

Exercise Sheet - Cypher in Neo4j

In this exercise session, we will explore a **graph database** in [Neo4j](https://neo4j.com/). This tool allows us to modelize, visualize and query — in the query language [Cypher](https://neo4j.com/docs/cypher-manual/) — interconnected data that is represented according to the **property graph model** (PGM).

PGM building blocks:


- the **nodes** are abstract entities,
- the **relationships/edges** are directed links between nodes,
- the **properties/attributes** are labels relative to a node **or** to a relation.

Cypher building blocks:

- **MATCH** : specification of the search pattern
- **WHERE** : filtering of results
- **RETURN** : formatting of results
- **LIMIT** : specification of the size of the displayed results
- **ORDER BY** : order of results

Manual

Option 1:

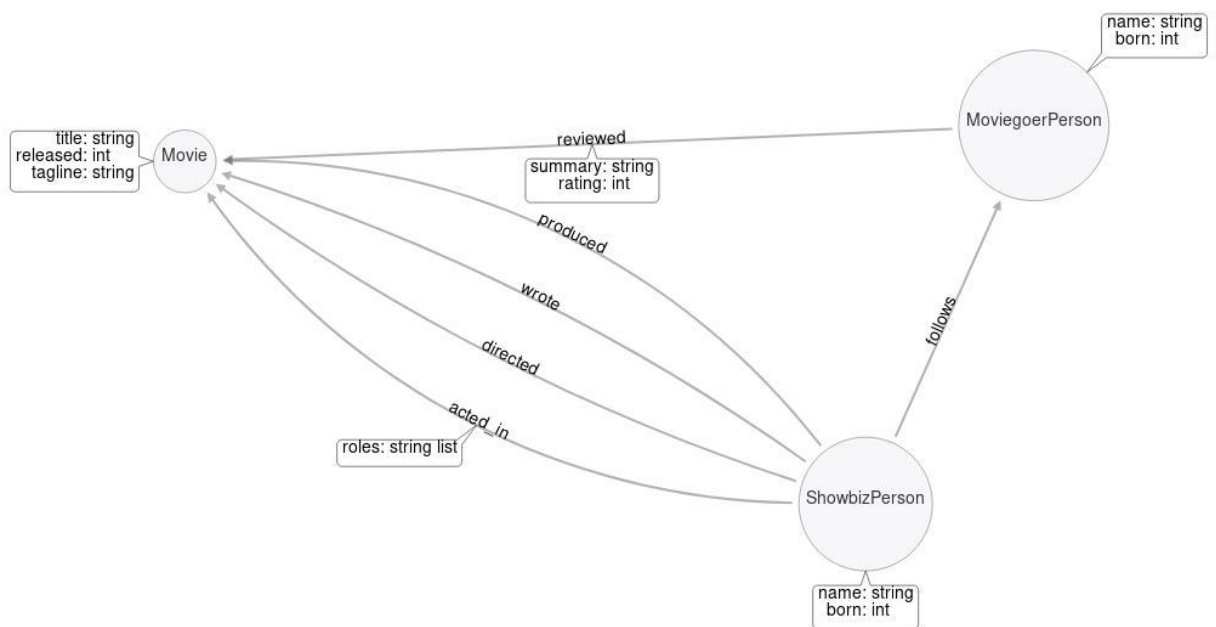
- Download the Neo4j app here: <https://neo4j.com/download/>
- Complete the form and, once the download is finished (≈ 5 min), right-click on the file and in Properties → Permissions, modify the permissions into “Read and write” and enable “Allow executing file as program”.
- Double-click to run the app
- Give a name to your project, click on the tabs Start and Manage; then click on the tab Open Browser
- You will see a prompt in which you will type in Cypher queries
- To run them, click on the third tab on the right of the prompt:  (Play)

Option 2:

- Go to the Neo4j Sandbox: <https://sandbox.neo4j.com/>
- Click on "New Project", then "Blank Sandbox", then on "Open in Browser".

