

TP - Neo4J

NoSQL databases

Larbi Boubchir



Neo4J browser

Help

The screenshot shows the Neo4j browser interface. On the left, there is a dark sidebar with icons for database, star, and file operations. Below these are sections for Documentation, Introduction, Help, and Useful Resources. The Help section contains links for Help, Cypher syntax, Available commands, Keyboard shortcuts, and developer manuals. The main content area has two tabs: one for the Help command (:help) and another for the play start command (:play start). The :help tab displays a help page for the Neo4j Browser command shell, providing usage instructions, topics, guides, examples, and reference links. The :play start tab shows the Neo4j logo and a summary of the system's status and monitoring features.

Documentation

Introduction

Getting started
Basic graph concepts
Writing Cypher queries

Help

Help
What is all this?

Usage: `:help <topic>`

Topics: [:help cypher](#) [:help commands](#) [:help keys](#)

Guides: [:play intro](#) [:play graphs](#) [:play cypher](#)

Examples: [:play movie graph](#) [:play northwind graph](#) [:play query template](#)

Reference: [Neo4j Manual](#)
[Neo4j Developer Pages](#)
[Cypher Refcard](#)

:play start

neo4j
COMMUNITY EDITION
3.1.0

Learn about Neo4j
A graph epiphany awaits you.

What is a graph database?
How can I query a graph?
What do people do with

[Start Learning](#)

Jump into code
Use Cypher, the graph query language.

Code walk-throughs
RDBMS to Graph
Query templates

[Write Code](#)

Monitor the system
Key system health and status metrics.

Disk utilization
Cache activity
Cluster health and status

[Monitor](#)

Queries and visualization

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



\$ CREATE (le:Person {name:"Euler"}),(db:Person {name:"Bernoulli"}), (le)-[:KNOWS {since:1768}]->(db) RETURN le, db

Graph Person(2)
Rows KNOWS(1)

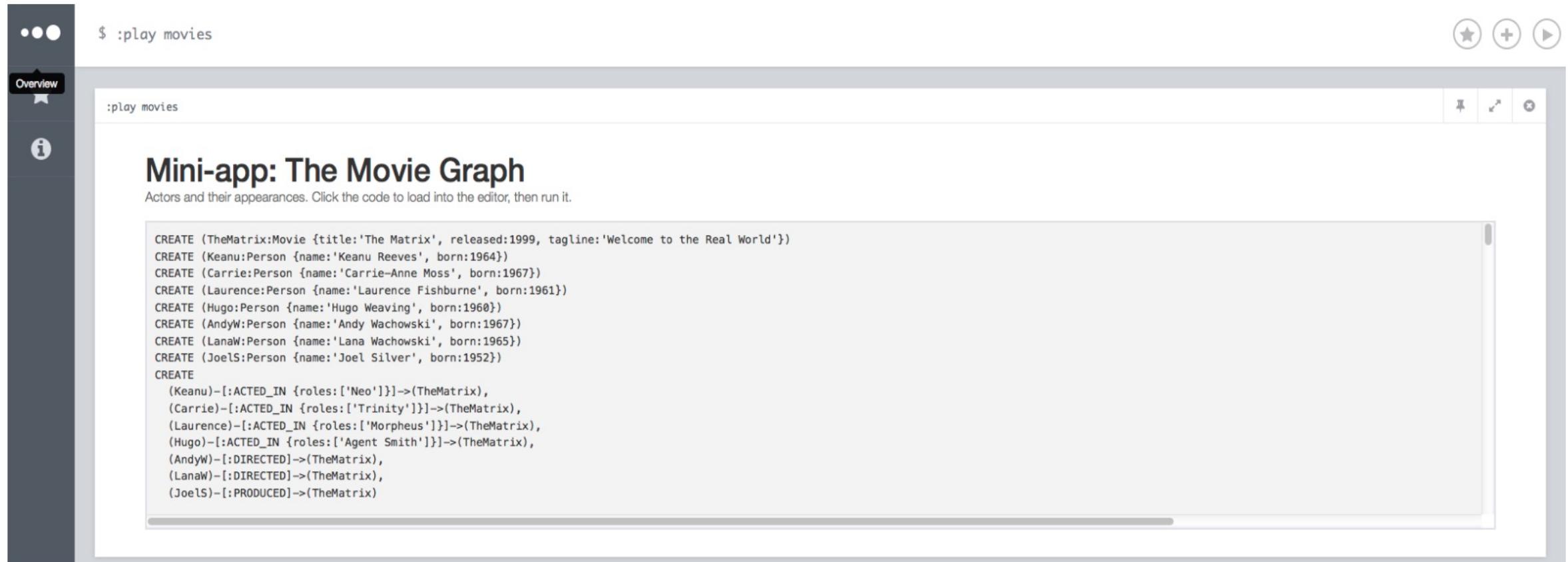
Displaying 2 nodes, 1 relationship (completed with 1 additional relationship).

