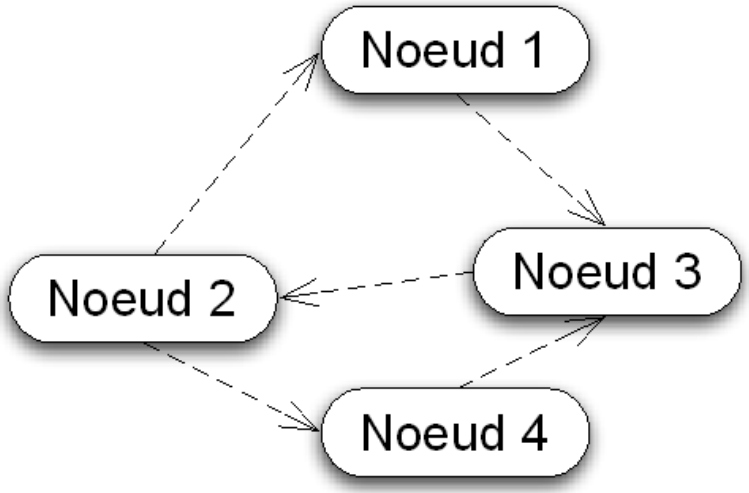
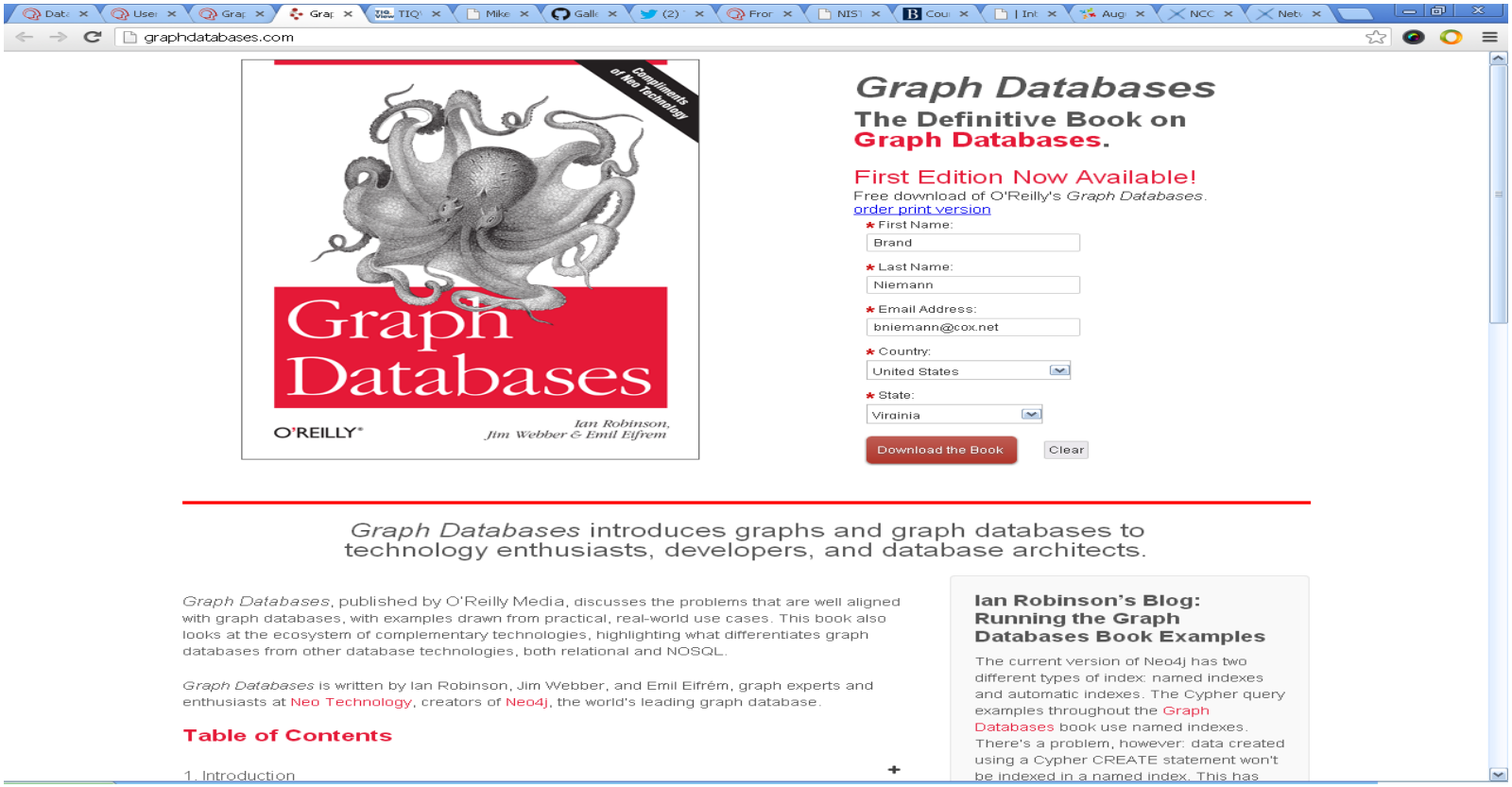
Graph Database with Neo4j

Graph Oriented Databases



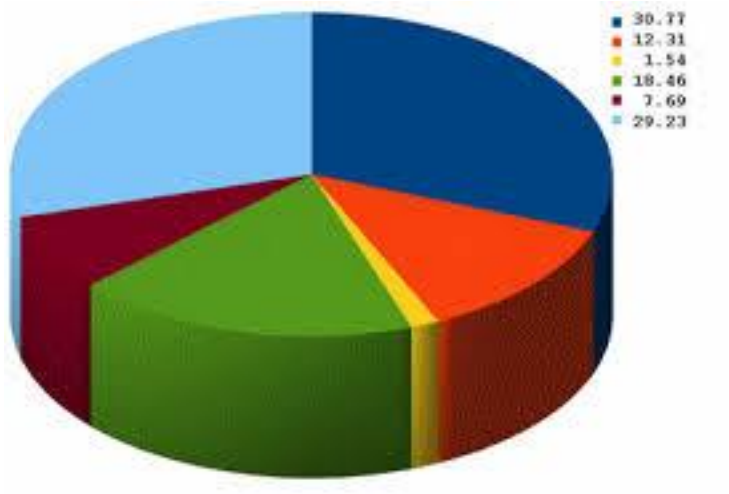
Neo4J,Infinite Graph, HyperGraphDB,...

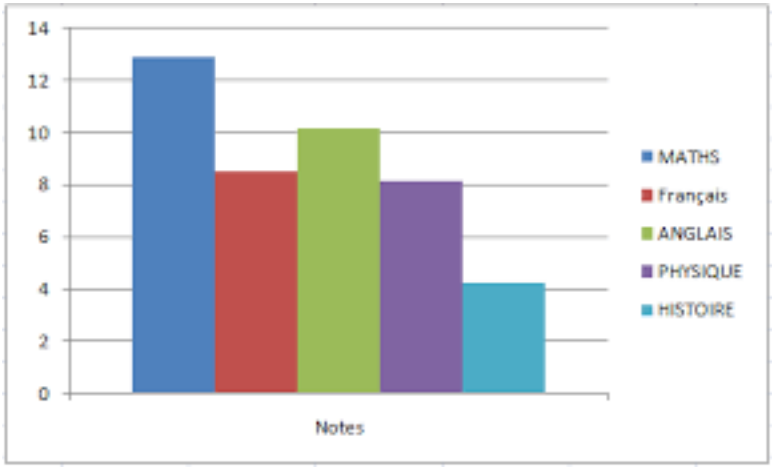
Graph Databases: Book



http://graphdatabases.com/

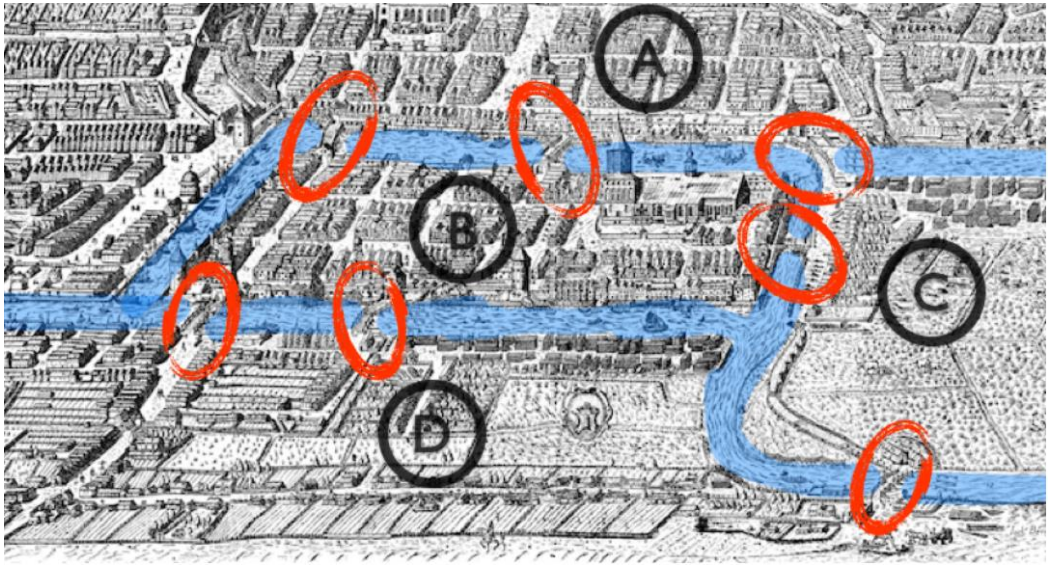
These are not graphs!





