

LAB 4 NEO4J



Professeur : BOUBCHIR, Larbi

Module : ADIF82

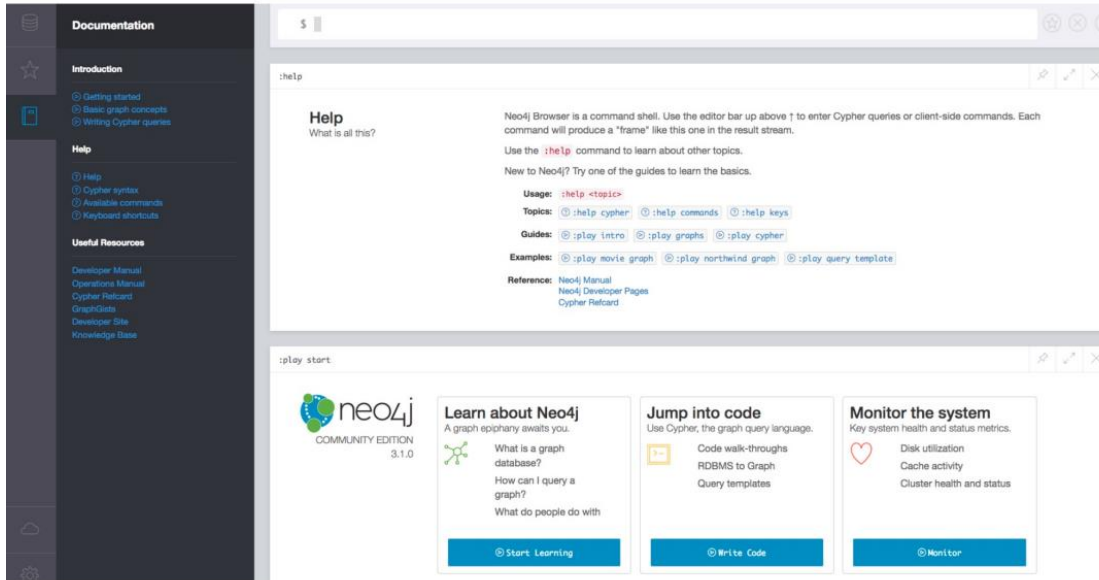
Table des matières

NEO4J Browser	4
Queries And Visualization.....	4
Import dataset.....	5
Neo4J browser – Display Data	5
Cours.....	6
Introduction to Cypher	6
Two Nodes, One Relationship.....	6
Optional Match.....	6
Two Nodes, A Known Relationship	7
Paths	7
Queries On « Movies » Example (1/3).....	7
Queries On « Movies » Example (2/3).....	8
Alias	8
More Queries.....	9
Aggregation Functions.....	9
Example – count(*).....	9
SORT And limit.....	10
Aggregation - collect.....	10
Find all the nodes	10
DISTINCT	11
Index Creation	11
Conditions.....	11
Conditions on the properties.....	11
Conditions with comparison.....	12
Conditions on patterns (1/2)	12
Conditions on patterns (2/2)	12
String Comparison	12
Exercise 1	13
Table	13
Text	13
Explication de la requête	13
Cours 2.....	14

Update With CYPHER.....	14
Creation with MERGE	14
Adding properties.....	14
Modifying properties.....	15
Adding Relationships	15
Modifying a Relationship Property.....	15
Delete a Node.....	16
Deleting nodes and all relationships	16
Exercise 2	17
Table	17
Explication de la requête	17
Recommandation	18
Recommendations : Overview	18
Definition	18
Recommender Systems	19
Two categories.....	19
Recommendation and graphs	19
Exercise 3.....	20
Table	20
Text	20
Explication de la requête	20
Go further	21
Matching many relationships	21
Path with variable length.....	21
Length of the relationship	21

