios involving social networks, recommendation engines While SQL used for querying relational databases where structured tabular data is the norm. Commonly used in business applications

TASK 4.

Create and execute the query that returns name and year of birth of one person of your choosing.

Paste the query and output as an answer.

```
MATCH (p:Person {name: 'Laurence Fishburne'})
RETURN p.name AS Name, p.born AS YearOfBirth;
```

```

neo4j$ MATCH (p:Person {name: 'Laurence Fishburne'})  

    RETURN p.name AS Name, p.born AS YearOfBirth;  

neo4j$ // Gete data movies released after the year 2005, return the count MATCH (m:Movie...  

    count(m)

```

Name	YearOfBirth
"Laurence Fishburne"	1961

TASK 5.

Query DB and check in how many movies acted Tom Hanks, Hugo Weaving, Halle Berrie

```

MATCH (actor:Person)-[:ACTED_IN]->(movie:Movie)
WHERE actor.name IN ['Tom Hanks', 'Hugo Weaving', 'Halle Berry']
RETURN actor.name AS Actor, COUNT(movie) AS count_of_movies_acted;

```

```
neo4j$ MATCH (actor:Person)-[:ACTED_IN]-(movie:Movie)  
WHERE actor.name IN ['Tom Hanks', 'Hugo Weaving', 'Halle Berry']  
RETURN actor.name AS Actor, COUNT(movie) AS count_of_movies_acted;
```

Actor	count_of_movies_acted
"Hugo Weaving"	5
"Tom Hanks"	12
"Halle Berry"	1

Started streaming 3 records after 14 ms and completed after 19 ms.

```
neo4j$ MATCH (p:Person) return p
```

p
{ "identity": 1, }

Paste the query and output as an answer.

TASK 6.

How many movies are in DB?

MATCH (m:Movie) RETURN COUNT(m)

The screenshot shows the Neo4j Browser interface. On the left, there's a sidebar with sections for Favorites, Local scripts, Sample Scripts (including Basic Queries, Create an Index, Create unique property constraint, Get some data, Hello World!, Example Graphs, Data Profiling, and Common Procedures), and Help.

The main area has two query results:

```
neo4j$ MATCH (m:Movie) RETURN COUNT(m)
```

	COUNT(m)
1	38

Started streaming 1 records after 2 ms and completed after 3 ms.

```
neo4j$ MATCH (actor:Person)-[:ACTED_IN]-(movie:Movie) WHERE actor.name IN ['Tom Hanks' ...]
```

Actor	count_of_movies_acted
"Hugo Weaving"	5
"Tom Hanks"	12

Paste the query and output as an answer.

TASK 7.

How many actors are in DB?

MATCH (actor:Person) RETURN COUNT(actor)

The screenshot shows the Neo4j Browser interface. On the left, there's a sidebar with sections for Favorites, Local scripts, Sample Scripts (including Basic Queries, Create an Index, Create unique property constraint, Get some data, Hello World!, Example Graphs, Data Profiling, and Common Procedures), Help, Settings, and Logout. The main area has two tabs. The top tab contains the query: `neo4j$ MATCH (actor:Person) RETURN COUNT(actor)`. The result is a table with one row: COUNT(actor) = 133. Below this, it says "Started streaming 1 records after 2 ms and completed after 3 ms." The bottom tab contains the query: `neo4j$ MATCH (actor:Person)-[:ACTED_IN]-(movie:Movie) WHERE actor.name IN ['Tom Hanks' ...]`. The result is a table with two rows:

Actor	count_of_movies_acted
"Hugo Weaving"	5
"Tom Hanks"	12

Paste the query and output as an answer.

TASK 8.

How many actors played in Cloud Atlas?