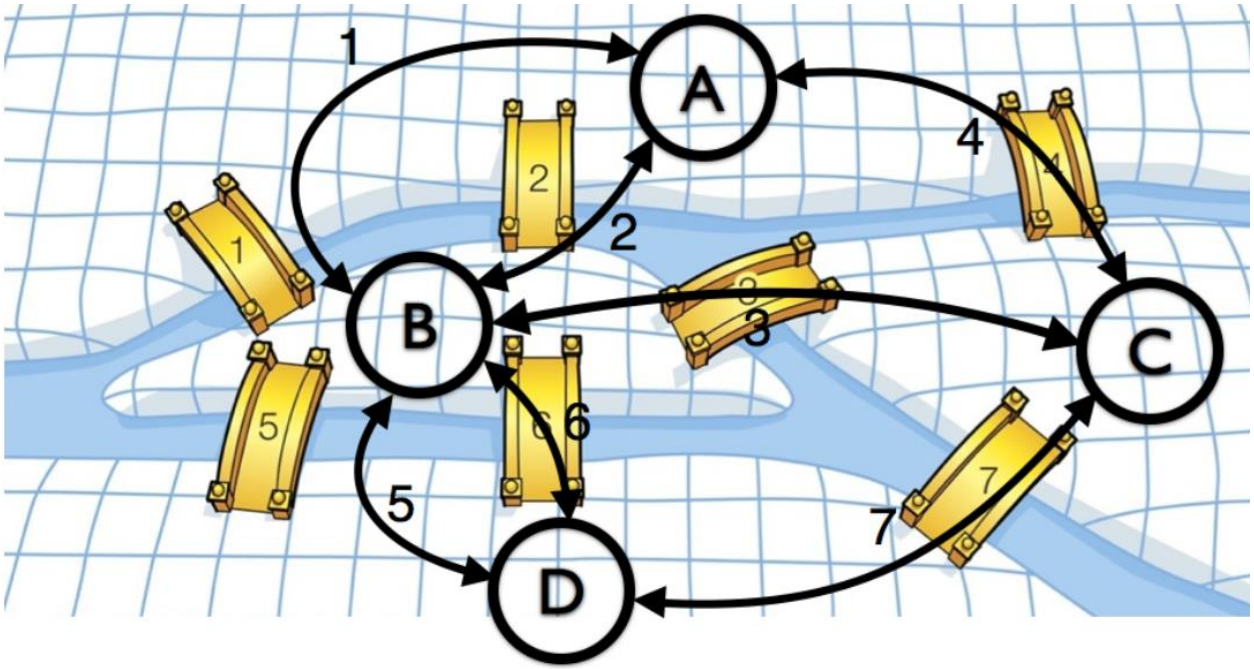
Leonhard Euler 1707-1783

- **Swiss Mathematician**
- **The problem of the seven bridges in Königsberg: The problem was to find a walk from a given point which would return to this point by passing once by each of the seven bridges in the town of Königsberg.**



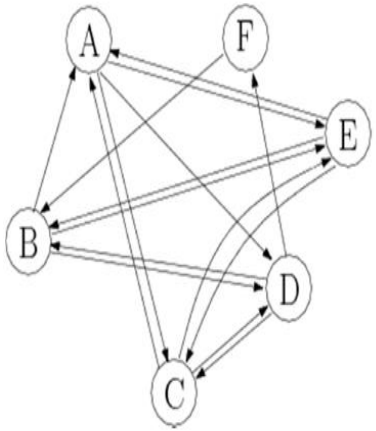
7 bridges of Königsberg – Model

- **It is a graph with nodes (vertices) and relationships (edges)**

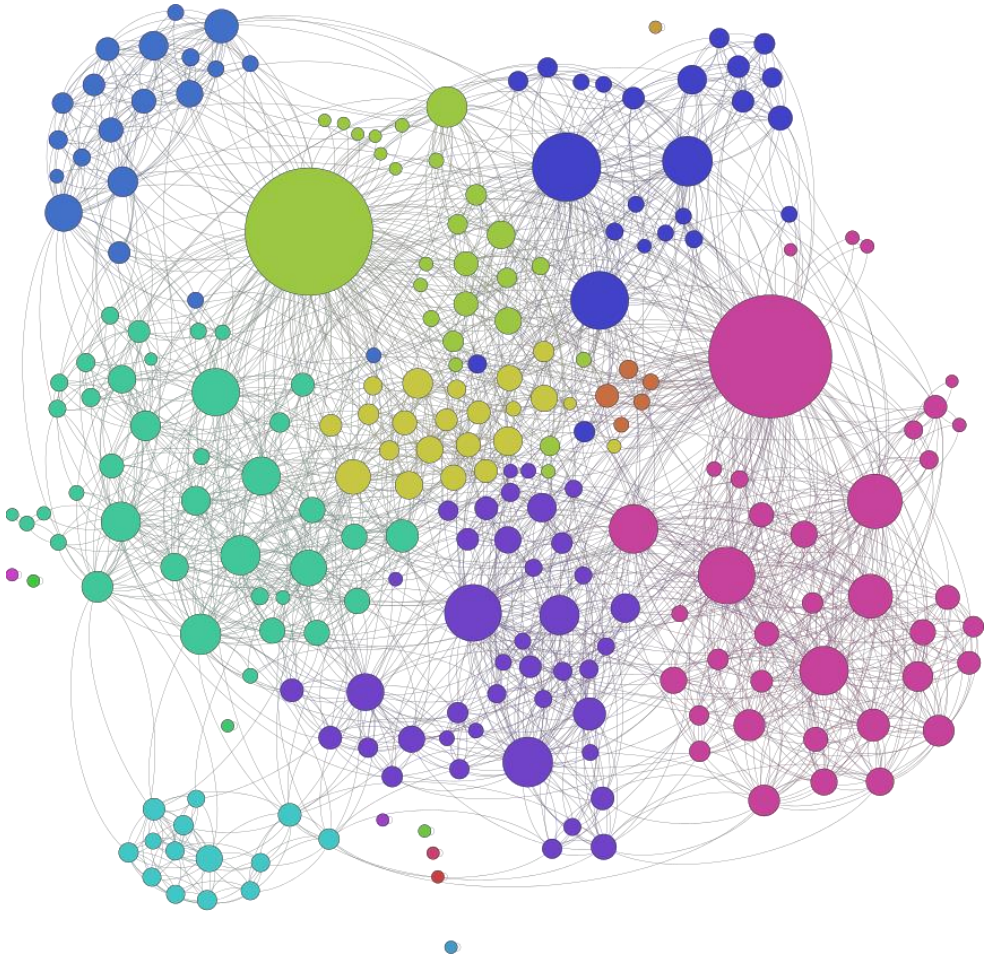


Graphs

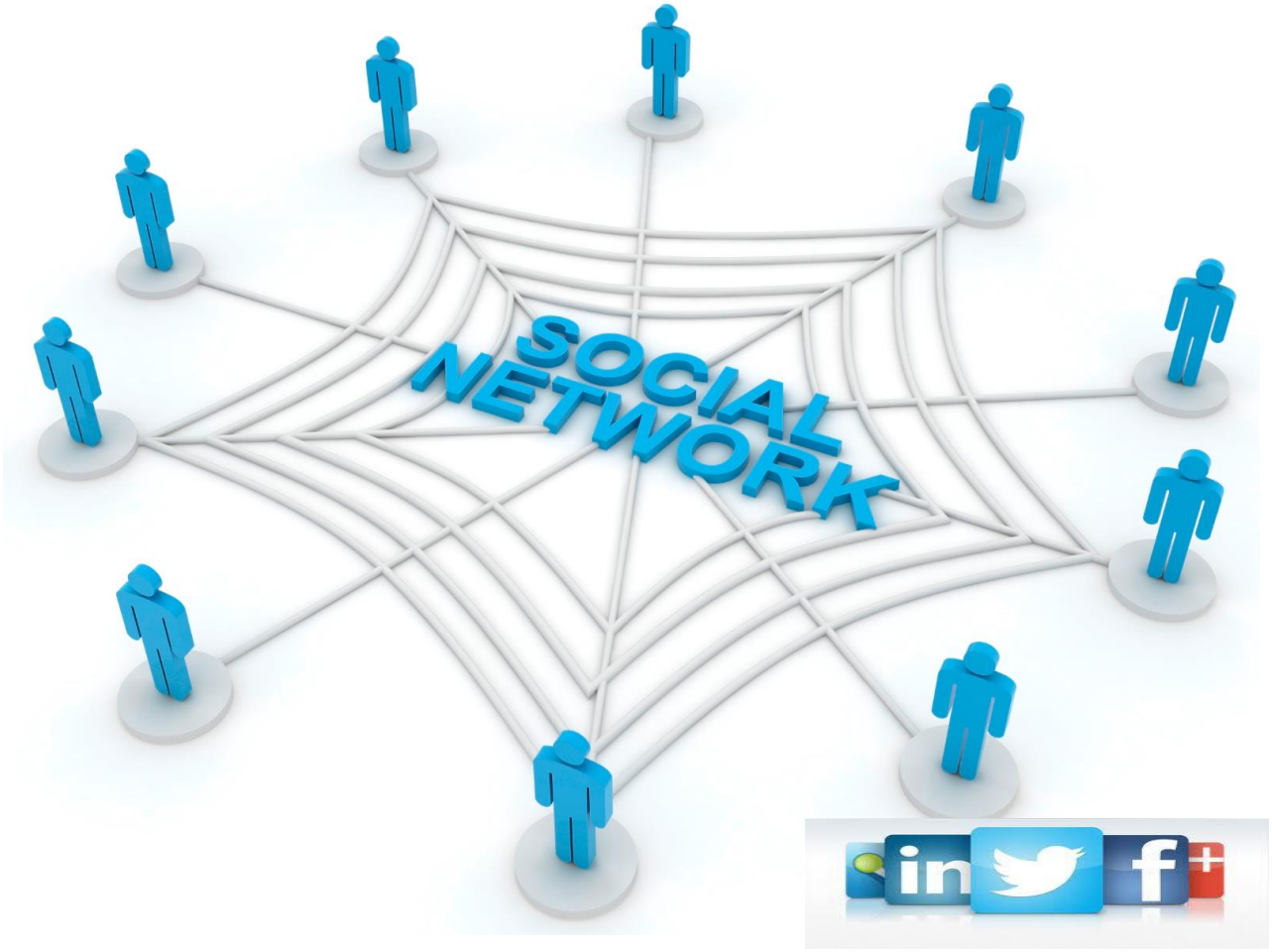
- **Represents a network**
- **A graph is a collection of nodes (vertices) and edges (relationships) that connect pairs of nodes.**
- **a graph is the general data structure suitable for any data that is related**