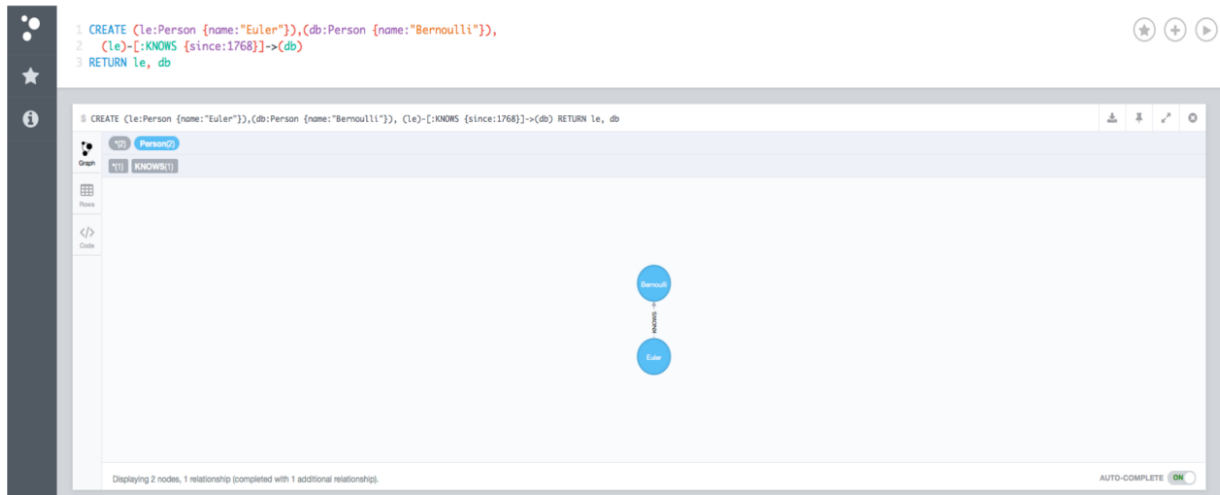
NEO4J Browser

Help



The screenshot shows the Neo4j Browser interface with the 'Help' sidebar on the left. The main content area displays the ':help' command output, which includes an introduction to the Neo4j Browser as a command shell, usage instructions, and links to various topics, guides, examples, and references. Below the help text, there are three interactive cards: 'Learn about Neo4j', 'Jump into code', and 'Monitor the system', each with a 'Start Learning', 'Write Code', or 'Monitor' button respectively.

Queries And Visualization



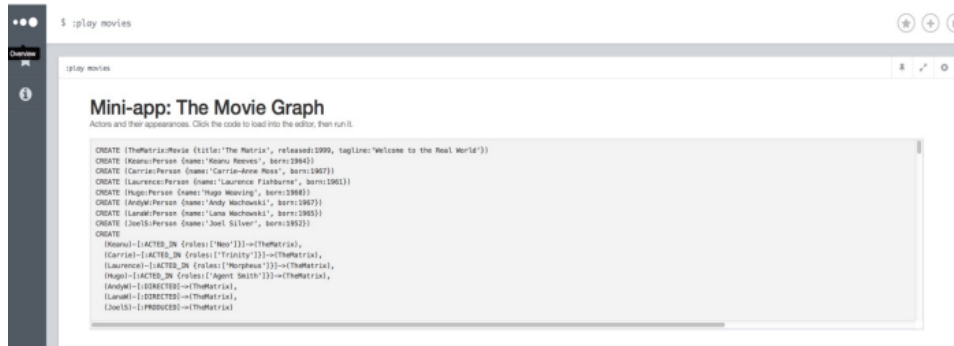
The screenshot shows the Neo4j Browser interface with a Cypher query entered in the editor. The query is:

```
1 CREATE (le:Person {name:"Euler"}),(db:Person {name:"Bernoulli"}),
2 (le)-[:KNOWS {since:1768}]->(db)
3 RETURN le, db
```

The query is executed, and the results are displayed in a table with two columns: 'le' and 'db'. The table shows the two nodes created. Below the table, a graph visualization is shown, displaying two nodes (Euler and Bernoulli) connected by a relationship labeled 'KNOWS'.

Import dataset

:play movies



Neo4J browser – Display Data

- **Example: MATCH (n)-[r]->(n2) RETURN r, n1, n2 LIMIT 25**



MATCH (n)-[r]->(n2) RETURN r, n, n2 LIMIT 25

MATCH est une clause utilisée pour spécifier un motif dans le graphique Neo4J. Le motif ici (n)-[r]->(n2) cherche des relations r où un nœud n est connecté à un autre nœud n2. Le -[r]-> indique une relation dirigée partant de n et allant vers n2.