Paste the query and output as an answer.

```
MATCH (:Movie {title: 'Cloud Atlas'})<-[ACTED_IN]-(actor:Person)
RETURN COUNT(DISTINCT actor) AS number_of_actress;
```

The screenshot shows the Neo4j Browser interface. On the left is a sidebar with sections for Favorites, Local scripts, Sample Scripts (including Basic Queries, Create an Index, Create unique property constraint, Get some data, Hello World!, Example Graphs, Data Profiling, and Common Procedures), Help, Settings, and Logout. The main area has two tabs. The top tab contains the query:

```
1 MATCH (:Movie {title:'Cloud Atlas'})->[:ACTED_IN]-(actor:Person)
2 RETURN COUNT(DISTINCT actor) AS number_of_actress;
```

The result table shows:

number_of_actress
4

Below the table, a message says "Started streaming 1 records after 11 ms and completed after 13 ms." The bottom tab contains the query:

```
neo4j$ MATCH (actor:Person) RETURN COUNT(actor)
```

The result table shows:

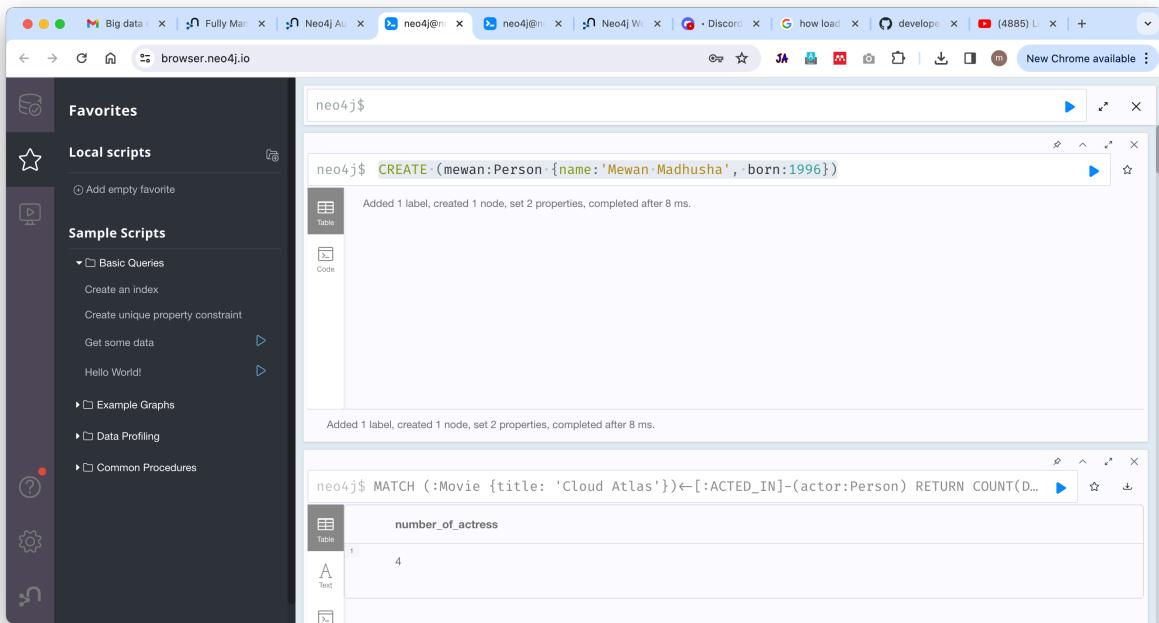
COUNT(actor)
133

TASK 9.

Add node with any new name to the Database.

Paste the query and output as an answer.

`CREATE (mewan:Person {name:'Mewan Madhusha', born:1996})`



TASK 10.

Add connection between user created name and favorite actor

Paste the query and output as an answer.

```
MATCH(actor:Person{name:"Keanu Reeves"})
MATCH (newUser:Person{name:'mewan madhusha'})
CREATE (newUser)-[:FOLLOWERS]->(actor)
```

The screenshot shows the Neo4j Browser interface. On the left, there's a sidebar with sections for Favorites, Local scripts (including Add empty favorite), Sample Scripts (Basic Queries, Example Graphs like Movie Graph and Northwind Graph, Data Profiling, and Common Procedures), Help, Settings, and Logout. The main area has two tabs: 'neo4j\$' and 'neo4j\$'. The top tab contains a query window with the following Cypher code:

```
1 MATCH(actor:Person{name:"Keanu Reeves"})
2 MATCH (newUser:Person{name:'mewan madhusha'})
3 CREATE (newUser)-[:FOLLOWS]-(actor)
```

Below the query window, there's a summary and response section. The summary shows the server version (Neo4j/5.15-aura) and address (bc14cf2e.databases.neo4j.io:7687). The response shows the query text and a note: "Created 1 relationship, completed after 42 ms." The bottom tab contains a query window with the following Cypher code:

```
neo4j$ MATCH (actor:Person{name:'mewan madhusha'}) RETURN actor
```

The results table shows one row with the label "actor" and an identity value of 174.