The MOVIES Database

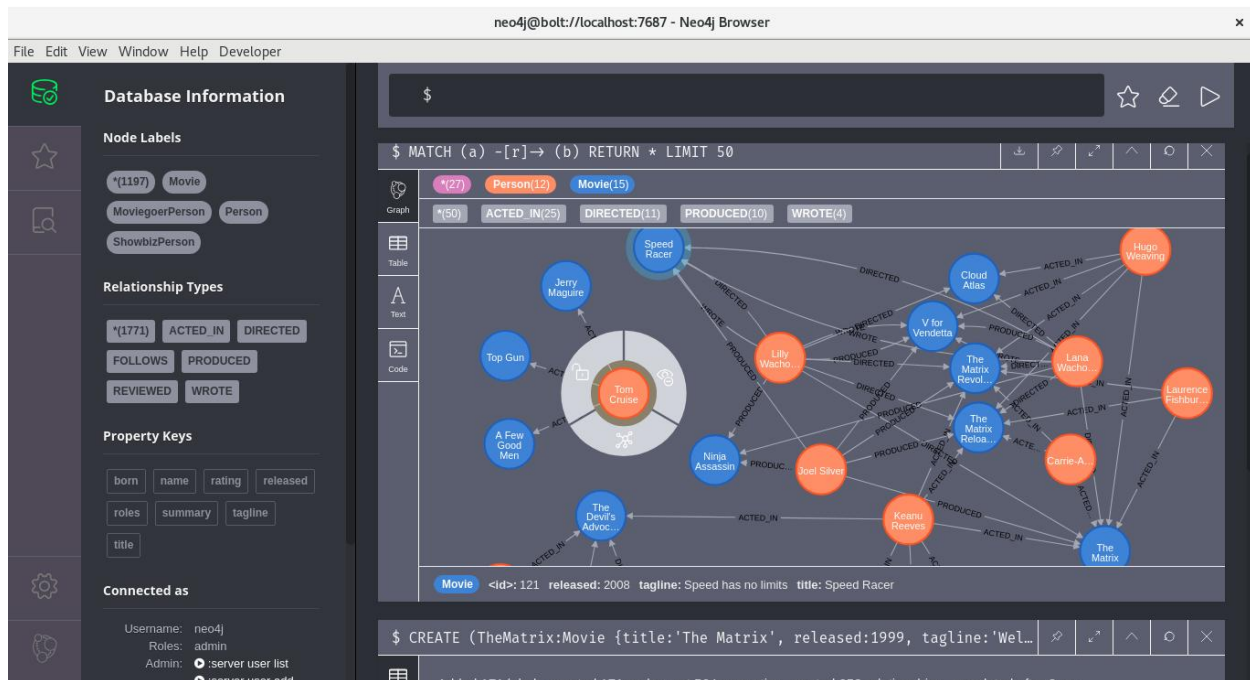
In the file at URL <http://web4.ensiie.fr/~stefania.dumbrava/movies.cql> is a Cypher script that creates a mini database respecting the model illustrated here:



To load the database, copy-paste (Edit → Paste) in the prompt. You will get the message "Added 171 labels, created 171 nodes, set 564 properties, created 253 relationships, completed after 8 ms."

To visualize the graph, type in the prompt the command: `MATCH (a) -[r]-> (b) RETURN * LIMIT 50`.

You will get a graph looking like the graph below. It is available by clicking on the tab "Graph".



To get help on the syntax of Cypher, type in the following instructions in the prompt:

- :help MATCH
- :help WHERE
- :help RETURN

Querying the MOVIE Database

You will write Cypher queries to answer the following questions on the movie database. For example, if you want to select all pairs of nodes (a , b) such that a is of type T_1 and has property $p_1 : v_1$ and b is of type T_2 and a and b are related by a path of label e , you should write:

$$\text{MATCH } (a : T_1 \{p_1 : v_1\}) -[: e] -> (b : T_2) \text{ RETURN } a, b$$

Warning: if you run the same query several times, **the results will be duplicated!** You need to delete all relations with `MATCH (a) -[r]-> (b) DELETE r` et tous les nœuds avec `MATCH (n) DELETE n`. In the following, you must reload the database with movies.cql.