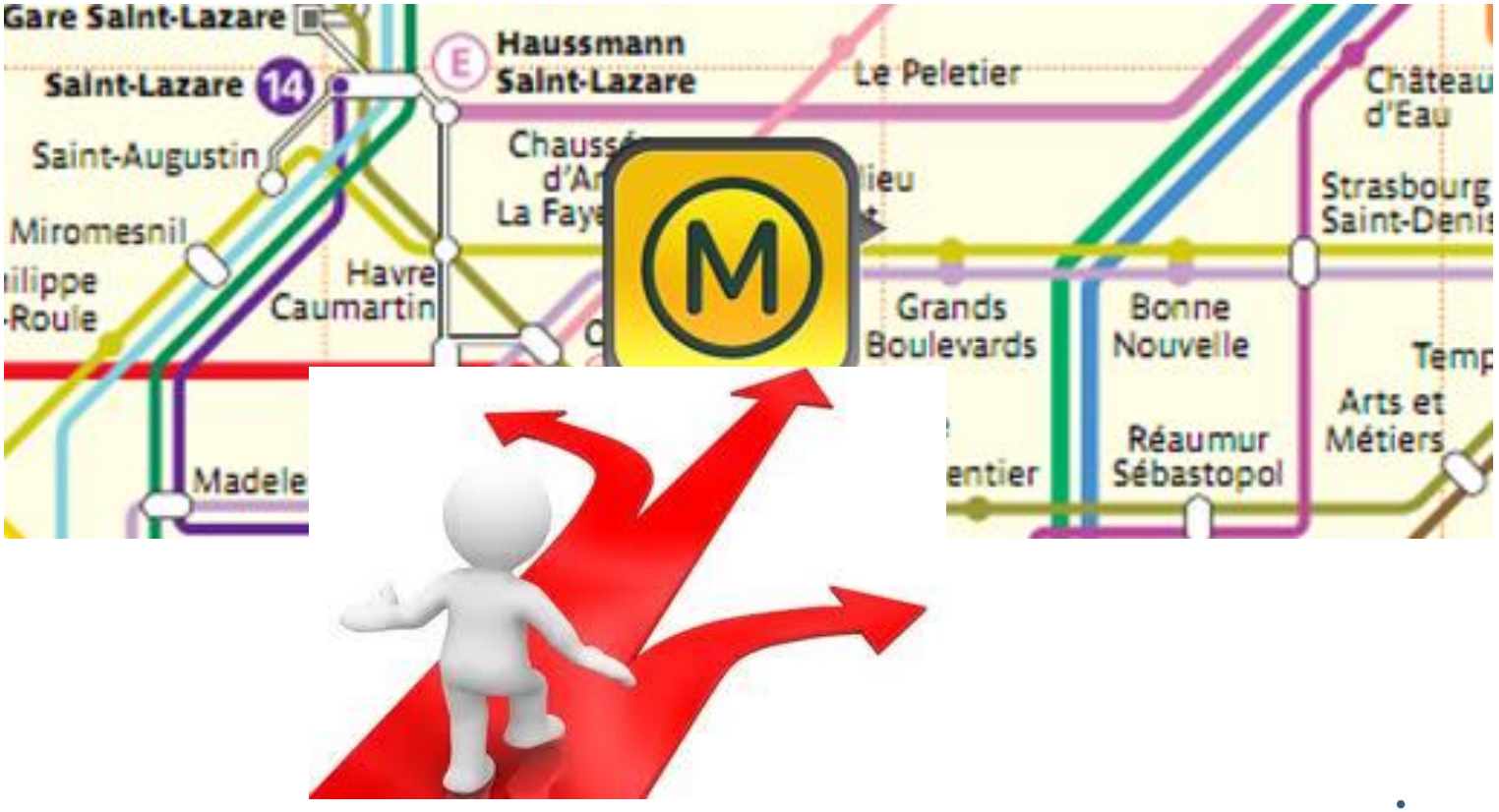
Graphs are everywhere



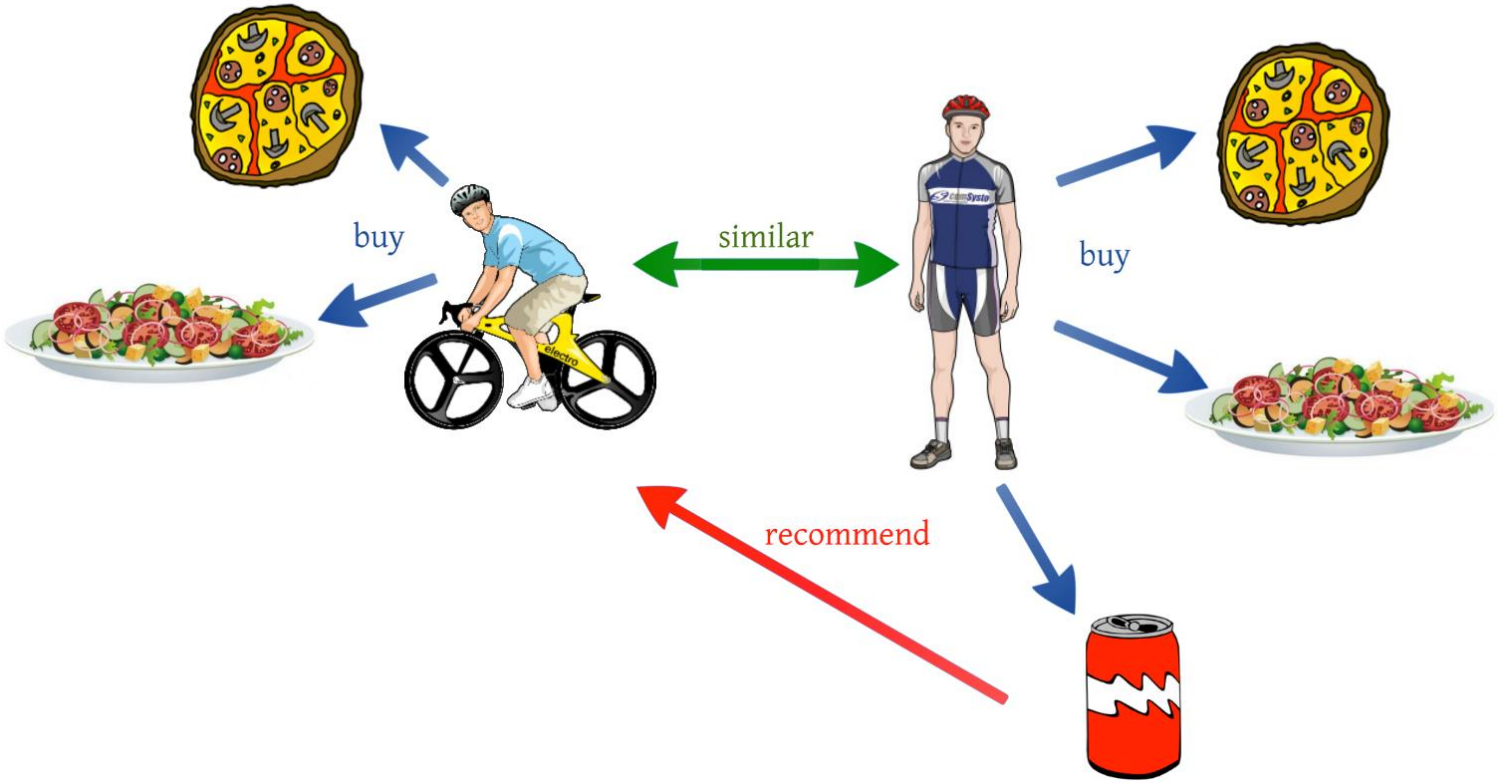
Social networks



Itinerary

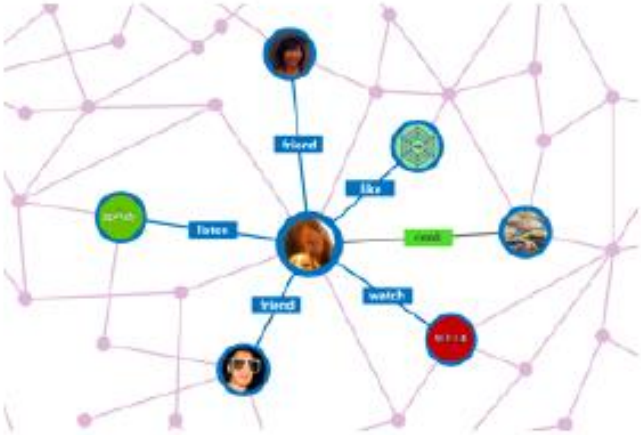

Gare Saint-Lazare
Saint-Lazare
14
Haussmann
Saint-Lazare
E
Le Peletier
Château
d'Eau
Saint-Augustin
Strasbourg
Saint-Denis
Miromesnil
Havre
Caumartin
Grands
Boulevards
Bonne
Nouvelle
Arts et
Métiers
Réaumur
Sébastopol
Madele
M

Recommender systems



Early adopters of Graph DB

facebook

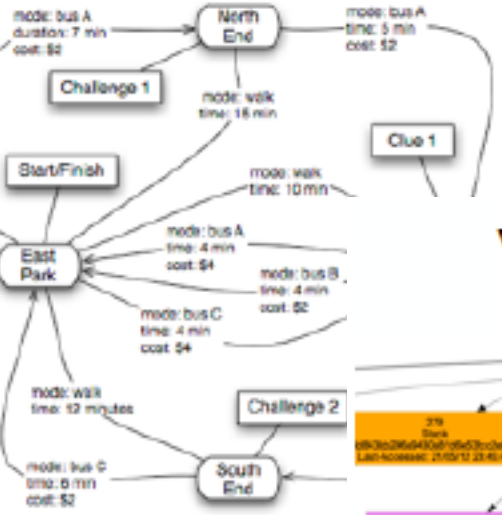


Google

Emergent Graph in other industries

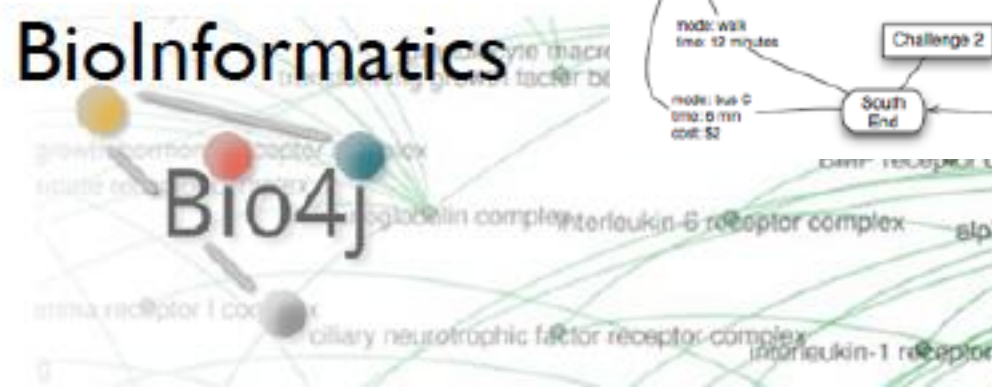
Insurance Risk Analysis

Geo Routing (Public Transport)



Gene Sequencing

BioInformatics



Web Browsing

Graph DB

- **Relational Databases Lack Relationships**
- **The other types of NOSQL Databases Also Lack Relationships**
- **Graph Databases Embrace Relationships**
- **Optimized for the connections between records**
- **Fast at querying across records**
- **A well known implementation: Neo4J**

  - A relational database may tell you how many books you sold last quarter,
  - But a graph database will tell your customer which book they should buy next.