TASK 11.

Remove any actor from DB

Paste the query and output as an answer.

```
MATCH(actor:Person{name:"Christopher Guest"})
DETACH DELETE actor;
```

```
neo4j$ MATCH(actor:Person{name:"Christopher Guest"}) DETACH DELETE actor;
```

Deleted 1 node, deleted 1 relationship, completed after 20 ms.

```
neo4j$ MATCH(actor:Person{name:"Keanu Reeves"}) MATCH (newUser:Person{name:'mewan madhusha'}) CREATE (actor)-[:FOLLOWING]->(newUser)
```

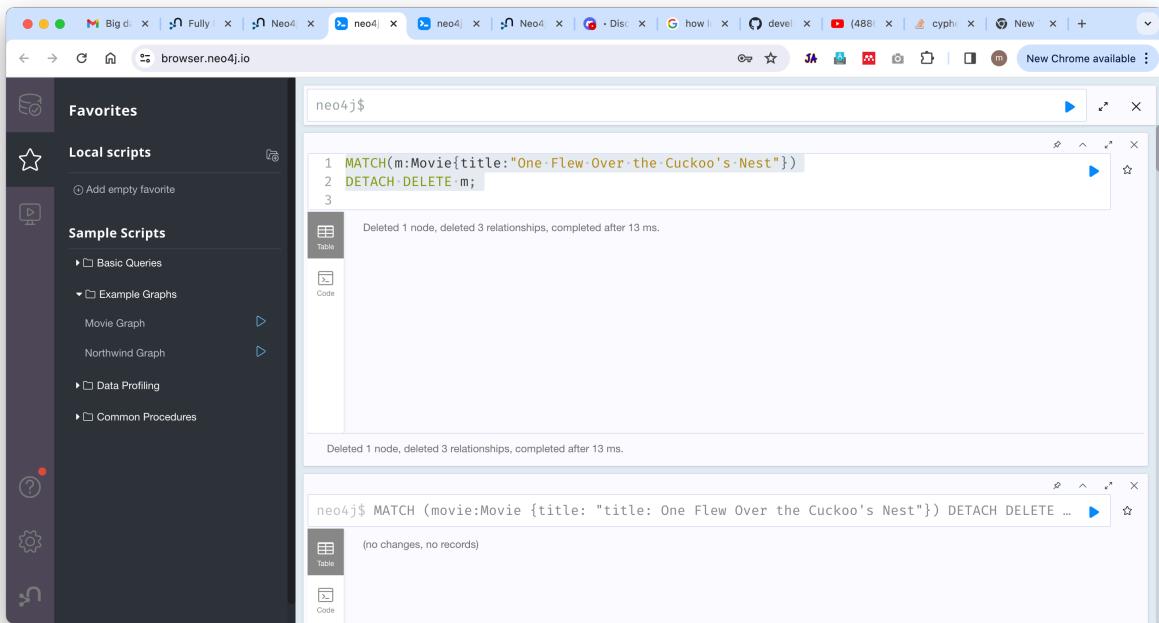
Server version: Neo4j/5.15-aura
Server address: bc14cf2e.databases.neo4j.io:7687
Query: MATCH(actor:Person{name:"Keanu Reeves"}) MATCH (newUser:Person{name:'mewan madhusha'}) CREATE (actor)-[:FOLLOWING]->(newUser)

TASK 12.

Remove any movie from DB

Paste the query and output as an answer.

```
MATCH(m:Movie{title:"One Flew Over the Cuckoo's Nest"})  
DETACH DELETE m;
```



TASK 13.