RETURN spécifie ce qui doit être retourné par la requête. Dans ce cas, r, n1, et n2 devraient être retournés.

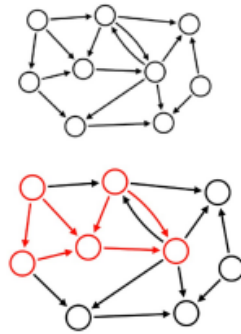
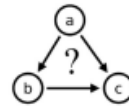
LIMIT 25 est une clause qui limite le nombre de résultats retournés par la requête. Ici, elle est configurée pour retourner seulement les 25 premières relations trouvées qui correspondent au motif.

Cours

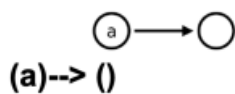
Introduction to Cypher

Cypher - definition

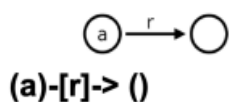
- Neo4j Query language
- Pattern-Matching Declarative Language
- SQL-Like
- Suitable for graphs

Principle

Two Nodes, One Relationship



MATCH (a) --> ()
RETURN a.name



MATCH (a) -[r]-> ()
RETURN a.name, type(r)

Optional Match

- We look for the node a with its relationships if they exist

OPTIONAL MATCH (a) -[r]-> ()
RETURN a.name, type(r)

Two Nodes, A Known Relationship



(a)-[:ACTED_IN]-> (m)

MATCH (a) -[:ACTED_IN]-> (m)

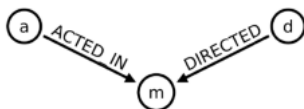
RETURN a.name, m.title

- Returning the properties of the relations

MATCH (a) -[r:ACTED_IN]-> (m)

RETURN a.name, r.roles, m.title

Paths



▪ **MATCH** (a) -[:ACTED_IN]-> (m) <-[:DIRECTED] - (d)

▪ **RETURN** a.name, m.title, d.name

Queries On « Movies » Example (1/3)

- Display the actor « Tom Hanks »

MATCH (tom {name: "Tom Hanks"}) **RETURN** tom

- Display the movie which title is « Cloud Atlas »

MATCH (cloudAtlas {title: "Cloud Atlas"}) **RETURN** cloudAtlas

- Display 10 persons

MATCH (people:Person) **RETURN** people.name **LIMIT** 10

- Display movies released in the '90s

MATCH (nineties:Movie) **WHERE** nineties.released > 1990
AND nineties.released < 2000 **RETURN** nineties.title

- Which actors have played in the same movie as Tom Hanks?

MATCH (tom:Person {name:"Tom Hanks"})-[:ACTED_IN]->(m)<-[:ACTED_IN]-(coActors) **RETURN** coActors.name

Queries on « Movies »

- Display Tom Hanks' movies
- Who directed the film "Cloud Atlas"?
- Which director also played in a movie?

```
MATCH (tom:Person {name: "Tom Hanks"})-[:ACTED_IN]->(tomHanksMovies)
RETURN tom,tomHanksMovies
```

```
MATCH (cloudAtlas {title: "Cloud Atlas"})<-[:DIRECTED]-(directors) RETURN
directors.name
```

```
MATCH (tom:Person {name:"Tom Hanks"})-[:ACTED_IN]->(m)<-[:ACTED_IN]-
(coActors) RETURN coActors.name
```

```
MATCH (a)-[:ACTED_IN]->(m)<-[:DIRECTED]-(d)
RETURN a.name, m.title
```

How people are related to "Cloud Atlas"...

```
MATCH (people:Person)-[:relatedTo]-(:Movie {title: "Cloud Atlas"}) RETURN
people.name, Type(relatedTo), relatedTo
```

Queries On « Movies » Example (2/3)

```
MATCH (a)-[:ACTED_IN]->(m)<-[:DIRECTED]-(d)
RETURN a.name, m.title, d.name
```

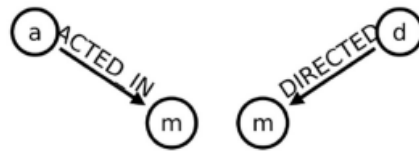
a.name	m.title	d.name
"Keanu Reeves"	"The Matrix"	"Andy Wachowski"
"Keanu Reeves"	"The Matrix Reloaded"	"Andy Wachowski"
"Noah Wyle"	"A Few Good Men"	"Rob Reiner"
"Tom Hanks"	"Cloud Atlas"	"Andy Wachowski"