RDBMS vs Graph DB

Person	
ID	Person
1	Alice
2	Bob
…	…
99	Zach

PersonFriend	
PersonID	FriendID
1	2
2	1
2	99
…	…
99	1

Bob's friend?

```
SELECT Person.person
FROM Person JOIN PersonFriend
  ON Person.person = PersonFriend.person
WHERE PersonFriend.friend = 'Bob'
```

 Alice

Comparison

- ~1000000 persons
- Average of 50 friends per person

Depth	RDBMS execution time (s)	Neo4j execution time (s)	Records returned
2	0.016	0.01	~2500
3	30.267	0.168	~110,000
4	1543.505	1.359	~600,000
5	Unfinished	2.132	~800,000

Strengths and weaknesses

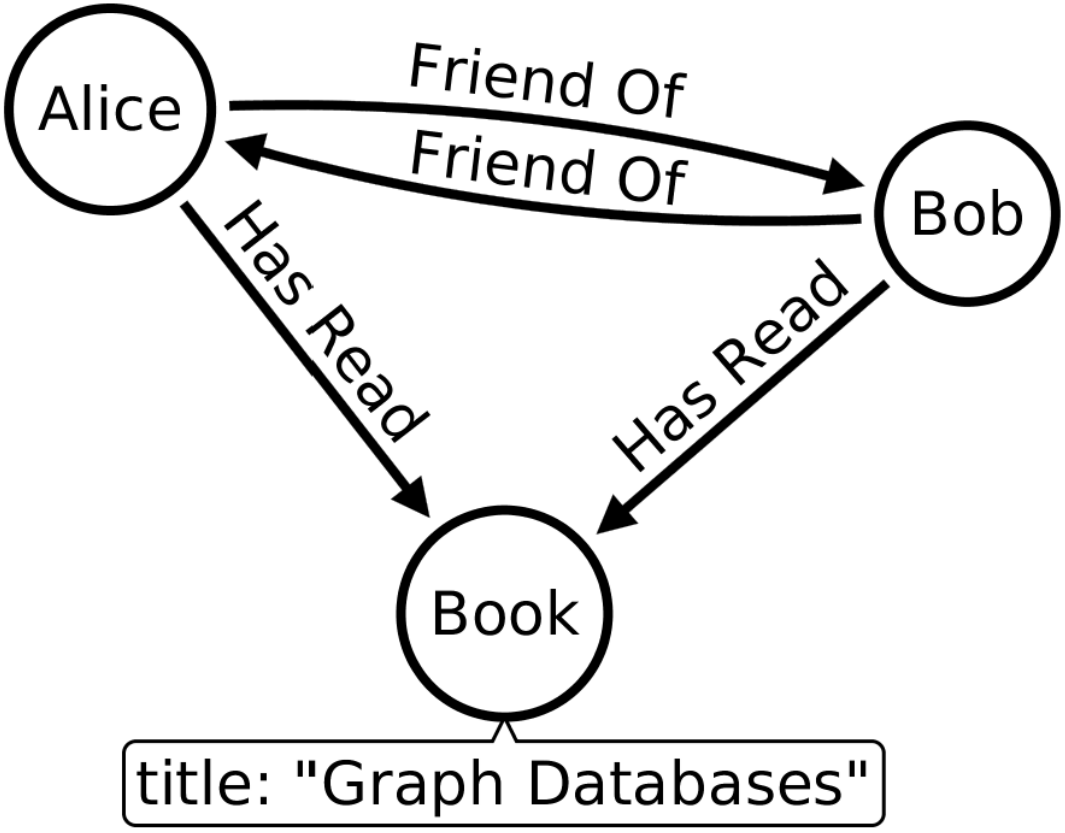
- **Pros :**
  - Powerful data model
  - For linked data
  - Efficient query models: Cypher, SPARQL
  - Algorithms from graph theory, such as shortest path, least expensive (Dijkstra or A *) ...
- **Cons :**
  - Relatively new concept
  - Scalability (Data Volume)

Use cases

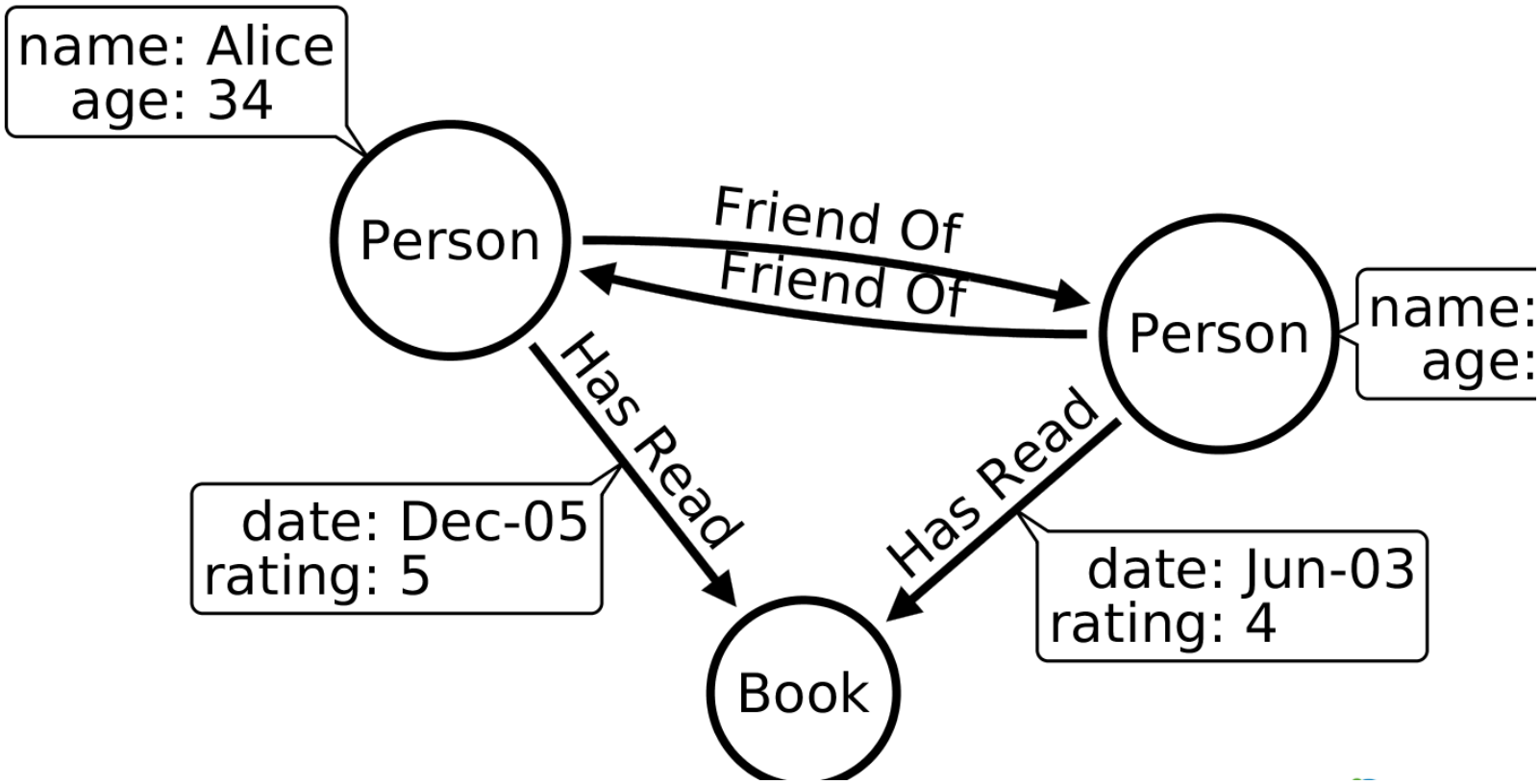
- **Use for:**
  - Network analysis (social, influencer, risk, fraud, etc.)
  - Recommendation engines
  - Heterogeneous data integration
  - Information discovery/ semantic search

