NoSQL graph database

Matteo Francia, Ph.D.

m.francia@unibo.it

Thanks to Enrico Gallinucci and Oscar Romero for these slides

“WITHOUT DATA, YOU’RE JUST ANOTHER PERSON WITH AN OPINION”

W. Edwards Deming, American Statistician

Graph Data Model in a Nutshell

Occurrence-oriented

- It is a schemaless data model
  - There is no explicit schema
  - Data (and its relationships) may quickly vary
- Objects and relationships as first-class citizens
  - An object o relates (through a relationship r) to another object o’
    - Such relationship is often known as a triple (o r o’)
  - Both objects and relationships may contain properties
- Built on top of the graph theory
  - Euler (18th century)
  - More natural and intuitive than the relational model to deal with relationships

Notation (I)

A **graph** G is a set of nodes and edges: G (N, E)

N - **Nodes** (or vertices): n1, n2, … Nm

E - **Edges** are represented as pairs of nodes: (n1, n2)

- An edge is said to be incident to n1 and n2
- Also, n1 and n2 are said to be adjacent
- An edge is drawn as a line between n1 and n2
- Directed edges entail direction: from n1 to n2
- An edge is said to be **multiple** if there is another edge exactly relating the same nodes
- An hyperedge is an edge inciding in more than 2 nodes.

Multigraph: If it contains at least one multiple edge

Simple graph: If it does not contain multiple edges

Hypergraph: A graph allowing hyperedges

Notation (II)

Size (of a graph): #edges

Degree (of a node): #(incident edges)

- The degree of a node denotes the node adjacency
- The neighbourhood of a node are all its adjacent nodes

Out-degree (of a node): #(edges leaving the node)

- Sink node: A node with 0 out-degree

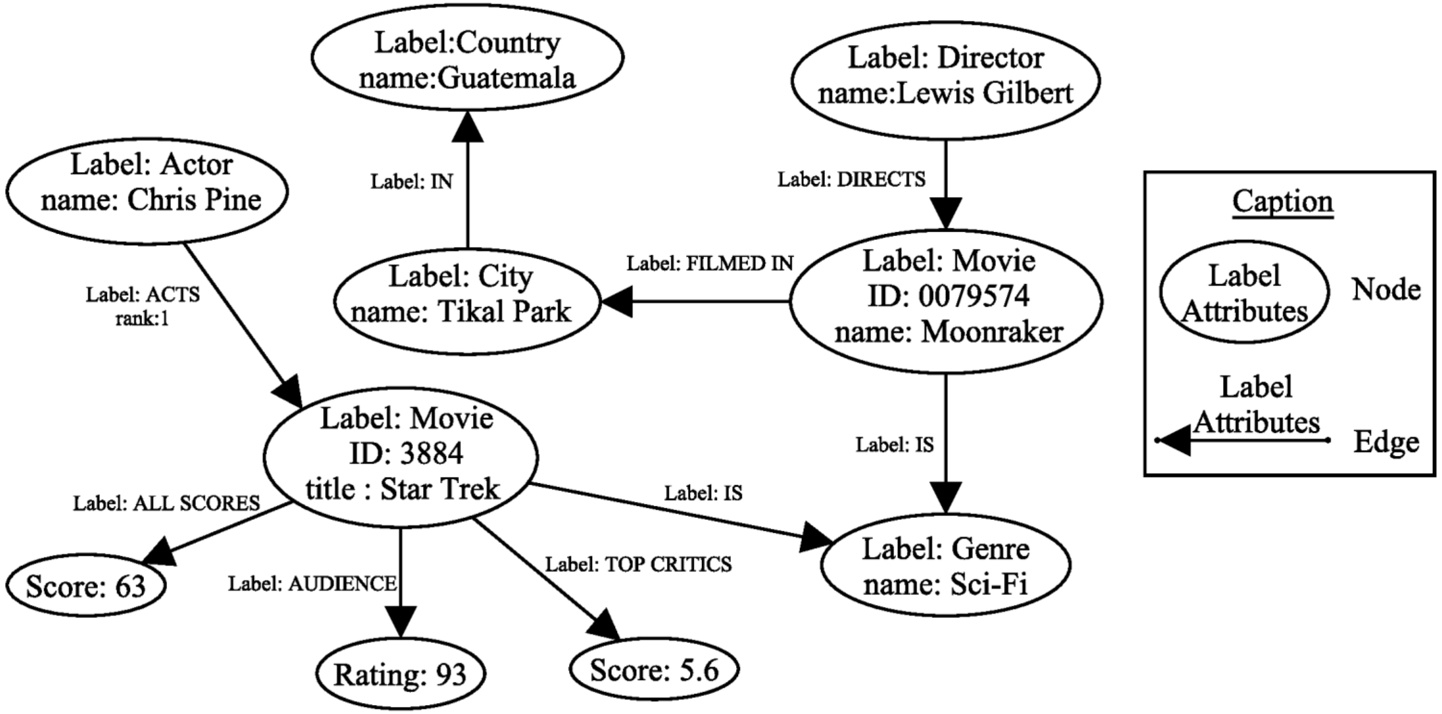
In-degree (of a node): #(incoming edges reaching the node)