A graph visualization showing two nodes: "Euler" and "Bernoulli". They are represented by blue circles. A vertical edge connects them, labeled "KNOWS" with an arrow pointing from Euler to Bernoulli. The interface includes tabs for Graph, Rows, and Code, and various export and search icons at the top right.

AUTO-COMPLETE

Import dataset

:play movies



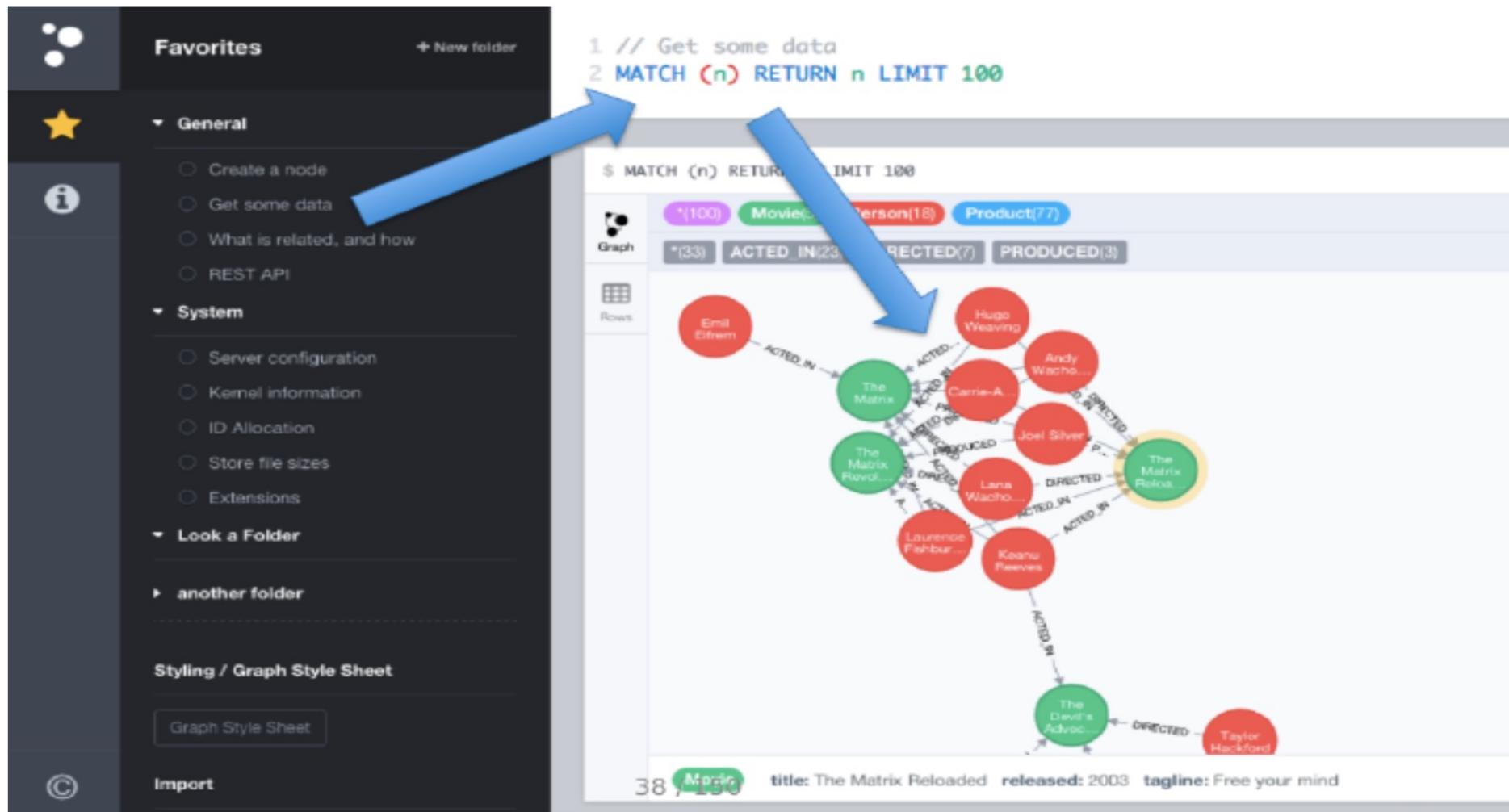
The screenshot shows a Neo4j browser window with the following details:

