



# Models and Systems for Big Data

## Graph Database: Neo4j & Cypher

Pr. Nacéra Seghouani

---

This lab assignment aims at putting into practice the main notions underlying graph databases using Neo4j<sup>1</sup> as graph database and Cypher as query language<sup>2</sup>.

The dataset `recommendations.db`, **to be downloaded from Edunao**, describes a set of movies, the artists who acted in and directed these movies, the users who gave a score between 1 and 5 to movies based on whether they dislike or like them.

### 1 NEO4J DESKTOP & DATA IMPORT

First install Neo4j Desktop on your laptop from official site given in footnote. Launch Neo4j Desktop on your computer, an interface will be displayed in a window (see Figure 1), click on the default project, create a database named Graph (remember your login/password if required).

To import data, **copy the content of the repository** `recommendations.db` **to replace the content of the created repository** `graph.db`.

`graph.db` repository is located in `<neo4j-home>/data/databases` or in `<neo4j-home>\data\databases` (if Windows system). This repository depends on the system<sup>3</sup>. For example in MacOS the default repository is:  
user home/Library/Application Support/Neo4j Desktop/Application/  
neo4jDatabases/database-xxxxx/installation-3.5.12/data/databases/graph.db.

Click on Start to connect to your database. Open the Neo4j Browser (see Figure 1).

The 'Overview' frame at the left in Figure 1 summarizes the node labels, the relationship types and the key properties. The frame at the top right is used to write Cypher queries. The frame at the down right displays query results.

---

<sup>1</sup><https://neo4j.com/download/>

<sup>2</sup><https://neo4j.com/docs/cypher-refcard/current/>

<sup>3</sup><https://neo4j.com/docs/operations-manual/current/configuration/file-locations/>