- Source node: A node with 0 in-degree

Cliques and trees are specific kinds of graphs

- Clique: Every node is adjacent to every other node
- Tree: A connected acyclic simple graph

Example



Label:Country
name:Guatemala
Label: Director
name:Lewis Gilbert
Label: Actor
name: Chris Pine
Label: IN
Label: DIRECTS
Caption
Label: City
name: Tikal Park
Label: FILMED IN
Label: Movie
ID: 0079574
name: Moonraker
Label
Attributes
Node
Label: ACTS
rank:1
Label
Attributes
Edge
Label: Movie
ID: 3884
title : Star Trek
Label: IS
Label: ALL SCORES
Label: IS
Label: TOP CRITICS
Label: AUDIENCE
Label: Genre
name: Sci-Fi
Score: 63
Rating: 93
Score: 5.6

Pros and Cons

Graphs

They are occurrence-oriented

Occurrences are **pointed by / point to** related occurrences

- Graph operators do not rely on schema
- Naturally facilitate data linking

The schema information is embedded together with data

- The concept of stand-alone catalog does not exit

Purely schemaless

- Semantics are fixed by the edge / node labels

Difficult to benefit from sequential access. Typically, it relies on random accesses

By definition, it follows an Open-World assumption (i.e., assumes incomplete data)

Key-oriented models

The relational model is schema-oriented. Document-stores and key-values are schemaless databases but still rely on key-based structures

Key-oriented models need to make a strong modeling call, which unbalances the logical / physical model

- As consequence, the degree of (de)normalisation has a big impact on queries

Can naturally benefit from sequencial reads

Views are either virtual definitions or, if materialised, additional stand-alone constructs

Poor relationship semantics: the relational model only deals with FK, document-stores / key-values do not support relationships

Relational model, and most key-value / document-stores, follow a Closed-World assumption (i.e., complete data)

Graph Data Models and Data Analytics

From a data management point of view:

- Extremely flexible
- Schemaless by definition
- Facilitate data governance
- Facilitate ad-hoc transformations

From a data analytics point of view:

- Allow to exploit the data structure topology
- Sits somewhere in between descriptive and probabilistic data analysis

Showcasing Graphs

Crossing data from social networks it is possible to identify a graph like the one that follows:

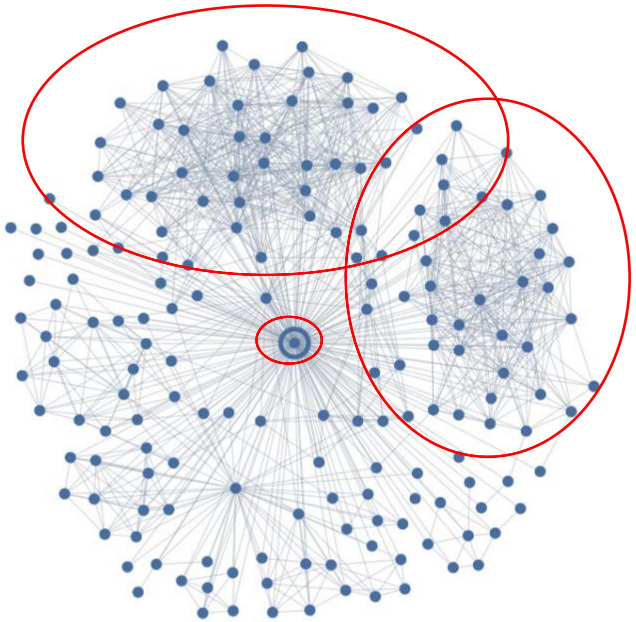
- In the centre there is a specific person *P*
- The rest are *P* connections and connections among them

Using sociology techniques…

- We can identify *P social foci*:
  - Dense clusters of friends corresponding to long periods of interaction
  - Typically, college friends, coworkers, relatives, etc.
- The *significant other* can be identified by a high *dispersion* rate
  - Highly connected with *P* connections,
  - But with a high dispersion degree wrt *P* social foci

Hypothesis: when the node with higher dispersion degree Identified is not the partner, this couple is likely to split up in a period of 60 days

L. Backstrom, J. Kleinberg. Romantic Partnerships and the Dispersion of Social Ties: A Network Analysis of Relationship Status on Facebook https://arxiv.org/pdf/1310.6753v1.pdf