- Toolbar:** Includes a three-dot menu, a star icon, a plus sign icon, and a right-pointing arrow icon.
- Header:** Shows the command `$:play movies`.
- Overview Tab:** Active tab, showing the query `:play movies`.
- Information Icon:** Shows an 'i' icon.
- Title:** **Mini-app: The Movie Graph**
- Description:** Actors and their appearances. Click the code to load into the editor, then run it.
- Code Block:** Contains the following Cypher query:

```
CREATE (TheMatrix:Movie {title:'The Matrix', 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 (AndyW:Person {name:'Andy Wachowski', born:1967})
CREATE (LanaW:Person {name:'Lana Wachowski', born:1965})
CREATE (JoelS:Person {name:'Joel Silver', born:1952})
CREATE
  (Keanu)-[:ACTED_IN {roles:['Neo']}]->(TheMatrix),
  (Carrie)-[:ACTED_IN {roles:['Trinity']}]->(TheMatrix),
  (Laurence)-[:ACTED_IN {roles:['Morpheus']}]->(TheMatrix),
  (Hugo)-[:ACTED_IN {roles:['Agent Smith']}]->(TheMatrix),
  (AndyW)-[:DIRECTED]->(TheMatrix),
  (LanaW)-[:DIRECTED]->(TheMatrix),
  (JoelS)-[:PRODUCED]->(TheMatrix)
```

Neo4j browser – Display data

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

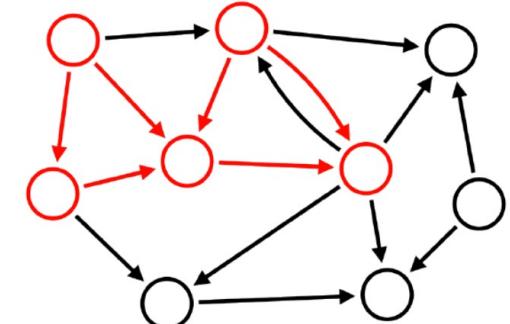
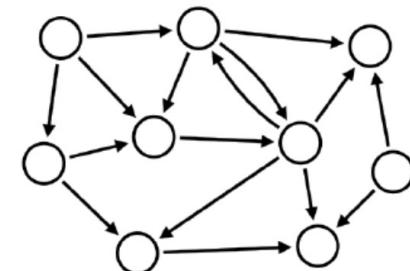
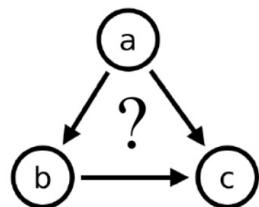


Introduction to Cypher

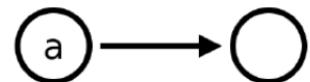
Cypher - definition

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

Principle



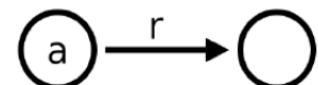
Two nodes, one relationship



(a)--> ()

MATCH (a) --> ()

RETURN a.name



(a)-[r]-> ()

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)