About Data

Alice and bob

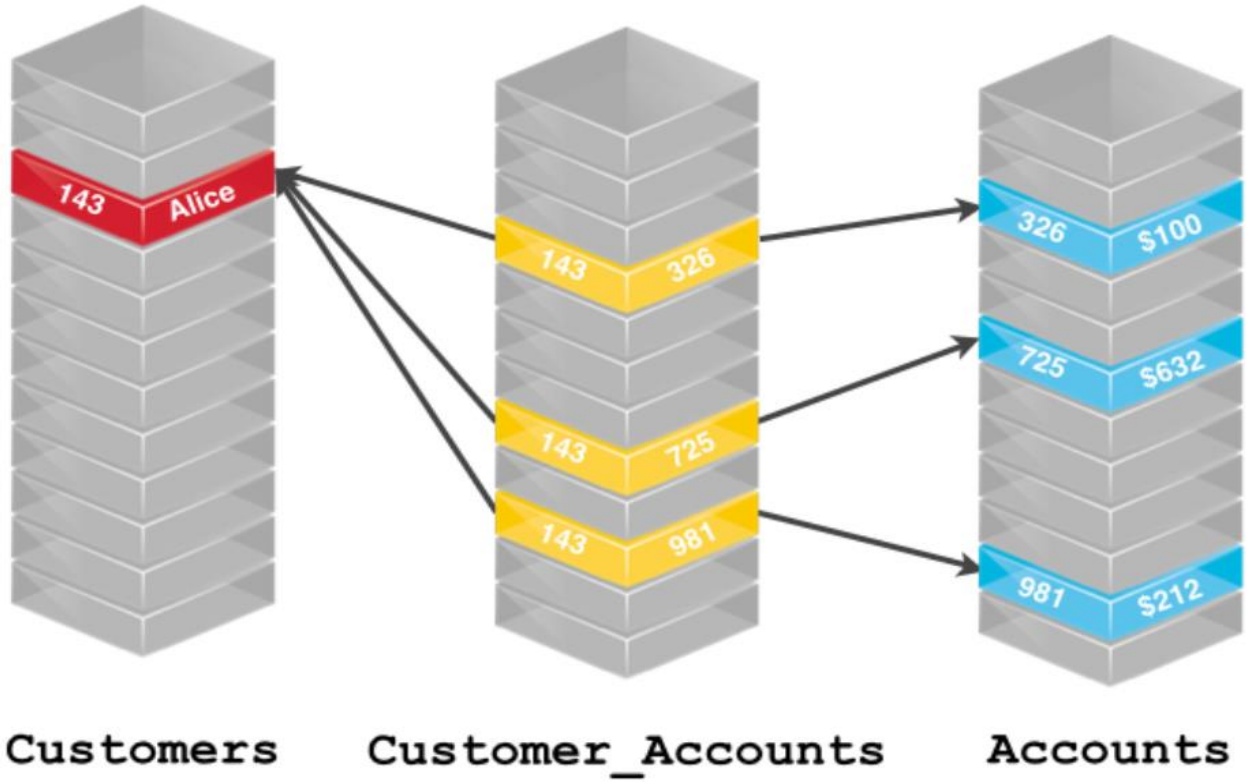


Adding details

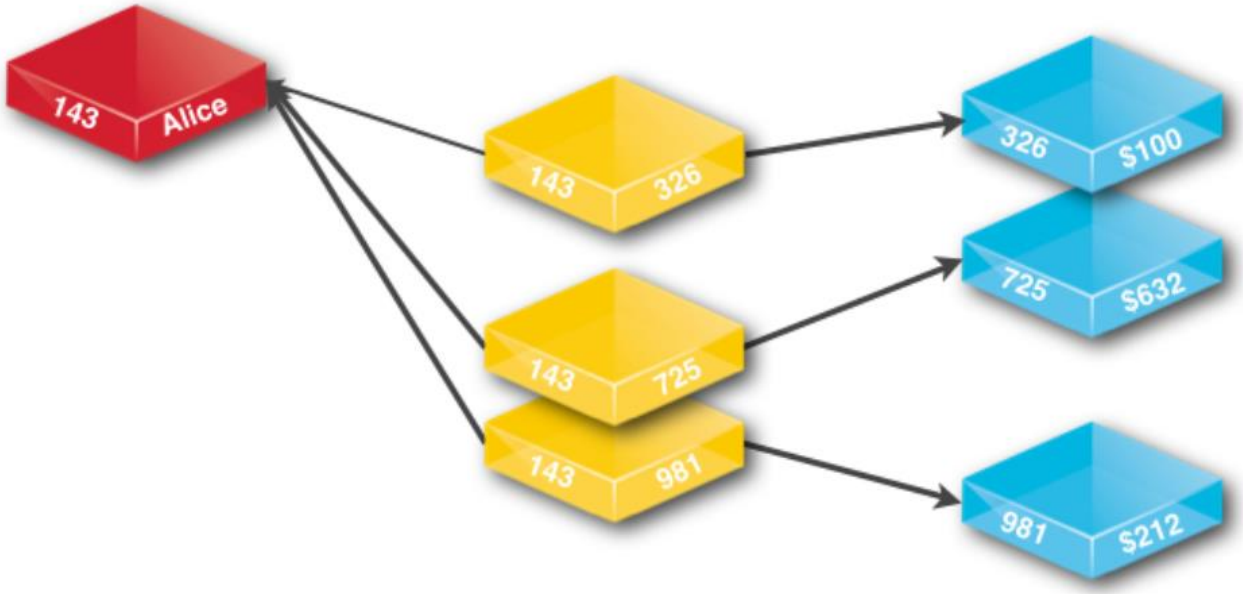


Graph DB Vs RDBMS

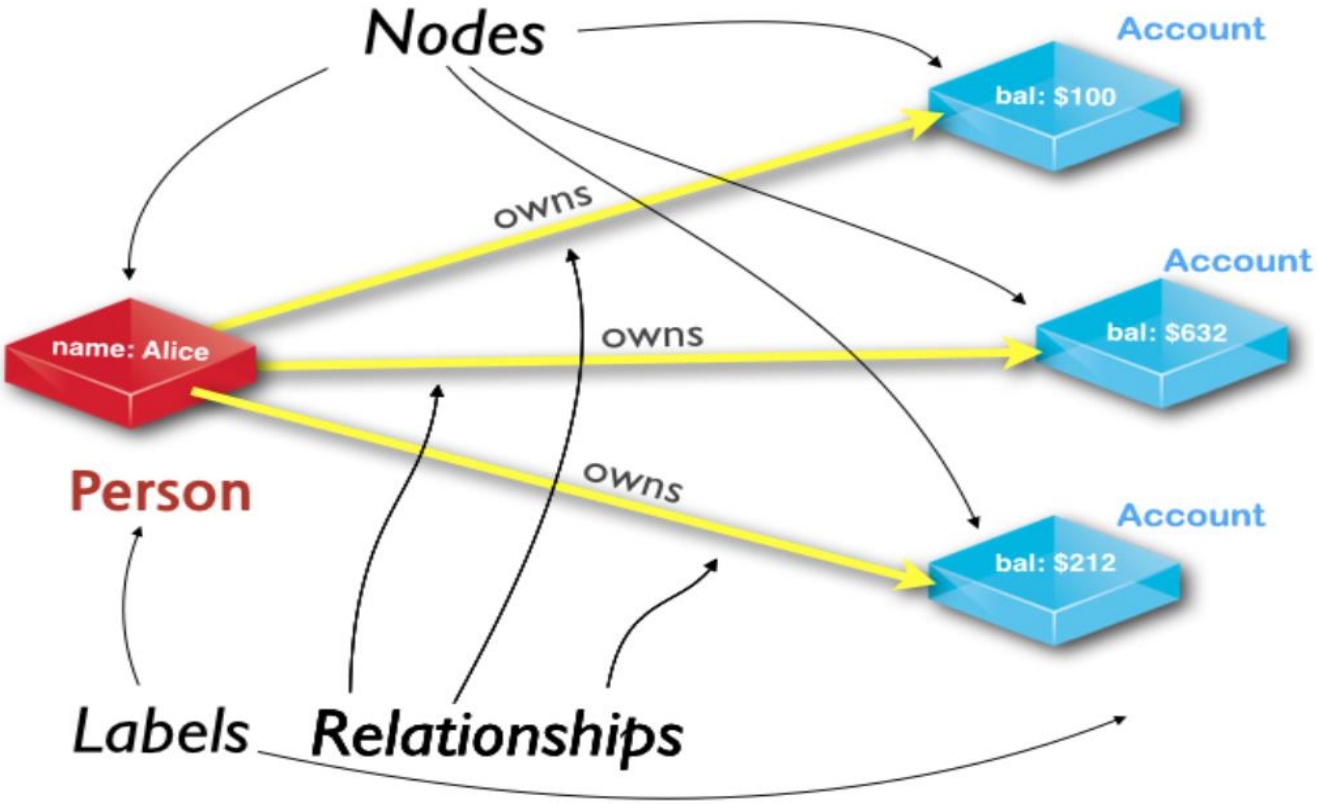
Relational


143
Alice
143
326
143
725
143
981
326
$100
725
$632
981
$212
Customers
Customer_Accounts
Accounts

Data



It is a graph

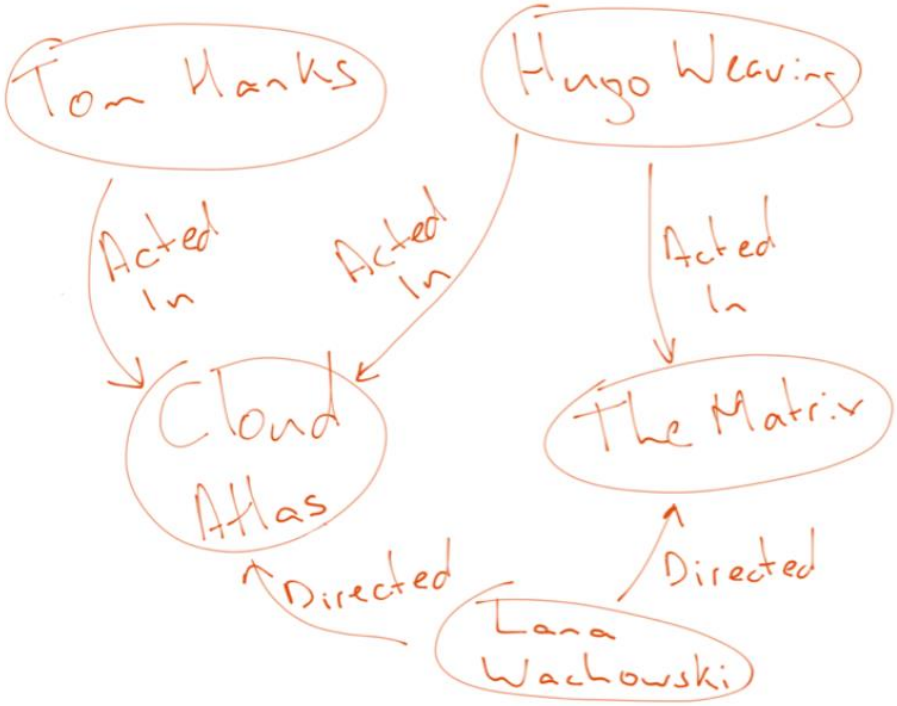

Nodes
Account
bal: $100
owns
Account
name: Alice
owns
bal: $632
Person
owns
Account
bal: $212
Labels
Relationships