Alias

```
MATCH (a)-[:ACTED_IN]->(m)<-[:DIRECTED]-(d)
RETURN a.name AS actor , m.title AS movie, d.name
AS director
```

actor	movie	director
"Keanu Reeves"	"The Matrix"	"Andy Wachowski"
"Keanu Reeves"	"The Matrix Reloaded"	"Andy Wachowski"
"Noah Wyle"	"A Few Good Men"	"Rob Reiner"
"Tom Hanks"	"Cloud Atlas"	"Andy Wachowski"

More Queries



1st way

MATCH (a) -[:ACTED_IN]-> (m), (m) <-[:DIRECTED] - (d)

RETURN a.name, m.title, d.name

2nd way:

MATCH (a) -[:ACTED_IN]-> (m), (d) -[:DIRECTED] -> (m)

RETURN a.name, m.title, d.name

Aggregation Functions

- **Count(x)** – The number of occurrences
- **Min(x)** – minimum value
- **Max(x)** – maximum value
- **Avg(x)** – average
- **Sum(x)** – sum
- **Collect(x)** – Aggregates data in a table

Example – count(*)

MATCH (a) -[:ACTED_IN]-> (m) <-[:DIRECTED] - (d)

RETURN a.name, d.name, count(*)

a.name	d.name	count(*)
"Aaron Sorkin"	"Rob Reiner"	2
"Keanu Reeves"	"Andy Wachowski"	3
"Hugo Weaving"	"Tom Tykwer"	1

MATCH (a) -[:ACTED_IN]-> (m) <-[:DIRECTED] - (d)

RETURN a.name **AS** actor, d.name **AS** director , count(m)
AS count

SORT And limit

```
MATCH (a) -[:ACTED_IN]-> (m) <-[:DIRECTED] - (d)
RETURN a.name AS actor, d.name AS director ,
count(m) AS count
ORDER BY count DESC
LIMIT 5
```

Aggregation- collect

Aggregation - collect

```
MATCH (a) -[:ACTED_IN]-> (m) <-[:DIRECTED] - (d)
RETURN a.name AS actor, d.name AS director ,
collect (m.title) AS list
```

Find all the nodes

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

Directors who directed movies with Tom Hanks as actor

```
MATCH (tom:Person) - [:ACTED_IN] ->
(movie:Movie), (director:Person) - [:DIRECTED] ->
(movie:Movie)
WHERE tom.name="Tom Hanks"
RETURN director.name
```

director.name
Mike Nichols
Robert Zemeckis
Penny Marshall
Robert Zemeckis
Ron Howard
Frank Darabont
Ron Howard

DISTINCT

```
MATCH (tom:Person) – [:ACTED_IN] ->
(movie:Movie), (director:Person) – [:DIRECTED] ->
(movie:Movie)
WHERE tom.name="Tom Hanks"
RETURN DISTINCT director.name
```

Index Creation

- The 'Person' nodes, indexed by their 'name'
CREATE INDEX ON :Person(name)
- The nodes 'Movie', indexed by their 'title'
CREATE INDEX ON :Movie(title)

Conditions

- Find movies where Tom Hanks and Kevin Bacon played
- **MATCH** (tom:Person) –[:ACTED_IN] -> (movie),
(kevin:Person)-[:ACTED_IN]->(movie)
- **WHERE** tom.name="Tom Hanks " **AND**
kevin.name= "Kevin Bacon"
- **RETURN** movie.title

Conditions on the properties

- The films where Keanu Reeves played the role of "Neo »

```
MATCH (actor:Person) –[r:ACTED_IN] -> (movie)
WHERE actor.name= "Keanu Reeves " AND "Neo "
IN (r.roles)
RETURN movie.title
```

- **2nd solution:**
MATCH (actor:Person) –[r:ACTED_IN] -> (movie)
WHERE actor.name= "Keanu Reeves " **AND ANY**(x
IN r.roles **WHERE** x="Neo")
RETURN movie.title