I. Data Description

To analyze the structure of the data in our movie database, we will ask the following queries:

1. Find the set of distinct labels that appear on nodes. Use the function `labels`.
2. Find the set of distinct lists of properties that are associated with nodes. Use the function `keys`. Compare the results with the screenshot above. What do you notice?

- Find all people who only have their name as an associated property. Use a condition in **WHERE**.
- Find all movies that have only their title and release date as associated properties.
- Find the set of distinct types of edges. Use the function **type**.
- Find the set of distinct lists of properties that are associated to edges. Compare the results with the above screenshot. What do you notice?
- Return all subgraphs of which the edges have no associated properties.
- Find all distinct properties that are associated with nodes. This time, use **UNWIND** to transform lists into tuples. Do the same for distinct properties associated with edges.

II. Basic Queries

- Which are the *titles* of movies in which "Keanu Reeves" has played?
- Which are the *titles* of movies that were released after 2000?
- Which are the *names* of the actors who played in a movie with "Meg Ryan"?
Write another query to return the subgraph that relates "Meg Ryan", the actors with whom he played, and the corresponding movies.
- Find all actors that have directed a movie in which they also played. Return the subgraph that contains the nodes of the corresponding actors and movies. Write another query that returns the pairs of the name of the actor and the movie for which they have been actor and director.
- Which are the *names* of all people who wrote reviews on the movie "The Replacements"?
- Which are the *names* of all people who gave a rating >60 to the movie "The Da Vinci Code"?
- Which are the *names* of the 5 movies in which the biggest number of actors have played?
Also display the corresponding number of actors.
- Which are the *names* of the people who wrote but did not produce the same movie? Hint: use **WHERE NOT**.

III. On the fly modifications

- Write a query to add to the movie database the most recent movie by "Tom Hanks": "A Beautiful Day in the Neighborhood".
Check that the movie has indeed been added by selecting it with a query.
- Remove all movies older than 2000. Careful: we need **two** queries: one to delete all relations that could mention those movies (**:ACTED_IN** | **:PRODUCED** | **:DIRECTED** | **:REVIEWED** | **:WROTE**) and another to delete all the corresponding nodes.
- Check with a query that the previous movies have been removed.
- Write a query to change the name of "Meg Ryan" into "Margaret Mary Emily Hyra" in all corresponding nodes dans tous les tous les nœuds. Hint: use the **SET** keyword.

Relational → Graph Databases

Consider the following relational database, named BLOGS:

User (US)		Follower (FR)		Tag (TG)	
<u>uid</u>	uname	<u>fuser</u>	<u>fblog</u>	<u>tuser</u>	<u>tcomment</u>
t ₁	u01	Date	t ₃	u01	b01
t ₂	u02	Hunt	t ₄	u01	b02
			t ₅	u01	b03
			t ₆	u02	b01
Blog (BG)		Comment (CT)			
<u>bid</u>	bname	admin	<u>cid</u>	cblog	cuser
t ₈	b01	Information Systems	u02		
t ₉	b02	Database	u01		
t ₁₀	b03	Computer Science	u02		
<u>cid</u>	cblog	cuser	msg	date	
t ₁₁	c01	b01	u01	Exactly what I was looking for!	25/02/2013

which has for foreign keys:

- FR.fuser ⇒ US.uid
- FR.fblog ⇒ BG.bid
- BG.admin ⇒ US.uid
- CT.cblog ⇒ BG.bid
- CT.cuser ⇒ US.uid
- TG.tuser ⇒ US.uid
- TG.tcomment ⇒ CT.cid.

1. Transform the BLOGS database into a graph database.

To create the corresponding property graph, go to <http://www.apcjones.com/arrows/>

- Click on **+Node** to create new nodes
 - Click and drag from the outer side of a node to create links to other nodes
 - Then, double click on each node and each edge to add labels and a property list
 - Once your graph is done, click on **Export Cypher** and copy-paste into Neo4j's prompt
 - Check you can indeed visualize your new graph in Neo4j.
2. Relate the MOVIES and BLOGS graphs by adding LIKES between users and MoviegoerPerson nodes.
Check the insertion worked by returning the corresponding subgraph.
 3. Remove all relations with MATCH (a) -[r]-> (b) DELETE r and all nodes with MATCH (n) DELETE n.