The problem with RDBMS

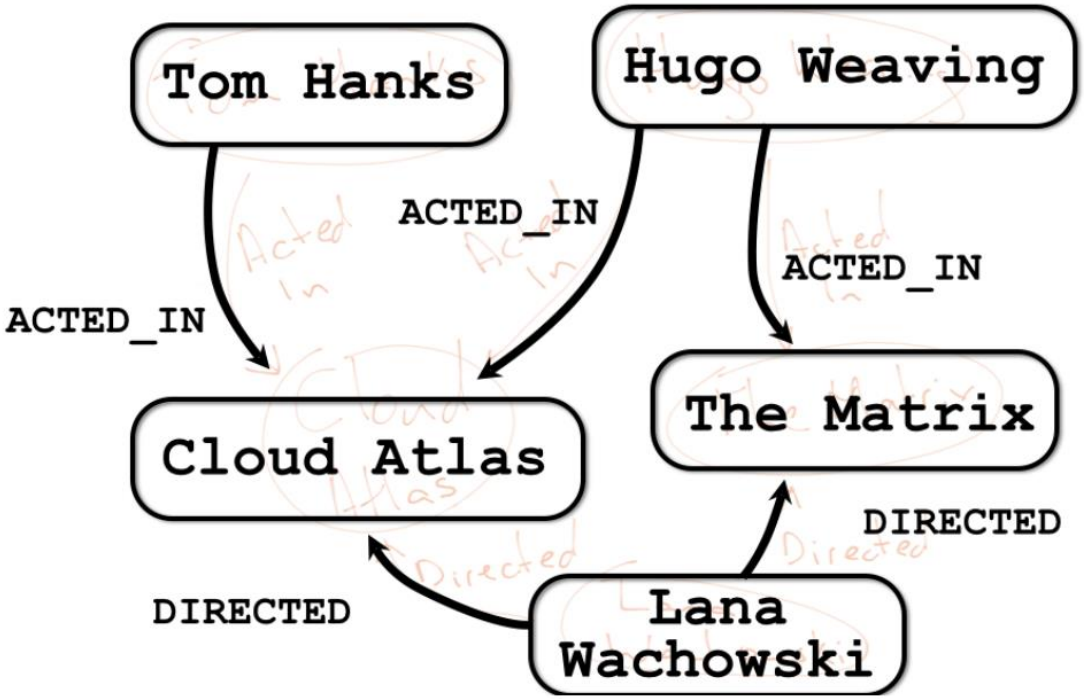
- all JOINs are executed every time your query (traverse) the relationship
- executing a JOIN means to search for a key in another table
- more entries => more lookups => slower JOINs