Conditions with comparison

- Find actors who have played with Tom Hanks and who are older than him

```
MATCH (tom:Person) -[:ACTED_IN] -> (movie),
(a:Person)-[:ACTED_IN]->(movie)
WHERE tom.name= "Tom Hanks "
AND a.born < tom.born
RETURN DISTINCT a.name, (tom.born-a.born) AS
diff
```

Conditions on patterns (1/2)

- Actors who have worked with Gene Hackman and who have previously directed films (are also directors)

```
MATCH (gene:Person)-[:ACTED_IN]->(movie),
(n)-[:ACTED_IN]->(movie)
WHERE gene.name="Gene Hackman"
AND (n)-[:DIRECTED]->()
RETURN DISTINCT n.name
```

Conditions on patterns (2/2)

- Actors who worked with " Keanu Reeves ", but not when he played with " Hugo Weaving "

```
MATCH (keanu:Person)-[:ACTED_IN]->(movie),
(n)-[:ACTED_IN]->(movie),
(hugo:Person)
WHERE keanu.name=« Keanu Reeves » AND
hugo.name=« Hugo Weaving »
AND NOT (hugo)-[:ACTED_IN]->(movie)
RETURN DISTINCT n.name
```

String Comparison

```
MATCH (a) -[:ACTED_IN]-> (matrix:Movie)
WHERE matrix.title='The Matrix' AND a.name
CONTAINS 'Emil'
RETURN a.name
```

- =~ "regexp »
 CONTAINS
 STARTS WITH
 ENDS WITH

Exercise 1

- **Display 5 directors who have directed the largest number of films**

Table

```

1 MATCH (d:Person)-[:DIRECTED]->(m:Movie)
2 RETURN d.name, count(m) as films_directed
3 ORDER BY films_directed DESC
4 LIMIT 5
5

```

	d.name	films_directed
1	"Lilly Wachowski"	25
2	"Lana Wachowski"	25
3	"Ron Howard"	15
4	"Rob Reiner"	15
5	"Nora Ephron"	10

Started streaming 5 records after 2 ms and completed after 3 ms.

Text

d.name	films_directed
"Lilly Wachowski"	25
"Lana Wachowski"	25
"Ron Howard"	15
"Rob Reiner"	15
"Nora Ephron"	10

Explication de la requête

MATCH (d:Person)-[:DIRECTED]->(m:Movie)

RETURN d.name, count(m) as films_directed

ORDER BY films_directed DESC

LIMIT 5

Recherche les nœuds avec le label Person qui ont une relation DIRECTED vers un nœud avec le label Movie. Cela correspond aux réalisateurs et aux films qu'ils ont dirigés.

Retourne le nom du réalisateur et le nombre de films qu'ils ont dirigés, avec une variable films_directed.

Ordonne les résultats par films_directed en ordre décroissant.

Limite le nombre de résultats à 5 pour ne retourner que les cinq premiers réalisateurs.

Cours 2

Update With CYPHER

Node creation

```
CREATE (p:Person {name: 'Me'})
```

```
MATCH (p:Person)  
WHERE p.name='Me'  
RETURN p
```

▪Example with 2 properties:

```
CREATE (m:Movie {title: 'Mystic River', released:  
1993})
```

Creation with MERGE

```
MERGE (p:Person {name: 'Me'})  
RETURN p
```

•Guarantees unique creation

With some options:

```
MERGE (p:Person {name: 'Me'})  
ON CREATE SET p.created=timestamp()  
ON MATCH SET p.accessed= coalesce(p.accessed,0)+1  
RETURN p
```

ON CREATE SET – Executed when creating

ON MATCH SET – Executed when Matching

Adding properties

```
MATCH (p:Person)  
WHERE p.name='Me'  
//Add property  
SET p.born='1980'  
RETURN p
```

Modifying properties

```
MATCH (p:Person)
WHERE p.name='Me'
//ajout de la propriété
SET p.born='1985'
RETURN p
```

Adding Relationships

```
MATCH (movie:Movie),(kevin:Person)
WHERE movie.title='Mystic River' AND kevin.name='Kevin
Bacon'
//creation of relationship
MERGE (kevin) -[:ACTED_IN {roles:['Sean']}] -> (movie)
```

```
MATCH (kevin)-[:ACTED_IN] -> (movie)
WHERE kevin.name ='Kevin Bacon'
RETURN movie.title
```