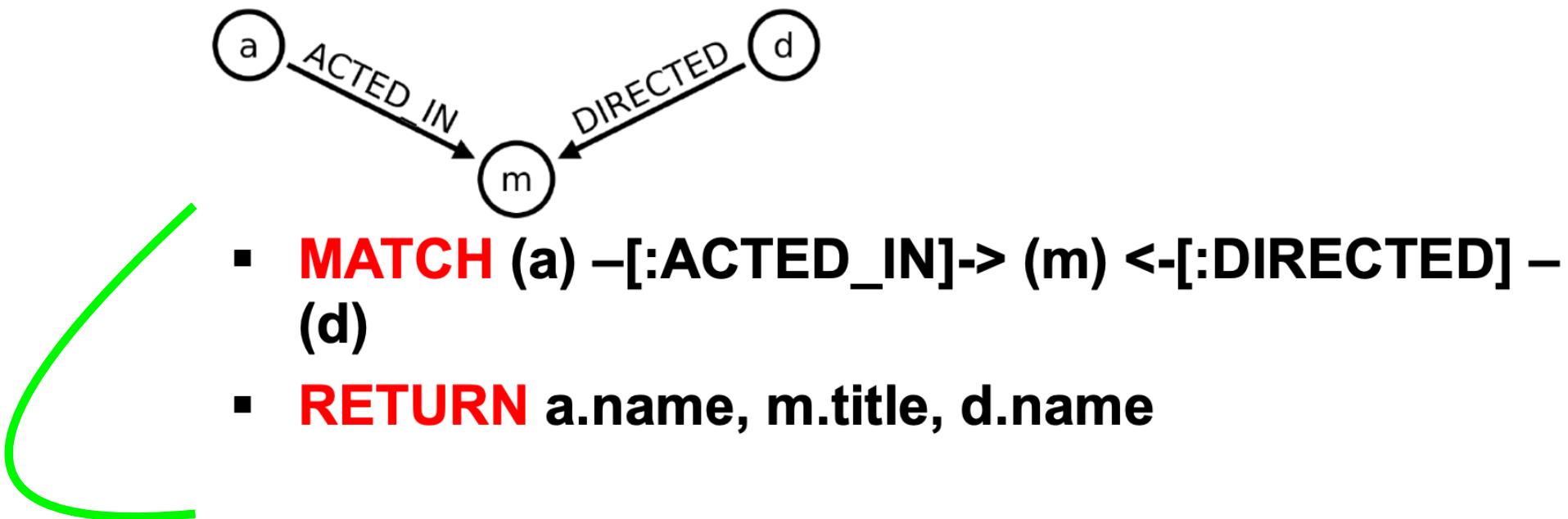
RETURN a.name, m.title

- Returning the properties of the relations

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

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

Paths



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
```

m represents movie

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

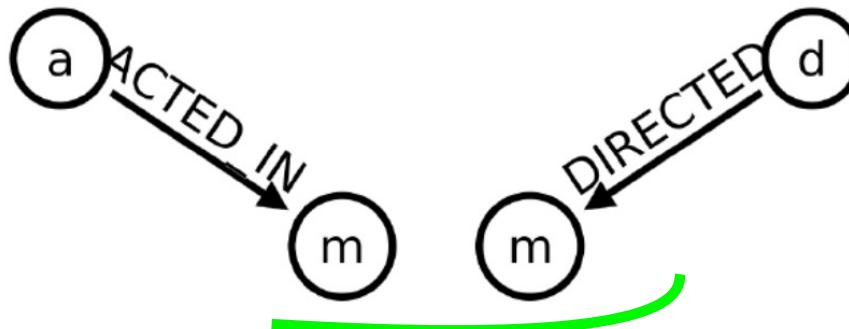
| a.name         | m.title               | d.name           |
|----------------|-----------------------|------------------|
| “Keanu Reeves” | “The Matrix”          | “Andy Wachowski” |
| “Keanu Reeves” | “The Matrix Reloaded” | “Andy Wachowski” |
| “Noah Wyle”    | “A Few Good Men”      | “Rob Reiner”     |
| “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

**2<sup>nd</sup> 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

```
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 Rives 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

- **2<sup>nd</sup> 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) -[r: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 Rives ", 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

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

# 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

## Delete a Node

```
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
```

## **NOT TO DO!!!**

- **Deleting all content from the database**

**MATCH (n)**

**OPTIONAL MATCH (n) –[r]-()**

**DELETE n, r**

## Exercise

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

# Recommendation

## Recommendations: Overview

Bonjour Chiky Raja. Découvrez nos conseils personnalisés. (Vous n'êtes pas Chiky Raja ?)