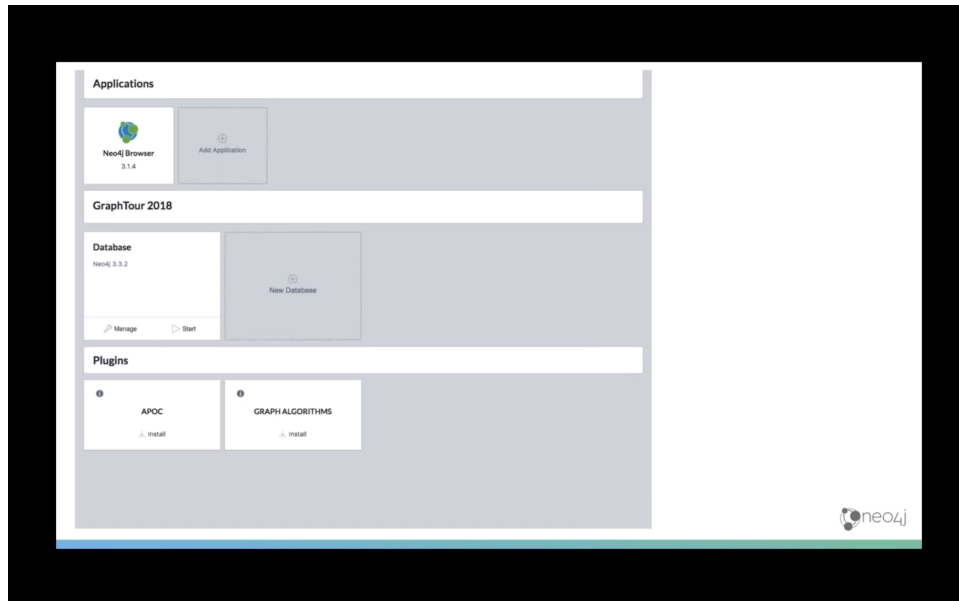


Figure 1: Neo4j Desktop Interface, Graph created and plugins added

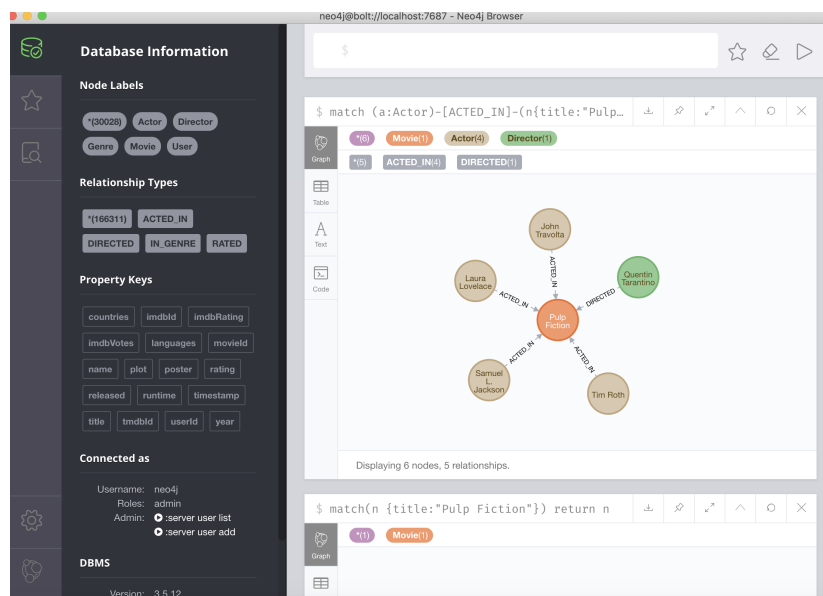


Figure 2: Neo4j Desktop Browser (Client)

Now, you can execute the following queries (**separately**) in the browser, to have the number of nodes, the number of different node labels, the number of relationships and the number of relationship types.

### Some Analysis Queries

```
MATCH (n) RETURN COUNT(n)
COUNT(n)
30028
MATCH (n) RETURN LABELS(n), COUNT(*)
LABELS(n) COUNT(*)
["Movie"] 9125
["Genre"] 20
["User"] 671
["Actor"] 15873
["Director"] 4339
MATCH ()-[r]-() RETURN COUNT(r)
COUNT(r)
332622
MATCH ()-[r]-() RETURN TYPE(r), COUNT(*)
TYPE(r) COUNT(*)
"IN_GENRE" 40680
"RATED" 200008
"ACTED_IN" 71858
"DIRECTED" 20076
```

## 2 QUERYING DATA PRELIMINARIES

To explore the graph, we will write Cypher<sup>4</sup> queries in the top frame starting with prompt symbol \$. What do you observe when you execute the following queries?

### Example 1

```
MATCH (n:Movie {title:"Toy Story"}) RETURN n
or
MATCH (n:Movie) WHERE n.titre="Toy Story" RETURN n
```



- MATCH clause specifies a node pattern: nodes labelled Movie and having the property title with "Toy Story" value.
- RETURN clause specifies the matched nodes stored in n variable to be returned.
- WHERE clause specifies the conditions on titre property value of n.

The result can be visualized as a text rather than a graph, as shown in Figure 3.

### Example 2

```
MATCH (n:Movie {title:"Pulp Fiction"})-[r]-(m)
RETURN n, m;
```



<sup>4</sup><https://neo4j.com/docs/cypher-refcard/current/>



Figure 3: Toy Story Query Results

#### Example 3

```
MATCH (n:Movie {title:"Pulp Fiction"})-[r:IN_GENRE]->(m:Genre)
RETURN n, m;
```

#### Example 4

```
MATCH (n:Movie)-[:ACTED_IN]-(a:Actor)
RETURN n.title, count(a) AS nbActors
ORDER BY nbActors DESC
LIMIT 5;
```

In this query:

- RETURN clause specifies `n.title` followed by `count(a)`. The result is the title of the 5 movies having the most number of actors and the number of actors of each title. In SQL language we would write:

```
SELECT m.title, count(*) AS nbActors
FROM movies_actors ma, movies m WHERE ma.movie_id = m.id
GROUP BY m.title
ORDER BY nbActors DESC
LIMIT 5;
```

In Cypher, GROUP BY clause is implicitly specified in the aggregate function count(a). As this function follows `n.title`, it is applied to each `n.title` value. The query without `n.title` would return the number of links of type `ACTED_IN` relating Actor nodes to Movie nodes.

- AS `nbActors` assigns a name to `count(a)` result.
- ORDER BY `nbActors` DESC LIMIT 5 specifies that the number of actors are sorted in descending order and limited to 5.

### Example 5

```
MATCH (n:Movie)<-[:RATED]-(u:User)
WITH n, count(u) AS nbRates
WHERE nbRates >= 100
RETURN n.title
LIMIT 5;
```

WITH clause specifies a pipe (chaining) between the Match output and the input of the next clause. In this case the input of Where clause is Movie nodes having incoming RATED links from User nodes and the number of links of each Movie node. The query returns the titles of movies that have been rated at least 100 times.

Note that we must assign a name to count(u) result to keep this result as input of the next clause.