Modifying a Relationship Property

- Change the role of Kevin Bacon in the movie Mystic River from "Sean" to "Sean Devine »

```
MATCH (kevin:Person)-[r:ACTED_IN] ->
(movie:Movie)
WHERE kevin.name ='Kevin Bacon' and
movie.title='Mystic River'
SET r.roles=['Sean Devine']
RETURN r.roles
```

Delete a Node

```
MATCH (emil:Person)
WHERE emil.name = 'Emil Eifrem'
DELETE emil
```

- The relationships still exist

```
MATCH (emil:Person) -[r]-()
WHERE emil.name = 'Emil Eifrem'
DELETE r
```

Deleting nodes and all relationships

```
OPTIONAL MATCH (emil) -[r]-()
where emil.name = "Emil Eifrem"
DELETE emil, r
```

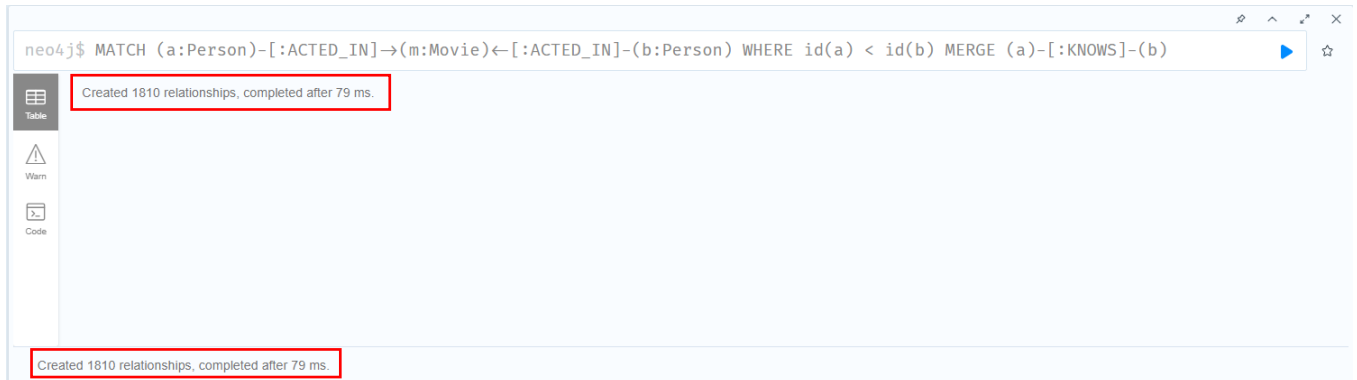
- Deleting all content from the database

```
MATCH (n)
OPTIONAL MATCH (n) -[r]-()
DELETE n, r
```


Exercise 2

- **Add the KNOWS relationship between all the actors in the same movie**

Table



The screenshot shows a Neo4j Cypher query interface. The query is: `neo4j$ MATCH (a:Person)-[:ACTED_IN]->(m:Movie)->[:ACTED_IN]-(b:Person) WHERE id(a) < id(b) MERGE (a)-[:KNOWS]-(b)`. The interface displays a status message: "Created 1810 relationships, completed after 79 ms." in a red-bordered box. The left sidebar contains icons for Table, Warn, and Code.

Explication de la requête

MATCH (a:Person)-[:ACTED_IN]->(m:Movie)->[:ACTED_IN]-(b:Person)

WHERE id(a) < id(b)

MERGE (a)-[:KNOWS]-(b)