Chez Chiky | Nos bonnes affaires | Chèques-cadeaux | Listes et idées cadeaux

Toutes nos boutiques Rechercher Livres en français

Livres Recherche détaillée Nos rubriques Actualité et nouveautés Meilleures ventes Bonnes affaires Livres d'occasion

**Bases de données : Concepts, utilisation et développement (Broché)**  
de Jean-Luc Hainaut (Auteur)  
Aucun commentaire client existant. Soyez le premier.  
Prix conseillé : EUR 36,00  
Prix : EUR 34,19 LIVRAISON GRATUITE En savoir plus.  
Économisez : EUR 1,81 (5%)  
**En stock.**  
Expédié et vendu par Amazon.fr. Emballage cadeau disponible.  
Plus que 3 ex (réapprovisionnement en cours). Commandez vite !  
Voulez-vous le faire livrer le jeudi 21 janvier ? Commandez-le dans les 7 h et 29 min et choisissez la livraison Éclair sur votre bon de commande. En savoir plus.  
5 neufs à partir de EUR 34,19 1 d'occasion à partir de EUR 34,20

ZOOMER Zoom

Partagez vos propres images client  
Éditeur : découvrez comment les clients peuvent feuilleter et chercher au cœur de ce livre.

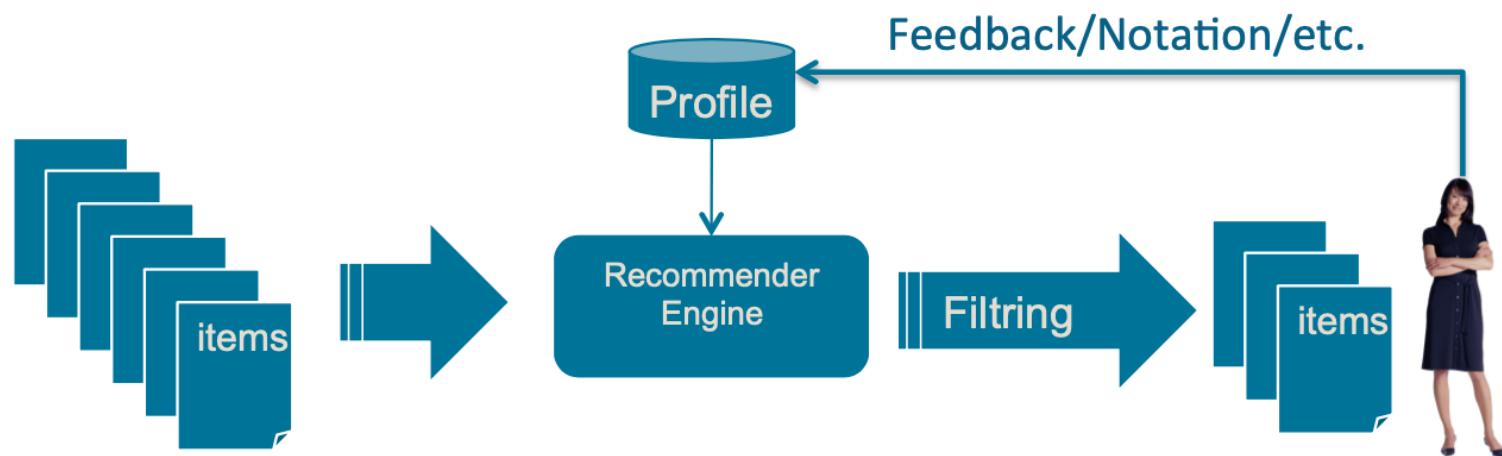
**Produits fréquemment achetés ensemble**  
Les clients achètent cet article avec [Programmer en Java](#) de Claude Delannoy  
Prix pour les deux : EUR 52,24  
Ajouter les deux au panier Afficher la disponibilité du produit et le mode de livraison