Extend graph with new information. To do so please attach node from Neo4j TASK to the database with the relationships. If relationships are not there yet, please create them.

Provide code and cypher output as the result.

```
MATCH(usermew:Person{name:'mewan madhusha'})  
CREATE(task:Task{tskname:"newtask"})  
CREATE (usermew)-[:ASSIGNED_TO]->(task)
```

browser.neo4j.io

Favorites

Local scripts

Add empty favorite

Sample Scripts

- Basic Queries
- Example Graphs
 - Movie Graph
 - Northwind Graph
- Data Profiling
- Common Procedures

neo4j\$

```

1 MATCH(usermew:Person{name:'mewan·madhusha'})-
2 CREATE(task:Task{tskname:"newtask"})
3 CREATE-(usermew)-[:ASSIGNED_TO]→(task)
  
```

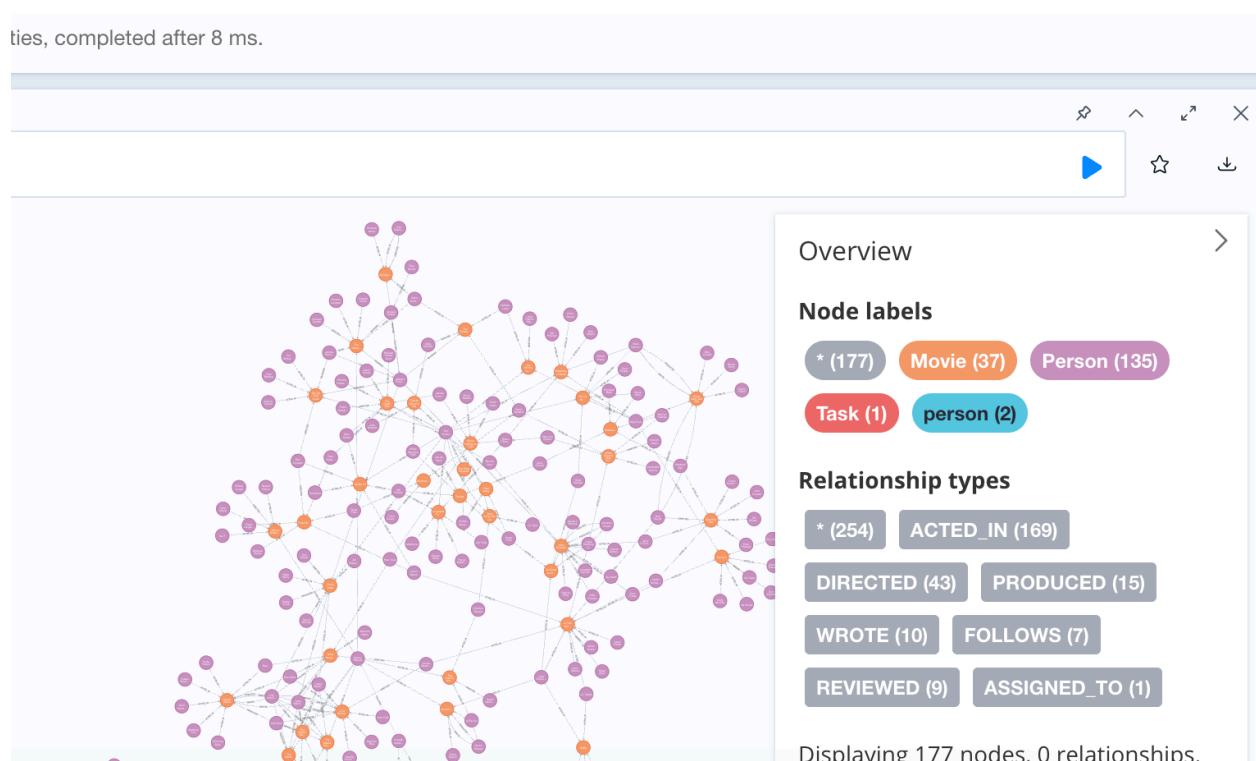
Added 1 label, created 1 node, set 1 property, created 1 relationship, completed after 25 ms.

neo4j\$ MATCH(task:Task {name: 'Initial task'}),(person:Person {name: 'mewan madhusha'}) MERG...