Graph modeling

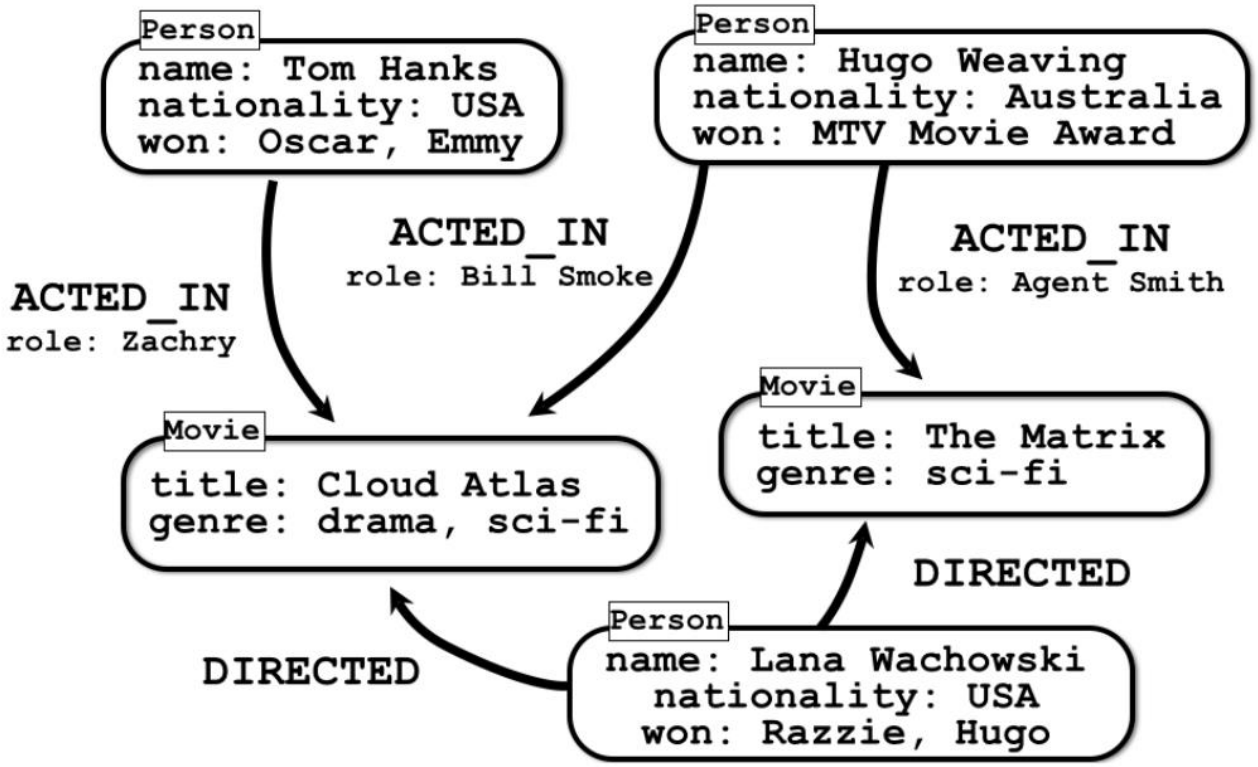
Everything starts with a whiteboard



More precisely

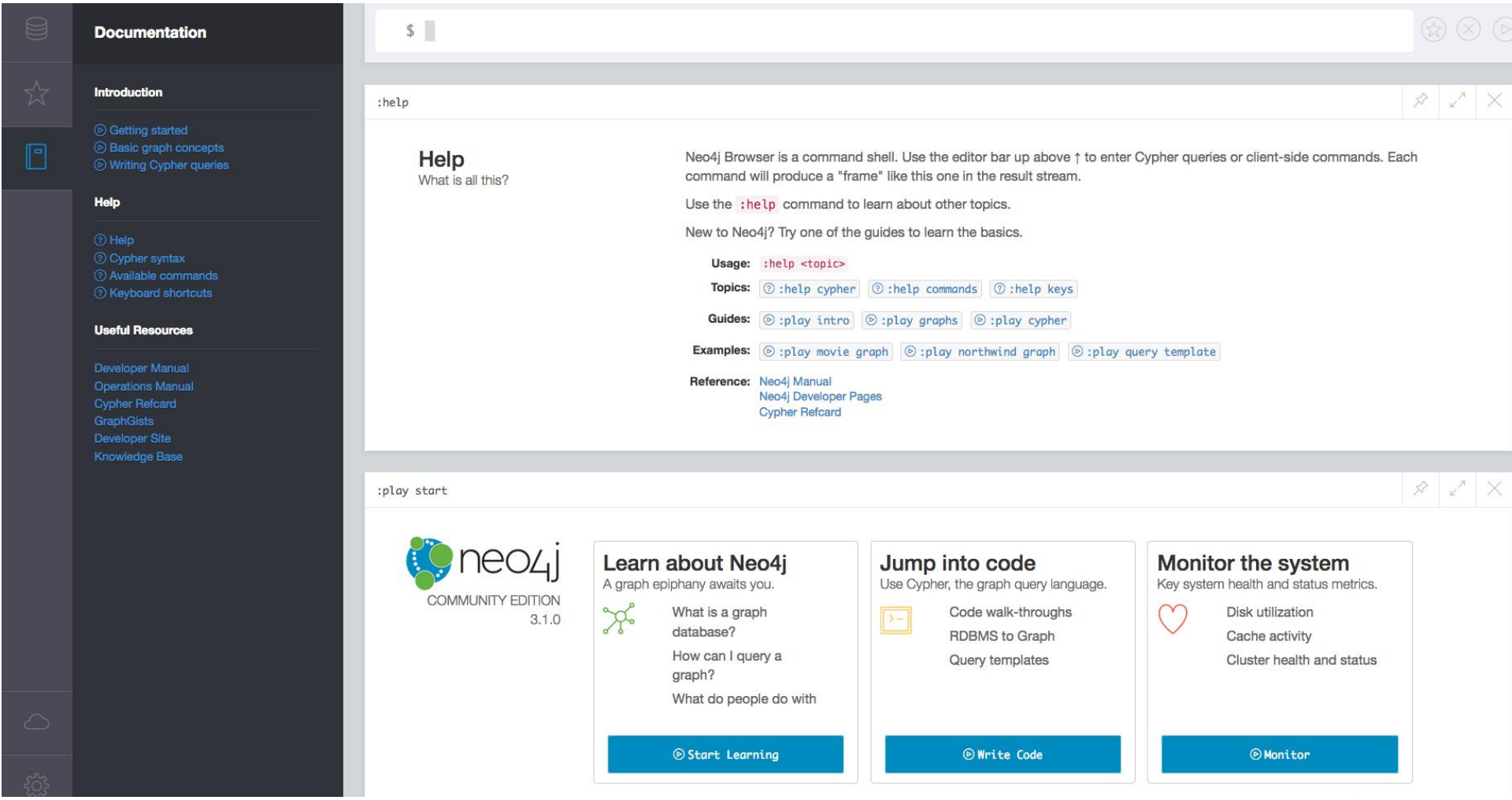


Adding properties and labels



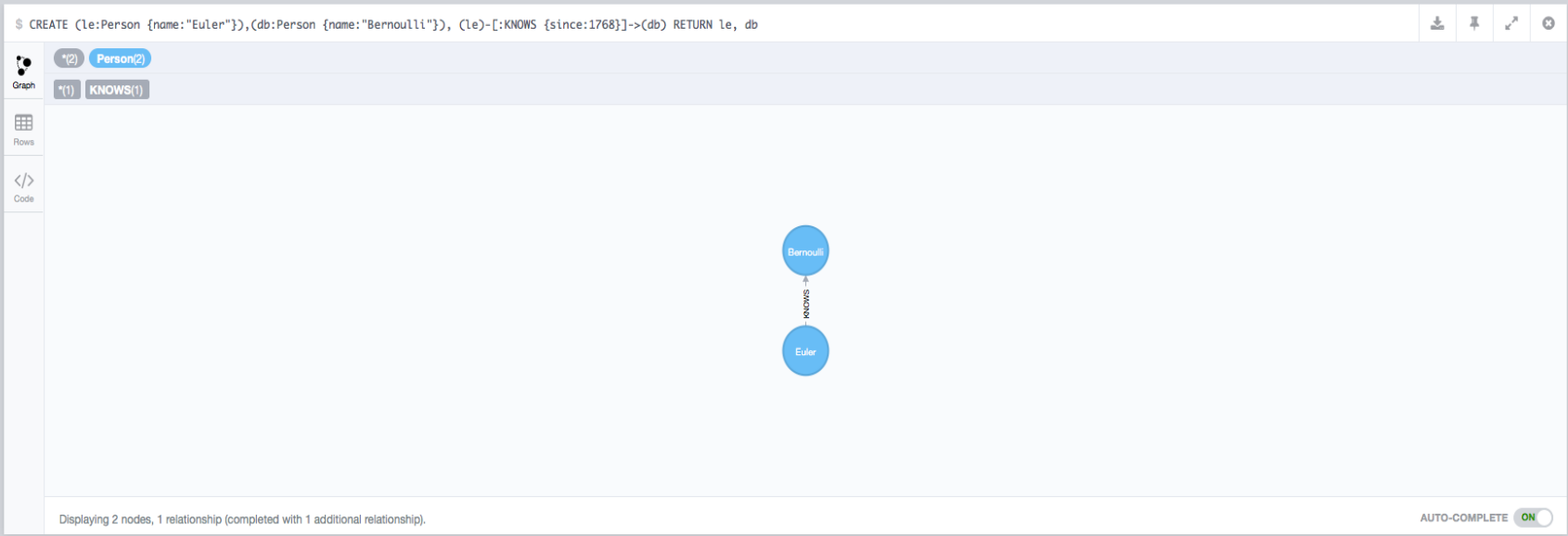
Neo4J browser

Help


Documentation
Introduction
Getting started
Basic graph concepts
Writing Cypher queries
Help
Help
Cypher syntax
Available commands
Keyboard shortcuts
Useful Resources
Developer Manual
Operations Manual
Cypher Refcard
GraphGists
Developer Site
Knowledge Base
$
:help
Help
What is all this?
Neo4j Browser is a command shell. Use the editor bar up above ↑ to enter Cypher queries or client-side commands. Each command will produce a "frame" like this one in the result stream.
Use the :help command to learn about other topics.
New to Neo4j? Try one of the guides to learn the basics.
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

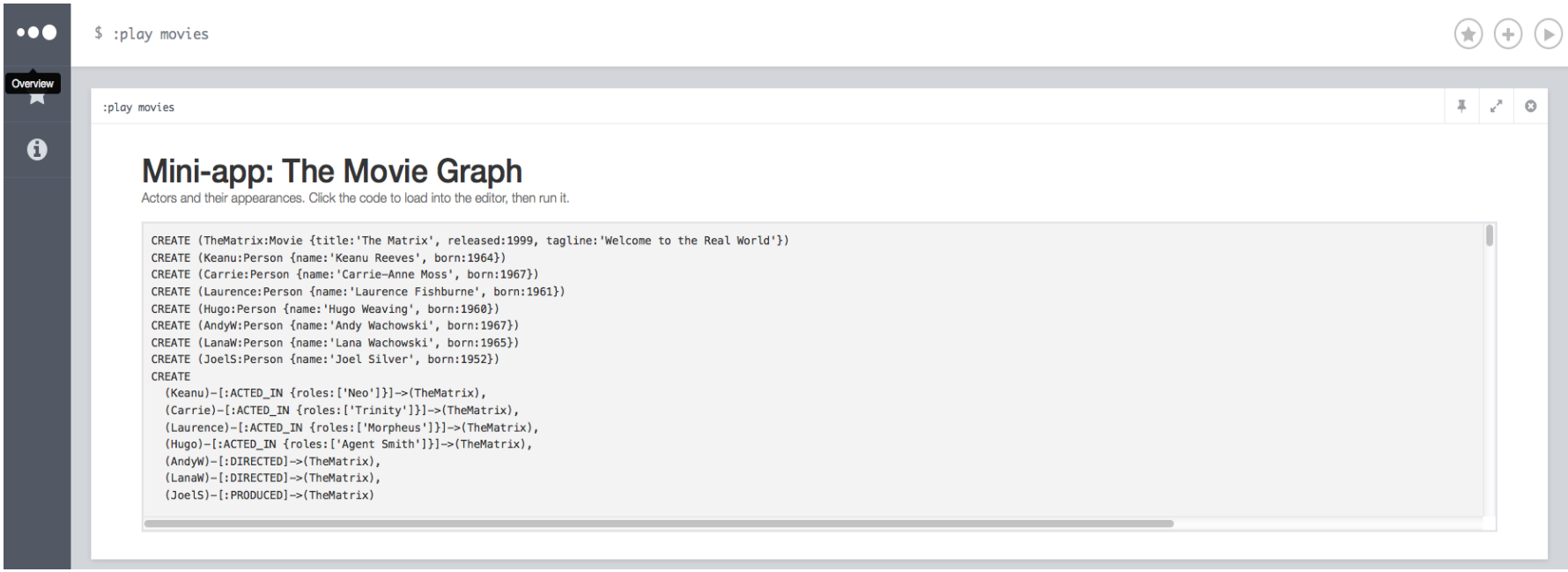
Queries and visualization

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



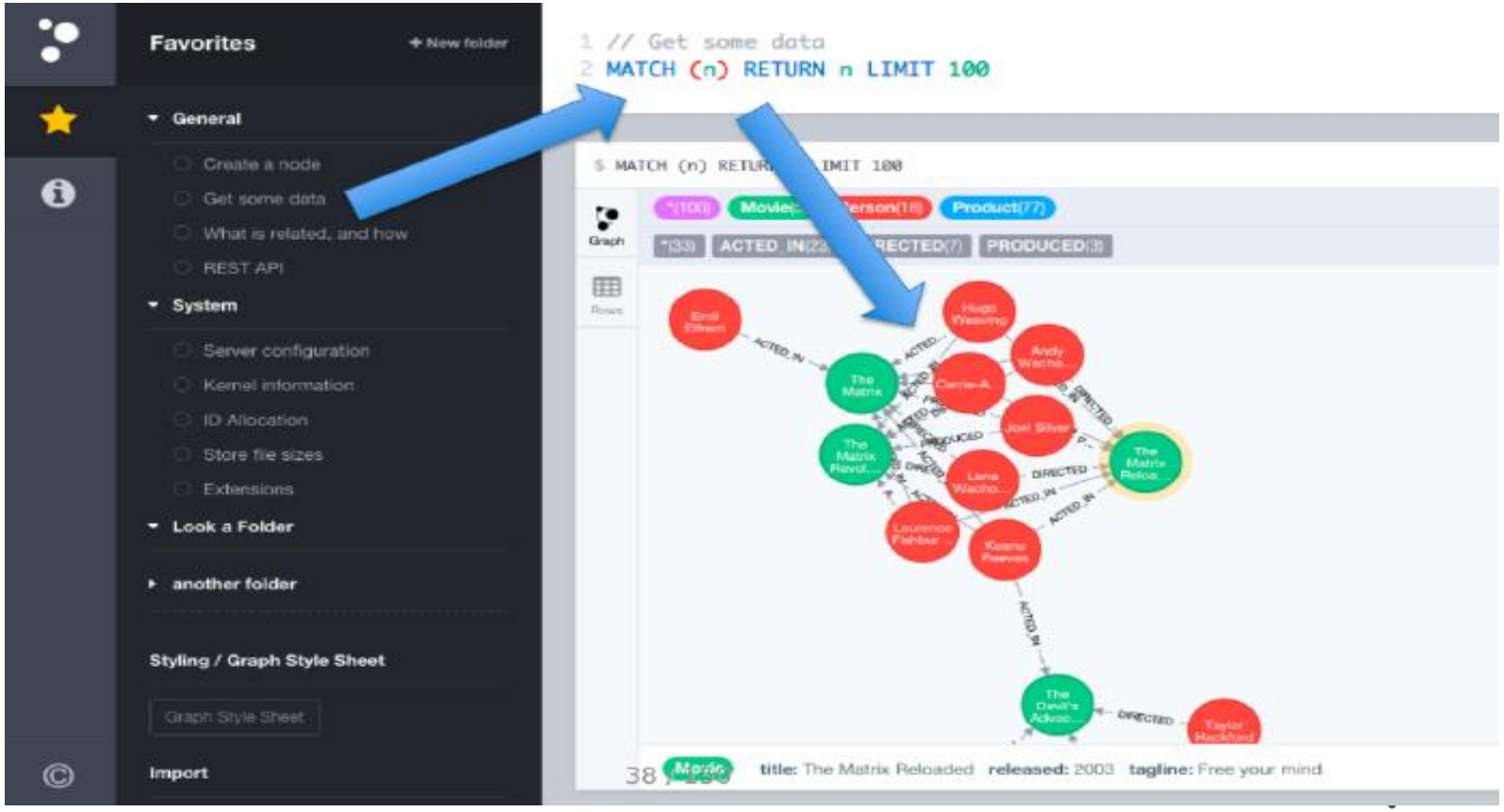
Import dataset

:play movies



Neo4j browser – Display data

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

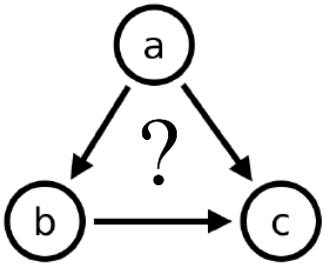


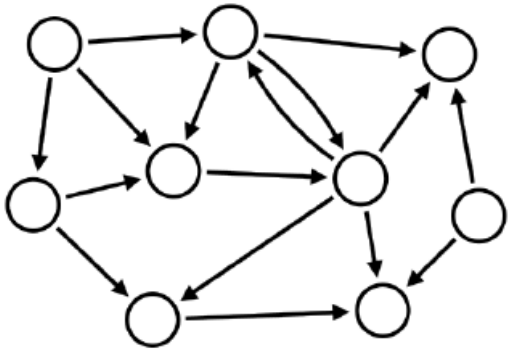
Intro to cypher

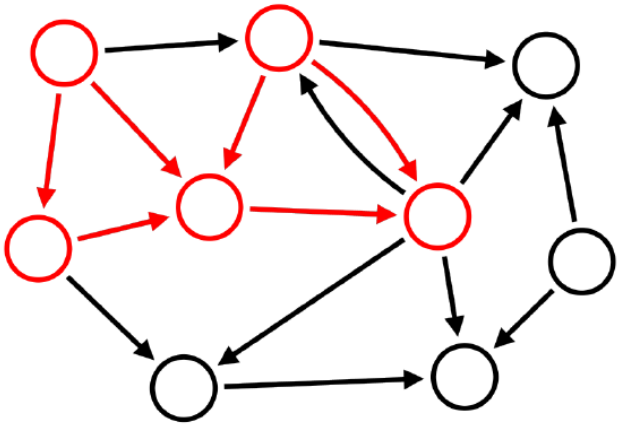
Cypher - definition

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

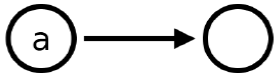
Principle





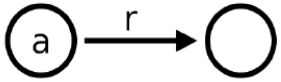


Two nodes, one relationship



(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)
```

```
MATCH (a:Movie { title: 'Wall Street' })
OPTIONAL MATCH (a)-->(x)
```

Two nodes, a known relationship

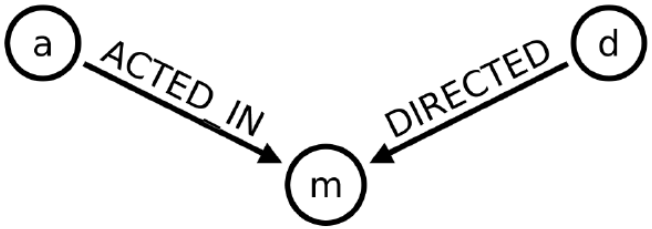
(a)-[:ACTED_IN]-> (m)