**Les clients ayant acheté cet article ont également acheté**

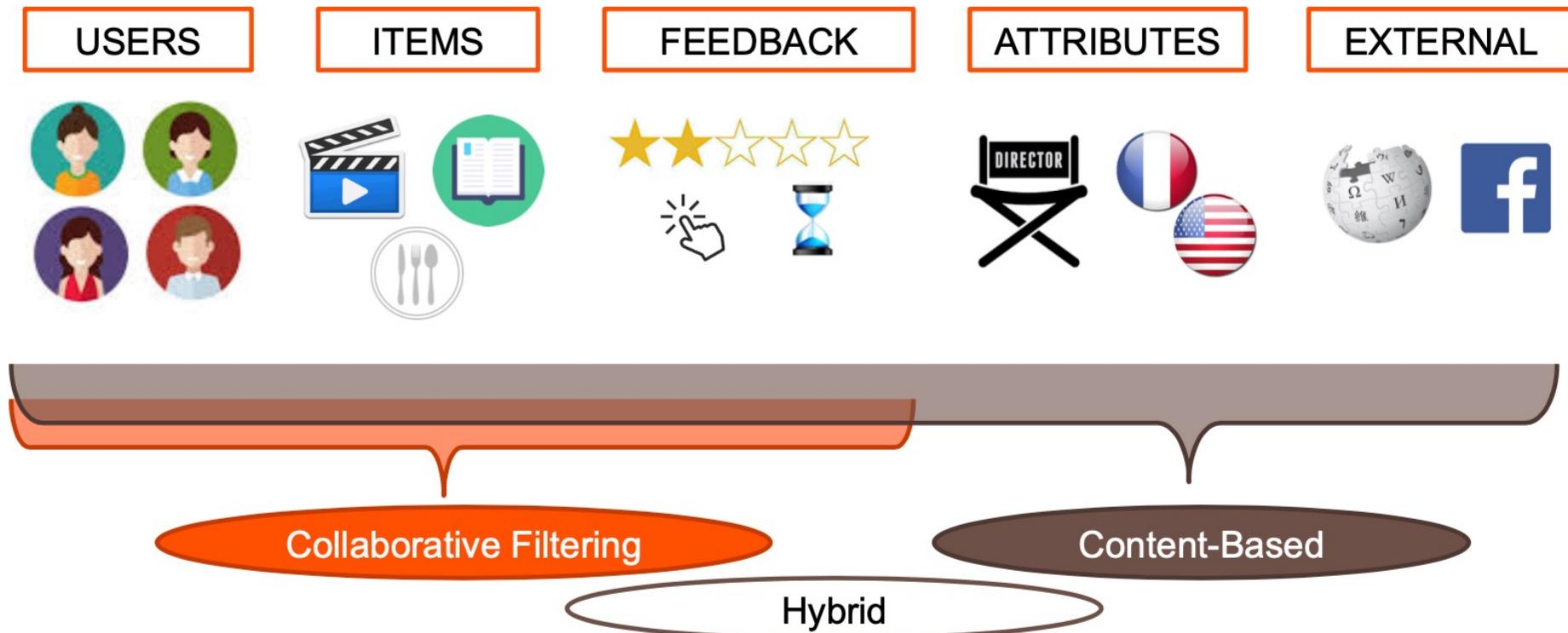
|                                                               |                                                             |                                                                                            |
|---------------------------------------------------------------|-------------------------------------------------------------|--------------------------------------------------------------------------------------------|
|                                                               |                                                             |                                                                                            |
| Programmer en Java de Claude Delannoy<br>★★★★★ (12) EUR 18,05 | Exercices en Java de Claude Delannoy<br>★★★★★ (6) EUR 18,91 | Imparfaits, libres et heureux : Pratiques de... de Christophe André<br>★★★★★ (26) EUR 7,98 |

## 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



LinkedIn™

amazon

NETFLIX

MENDELEY

## 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**

## Exercise

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

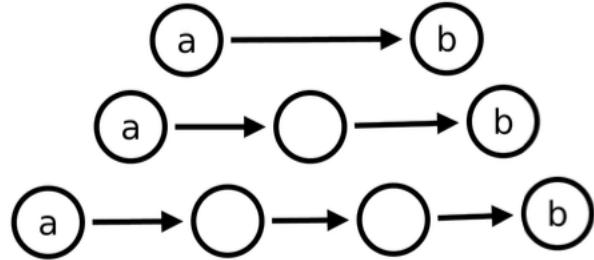
# Go further...

## 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));
```