INFORMATION

This query builds a cartesian product between disconnected patterns.

ties, completed after 8 ms.



TASK 14.

Express a repeatable (idempotent) update operation. Check if movie Casablanca exists. If not exists - create one with corresponding relationships and add authors / actors as well. Movie should have at least one author and three actors.

NB! Use the most appropriate CYPHER feature to cover this task.

As a result please provide query and output. Explain why you have used particular clause(s).

```
MERGE(movie:Movie {title: 'casablanca'})
ON CREATE SET movie.released = 1942
WITH movie
MERGE (author:Person {name: 'peter1'})
MERGE (actor1:Person {name: 'ingrid2'})
MERGE (actor2:Person {name: 'paul3'})
MERGE (actor3:Person {name: 'bograt4'})

MERGE (author)-[:DIRECTED]->(movie)
MERGE (actor1)-[:ACTED_IN]->(movie)
MERGE (actor2)-[:ACTED_IN]->(movie)
MERGE (actor3)-[:ACTED_IN]->(movie);
```

```

neo4j$ 
1 MERGE (movie:Movie{title: 'casablanca'})
2 ON CREATE SET movie.released = 1942
3 WITH movie
4 MERGE (author:Person {name: 'peter1'})
5 MERGE (actor1:Person {name: 'ingrid2'})
6 MERGE (actor2:Person {name: 'paul3'})
7 MERGE (actor3:Person {name: 'bograt4'})
8
9 MERGE (author)-[:DIRECTED]→(movie)
10 MERGE (actor1)-[:ACTED_IN]→(movie)
11 MERGE (actor2)-[:ACTED_IN]→(movie)
12 MERGE (actor3)-[:ACTED_IN]→(movie);

```

Added 5 labels, created 5 nodes, set 6 properties, created 4 relationships, completed after 42 ms.

Added 5 labels, created 5 nodes, set 6 properties, created 4 relationships, completed after 42 ms.

The MERGE (movie:Movie {title: 'Casablanca'}) checks if the movie node with the title 'Casablanca' exists. If it doesn't, it creates one. And then adding released data properties also if the movie node created , to do that we can use The ON CREATE SET movie.year = 1942, It's create only if new movie create. Apart from that other merge statements ensure that author and actor nodes exist, and the relationships (DIRECTED and ACTED_IN) are created between them and the movie node.

TASK 15.

Build a Cypher statement to find the title and year of release for every :Movie that L.Wachowski has :DIRECTED. Try to use CYPHER practices you already know to identify proper director name.

Provide query and output.

```

MATCH (director:Person)
// WHERE director.name =~ 'L\\..*Wachowski'
WHERE (director.name) CONTAINS 'Wachowski'
MATCH (director)-[:DIRECTED]->(movie:Movie)
RETURN movie.title AS Title, movie.released AS YearOfRelease;

```

The screenshot shows the Neo4j browser interface. On the left, there's a sidebar with various icons and sections: 'Database Information' (selected), 'Use database' (set to 'neo4j'), 'Node labels' (Movie, Person, Task, person), 'Relationship types' (ACTED IN, ASSIGNED TO, DIRECTED, FOLLOWS, PRODUCED, REVIEWED, WROTE), and 'Property keys' (born, data, id, name, nodes, rating, relationships, released, roles, style, summary, tagline, title, tsname, visualisation). The main area is titled 'neo4j\$' and contains a code editor with the following Cypher query:

```
1 MATCH (director:Person)
2 // WHERE director.name =~ 'L\\..*Wachowski'
3 WHERE (director.name) CONTAINS 'Wachowski'
4 MATCH (director)-[:DIRECTED]-(movie:Movie)
5 RETURN movie.title AS Title, movie.released AS YearOfRelease;
```

Below the code editor is a table with the results:

	Title	YearOfRelease
1	"The Matrix"	1999
2	"The Matrix"	1999
3	"The Matrix Reloaded"	2003
4	"The Matrix Reloaded"	2003
5	"The Matrix Revolutions"	2003
6	"The Matrix Revolutions"	2003