Someone might try the following query. The use of an aggregate function in WHERE clause is not allowed. Indeed, the WHERE condition is applied to each matched result whereas the aggregate function needs to be applied to all the results; in this case all the User nodes found (matched).

```
MATCH (n:Movie)<-[:RATED]-(u:User)
WHERE count(u) >= 100
RETURN n.title
LIMIT 5;
```

## 3 FIRST PART TO DO: DATA EXPLORATION QUERIES

Write and execute the following queries:

Exercise 1. The genres of movies in the database

Exercise 2. The number of movies in the database

Exercise 3. The title of movies released in 1995

Exercise 4. The number of directors by movie sorted in decreasing order

Exercise 5. The names of the directors and the title of the movies that they directed and they also played in

Exercise 6. The genres of movies that Tom Hanks played in

Exercise 7. The title and the rate of movies rated by the user number 46 sorted by rate in decreasing order

Exercise 8. At most 5 movies that obtained the highest average ratings and that have been rated by at least 100 rates

## 4 SECOND PART TO DO : RECOMMENDATION SYSTEM QUERIES

In this part, we want to write queries for a movie collaborative-based recommendation system. This consists in recommending a user  $u$  movies s/he hasn't rated yet that are liked by a similar user  $v$ . In this context, we consider that a user liked a movie if s/he rated it at least 3. Figure 4 shows that the user  $u$  liked 6 movies and 3 of them are also liked by the user  $v$ . On the basis of collaborative reasoning, the user  $u$  might like the other movies liked by the user  $v$ .

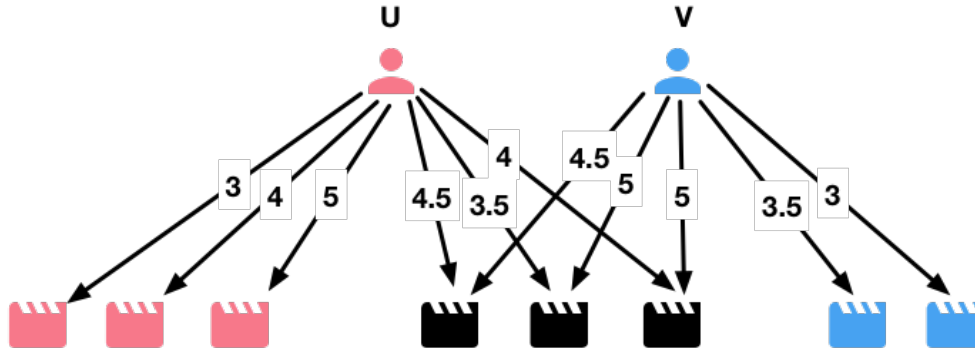


Figure 4: Collaborative Filtering

In order to compute a similarity score between two users, we can use the Jaccard index. Let  $L(u)$  and  $L(v)$  be the sets of movies that  $u$  and  $v$  liked respectively. The similarity score is defined as follows:

$$J(u, v) = \frac{|L(u) \cap L(v)|}{|L(u) \cup L(v)|} \quad (1)$$

The recommendation system will propose to a target user the movies liked by the  $k$  most similar users according to Jaccard index.

**Exercise 1.** To compute  $|L(target)|$ , write and execute the query which returns the target user (userId: "3") and the number of movies s/he liked

**Exercise 2.** To compute  $|L(other)|$ , write and execute the query which returns each other user (except the target user 3) and the number of movies that each one liked

**Exercise 3.** To compute  $|L(target) \cup L(other)|$ , write and execute the query which returns the target user 3, the number of movies that the target liked, the other users and the number of movies that each one liked

☛ Use the WITH clause to chain the previous queries (See Example 2).

**Exercise 4.** To compute  $|L(target) \cap L(other)|$ , write and execute the query which returns each user (except the target) and the number of movies s/he and the target user liked too

**Exercise 5.** Write and execute the query which returns the five users most similar to the target user and their similarity score with the target user using  $J(target, other)$  formula  
1

- ☛ Use the WITH clause to chain the previous queries (Example 2).
- ☛ Multiply first the numerator before dividing, otherwise Cypher computes an integer division.



**Exercise 6.** Write and execute the query to obtain the list of the movies liked by the user the highest similarity (using the identifier value found in the previous query) that the target user hasn't rated yet. Sort this list by decreasing rate

- ☛ As `count()`, `collect()` is an aggregation function which returns a list of values.
- ☛ The expressions `x in list` and `not(x in list)` return a boolean value.