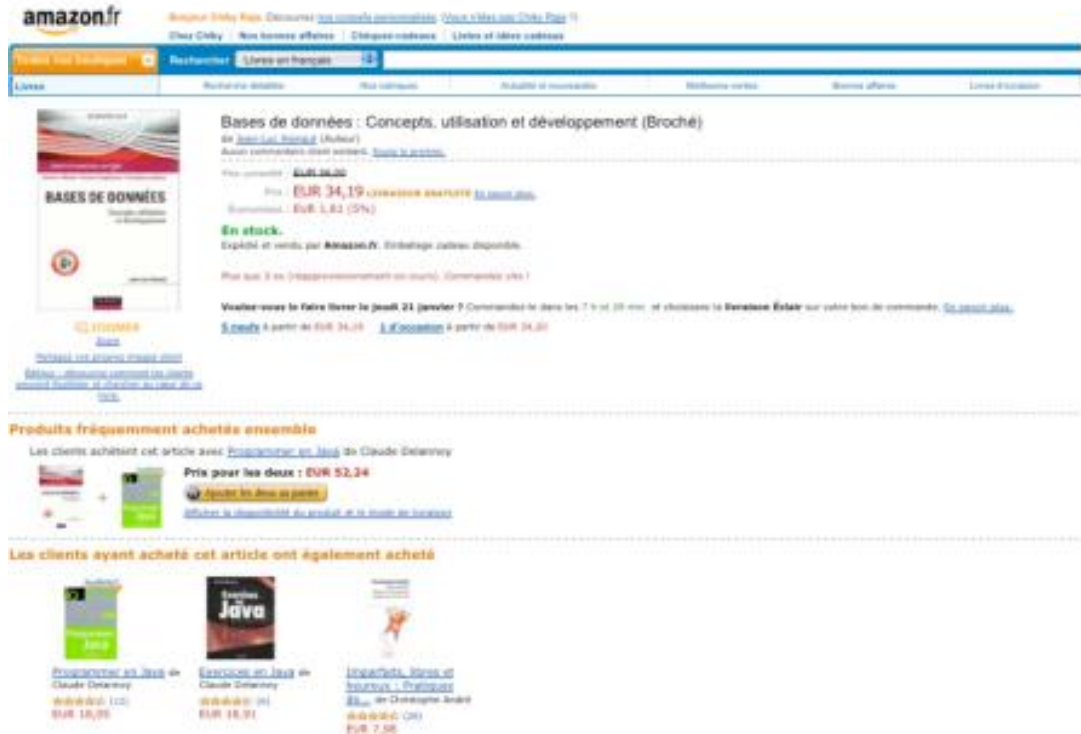
MATCH (a:Person)-[:ACTED_IN]->(m:Movie)->[:ACTED_IN]-(b:Person) trouve toutes les paires d'acteurs (a) et (b) qui ont joué dans le même film (m).

WHERE id(a) < id(b) s'assure que chaque paire est traitée une seule fois et évite de créer des doublons de relations (car sans cette condition, vous créeriez deux relations KNOWS entre les mêmes acteurs, une dans chaque direction).

MERGE (a)-[:KNOWS]-(b) crée la relation KNOWS entre les acteurs s'il n'en existe pas déjà une.

Recommandation

Recommendations : Overview



amazon.fr

Bases de données : Concepts, utilisation et développement (Broché)
 de [Jean-Paul Hwang](#) (Auteur)
 Auteurs recommandés: [David Schreier](#), [Sergey Brin](#)

Plus communément : **EUR 34,19** **COUPON 55% OFFRE** **Le plus cher**
 Economisez : **EUR 18,82 (54%)**

En stock.
 Expédié et vendu par Amazon.fr. Emballage cadeau disponible.

Plus que 3 ex (disponibilités limitées). Commandez vite !

Vous pouvez le faire livrer le **jeudi 21 janvier** ? Commandez-le dans les 7 h et 38 min, et choisissez la livraison **Express** sur votre bon de commande. [En savoir plus](#).

5 euros à partir de **EUR 34,19** **à l'occasion** à partir de **EUR 34,19**

Produits fréquemment achetés ensemble
 Les clients achetant cet article avec [Programmation en Java](#) de Claude Delannoy

Prix pour les deux : EUR 52,34
[Ajouter les deux au panier](#)
 Affichez la disponibilité du produit et le mode de livraison

Les clients ayant acheté cet article ont également acheté

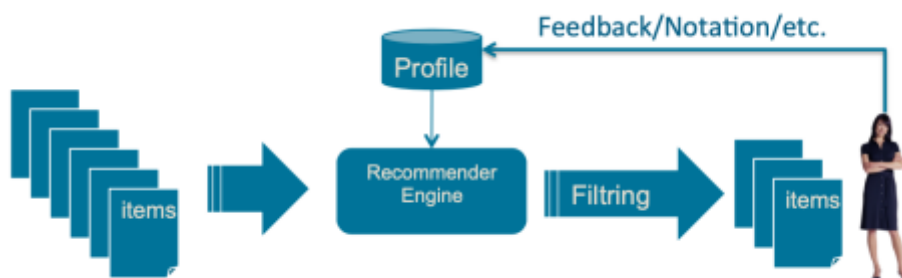
[Programmation en Java](#) de Claude Delannoy
 Disponible : (12)
 EUR 18,95