```
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
```

Homework : Queries on « Movies »

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

Alias

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

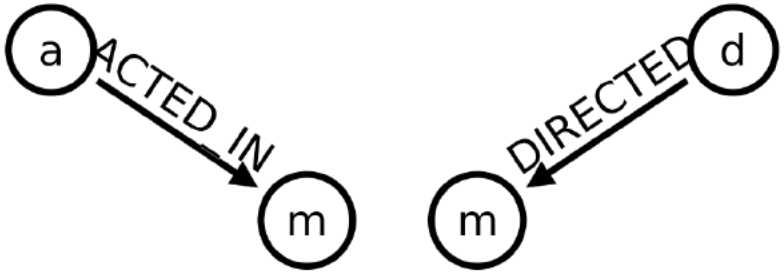
actor	movie	director
"Keanu Reeves"	"The Matrix"	"Andy Wachowski"
"Keanu Reeves"	"The Matrix Reloaded"	"Andy Wachowski"
"Noah Wyle"	"A Few Good Men"	"Rob Reiner"
"Tom Hanks"	"Cloud Atlas"	"Andy Wachowski"

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"

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

```
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
```

- 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, (a.born-tom.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**

```
MATCH (d:Person)-[:DIRECTED]->(m:Movie)
WITH d, COUNT(m) AS Totalfilms
ORDER BY Totalfilms DESC
LIMIT 5
RETURN d.name AS Nomdirector,Totalfilms
```

Update with CYPHER

MISE A jour avec 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
```

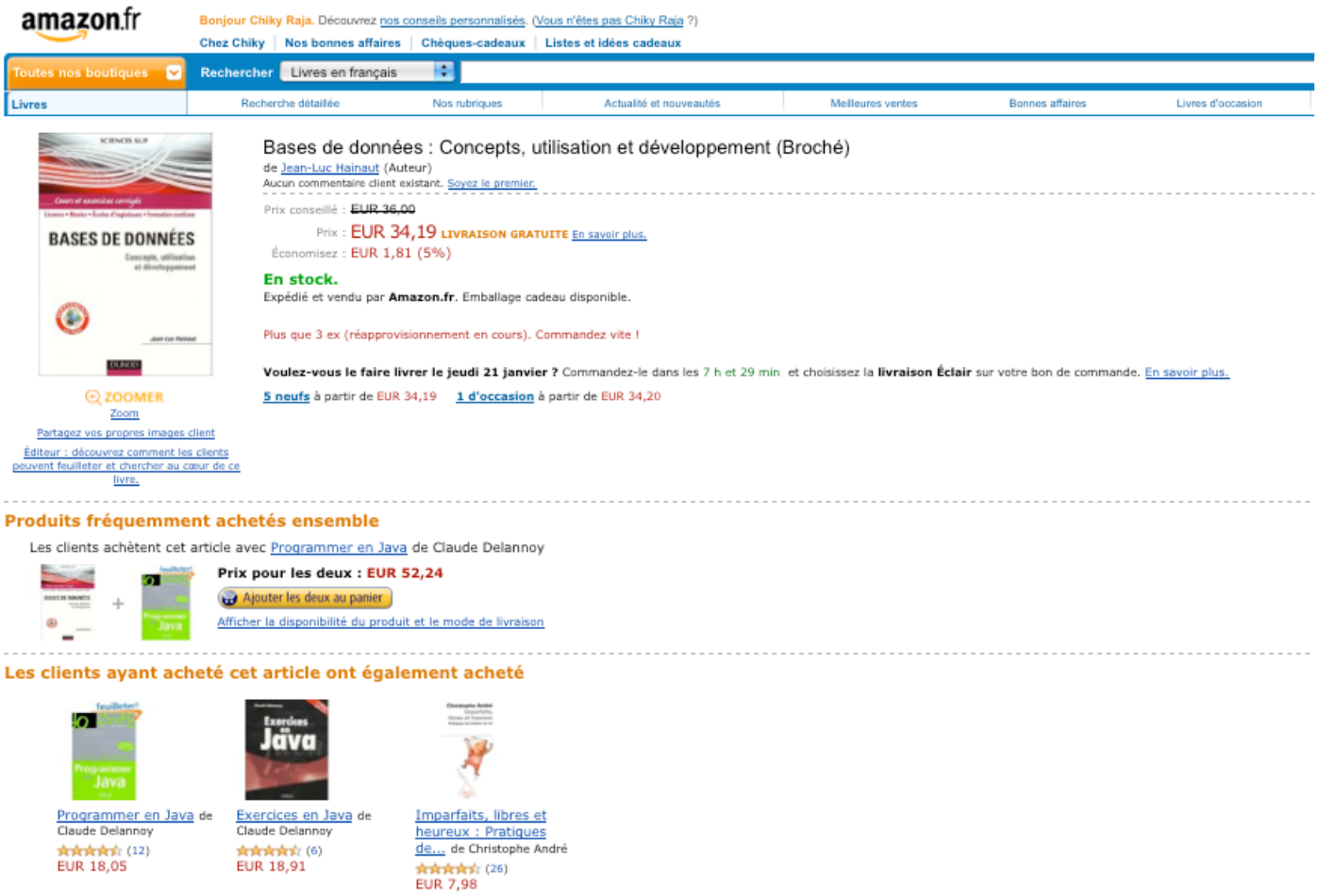
Exercise

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

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

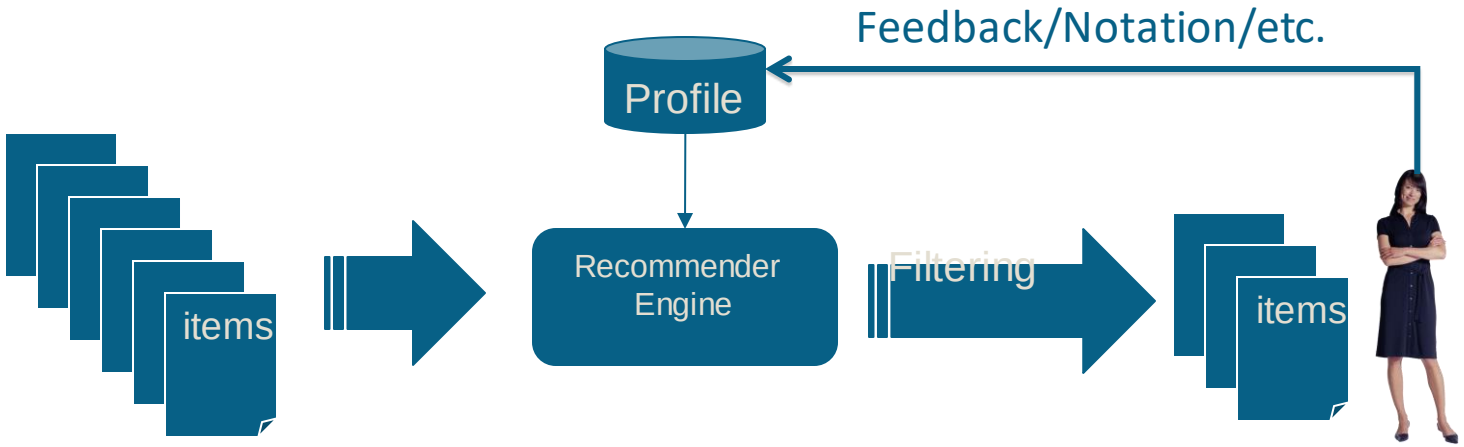
Recommendation

Recommendations: Overview



Definition

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



Recommender Systems

USERS | ITEMS | FEEDBACK | ATTRIBUTES | EXTERNAL

Collaborative Filtering

Content-Based

Hybrid

LinkedIn amazon NETFLIX MENDELEY

isep

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**

```
MATCH (keanu:Person)-[:ACTED_IN]->()<-[:ACTED_IN]-(c),(c)-[:ACTED_IN]->()<-[:ACTED_IN]-(coc)
WHERE keanu.name="Keanu Reeves"
AND coc <> keanu
AND NOT ((keanu)-[:ACTED_IN]-()<-[:ACTED_IN]-(coc))
RETURN coc.name, count(coc)
ORDER BY count(coc) DESC
LIMIT 3
```

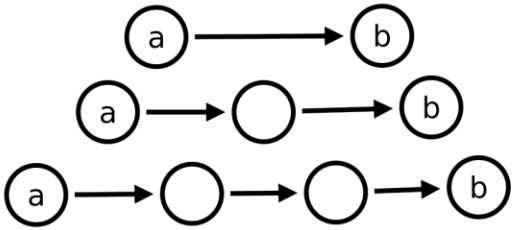
Go further…

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 demi.name="Demi Moore"
RETURN length(p);
```

Neo4J-users


Finance
Télécom
Santé
RH
Média
Web Social
Industrie & Logistique
UBS
First Data.
ADVENT
NOMURA
veda
die Bayerische
ice
cisco
hp
SFR
EarthLink
telenor
maaii
Let's connect
EarthLink
GoodStart Genetics
janssen
ZEPHYR
curaspan
doximity
Care.com
careerbuilder
InfoJobs
EQUILAR
SHRM
NATIONAL GEOGRAPHIC
LIFECHURCH.TV
codex
Livestation
Perigee
TechCrunch
classmates
eHarmony
Hinge
meetic
SNAP
ebay now
noble group
SNCF
KiwiRail
ConocoPhillips
Impact Technologies
Jeux
Commerce
Business Services
Information Services
Bally
bwin.party
gamesys
YELAGO
Walmart
Juice PLUS+
cobrain
compete
Cerved Group
scribestar
LOCKHEED MARTIN
Lufthansa Systems

Thanks!