[Introduction à Java](#) de Claude Delannoy
 Disponible : (4)
 EUR 18,91

[Introduction à Java et Android](#) de Christophe André
 Disponible : (3)
 EUR 7,50

Definition

- **Recommend= "strongly advise something to someone"**
Recommender system system: a variety of processes aimed at providing information to people in line with their interests.



Recommender Systems



Two categories

- **Content-based systems**
 - It is based on the content of the elements visited and look for similarities. The content (documents, articles, etc.) are composed of feature vectors and the similarity calculations are done according to these vectors
- **Collaborative filtering systems**
 - Predict the preferences of articles / objects of users taking into account opinions (notes, votes, etc.) made by "similar" users

Recommendation and graphs

- **Items / users and their characteristics can be represented by nodes**
- **The relations between the users, the items, the users-items can be represented naturally by the relationships in a graph**
- **The recommendation logic can be implemented in a graph DB**

Exercice 3

- **Recommend 3 actors with whom Keanu Reeves could work but this has never been the case**

Table

```

1 MATCH (keanu:Person {name: 'Keanu Reeves'})-[:ACTED_IN]->(m:Movie)
2 WITH keanu, collect(m) AS keanuMovies
3 MATCH (actor:Person)-[:ACTED_IN]->(m2:Movie)
4 WHERE NOT m2 IN keanuMovies AND actor <> keanu
5 WITH actor, COUNT(m2) AS moviesCount
6 RETURN actor.name AS Recommended_Actor
7 ORDER BY moviesCount DESC
8 LIMIT 3
9

```

	Recommended_Actor
1	"Tom Hanks"
2	"Tom Hanks"
3	"Tom Hanks"

Started streaming 3 records after 1 ms and completed after 6 ms.

Text

Recommended_Actor
"Tom Hanks"
"Tom Hanks"
"Tom Hanks"

Explication de la requête

MATCH (keanu:Person {name: 'Keanu Reeves'})-[:ACTED_IN]->(m:Movie)

WITH keanu, collect(m) AS keanuMovies

MATCH (actor:Person)-[:ACTED_IN]->(m2:Movie)

WHERE NOT m2 IN keanuMovies AND actor <> keanu

WITH actor, COUNT(m2) AS moviesCount

RETURN actor.name AS Recommended_Actor

ORDER BY moviesCount DESC

LIMIT 3

D'abord, elle trouve tous les films dans lesquels Keanu Reeves a joué et les collecte dans une liste.

Ensuite, elle trouve tous les autres acteurs qui ont joué dans des films, en excluant les films dans lesquels Keanu Reeves a joué.

Puis, elle regroupe les résultats par acteur et compte le nombre de films pour chaque acteur.

Elle retourne les noms des acteurs qui ont joué dans le plus grand nombre de films distincts de ceux de Keanu Reeves.

Enfin, elle limite les résultats aux trois premiers acteurs.

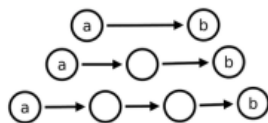
Go futher

Matching many relationships

```
MATCH (a)-[:ACTED_IN|:DIRECTED]->()-<[:ACTED_IN|:DIRECTED]-(b)
MERGE (a)-[:KNOWS]-(b);
```

(Creation of the KNOWS relationship between the actors and directors who worked together)

Path with variable length



(a)-[*n]->(b)

▪ Friends of friends:

```
MATCH (keanu:Person)-[:KNOWS*]->(fof)
WHERE keanu.name="Keanu Reeves" AND NOT
(keanu)-[:KNOWS]-(fof)
RETURN DISTINCT fof.name;
```

Length of the relationship

```
MATCH p=shortestpath((keanu:Person)-[:KNOWS*]->(demi:Person))
WHERE keanu.name="Keanu Reeves" AND NOT
demi.name="Demi moore"
RETURN length(rels(p));
```