Graph Data Models

There is no single graph data model

Two main families of graphs

- Property Graphs
  - Born in the database field
  - Not predefined semantics
  - Follow a Closed-World assumption
  - Generate data silos
  - Algebraic operations on top of traditional graph operations
- Knowledge Graphs
  - Born in the knowledge representation field
  - May assume the Open-World assumption
  - Facilitate data sharing and linking
  - Two main families
    - RDF and RDF(S)
      - Born in the semantic web field
      - Vocabulary-based pre-defined semantics
      - Combine algebraic operations with simple reasoning operations
    - Description Logics (DL)-based (e.g., OWL)
      - Representation of (subsets of) first-order logic
      - Pre-defined semantics based on logics
      - Reasoning operations founded in their logics nature

The Property Graph Data Model

Born in the database community

- Meant to be queried and processed
- **<u>THERE IS NO STANDARD!</u>**

Two main constructs: nodes and edges

- Nodes represent entities,
- Edges relate pairs of nodes, and may represent different types of relationships

Nodes and edges might be labeled,

and may have a set of properties represented as attributes (key-value pairs)***

Further assumptions:

- Edges are directed,
- Multi-graphs are allowed

**** Note: in some definitions (the least) edges are not allowed to have attributes*

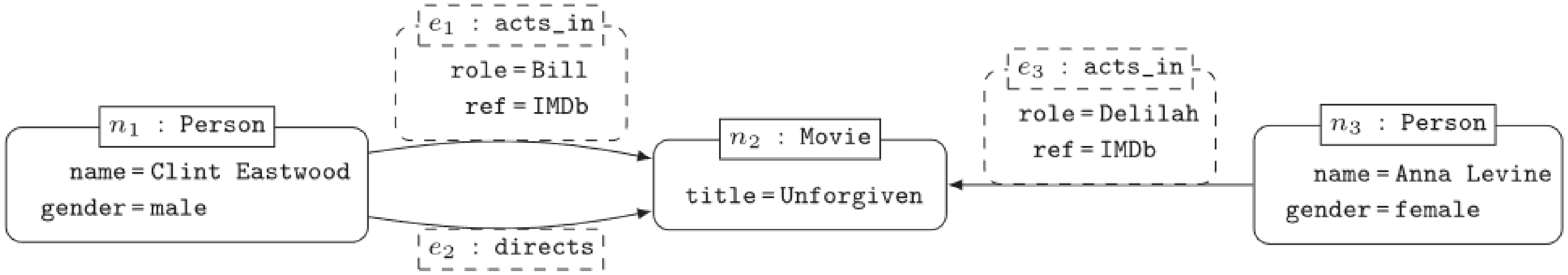
Formal Definition

Definition 2.3 (Property graph). A property graph G is a tuple $(V, E, \rho, \lambda, \sigma)$, where:

(1) V is a finite set of *vertices* (or *nodes*).
(2) E is a finite set of *edges* such that V and E have no elements in common.
(3) $\rho : E \rightarrow (V \times V)$ is a total function. Intuitively, $\rho(e) = (v_1, v_2)$ indicates that e is a directed edge *from* node v_1 *to* node v_2 in G.
(4) $\lambda : (V \cup E) \rightarrow Lab$ is a total function with Lab a set of labels. Intuitively, if $v \in V$ (respectively, $e \in E$) and $\rho(v) = \ell$ (respectively, $\rho(e) = \ell$) , then ℓ is the label of node v (respectively, edge e) in G.
(5) $\sigma : (V \cup E) \times Prop \rightarrow Val$ is a partial function with $Prop$ a finite set of properties and Val a set of values. Intuitively, if $v \in V$ (respectively, $e \in E$), $p \in Prop$ and $\sigma(v, p) = s$ (respectively, $\sigma(e, p) = s$), then s is the value of property p for node v (respectively, edge e) in the property graph G.

Extracted from: R. Angles et al. Foundations of Modern Query Languages for Graph Databases

Example of Property Graph



$$V = \{n_1, n_2, n_3\} \qquad E = \{e_1, e_2, e_3\}$$

$$\rho(e_3) = (n_3, n_2)$$

$$\lambda(n_3) = \texttt{Person} \qquad \lambda(e_1) = \texttt{acts\_in}$$

$$\lambda(e_2) = \texttt{directs} \qquad \lambda(e_3) = \texttt{acts\_in}$$

$$\begin{aligned}
\sigma(n_1, \texttt{gender}) &= \texttt{male}\\
\sigma(n_2, \texttt{title}) &= \texttt{Unforgiven}\\
\sigma(n_3, \texttt{name}) &= \texttt{Anna Levine}\\
\sigma(n_3, \texttt{gender}) &= \texttt{female}\\
\sigma(e_1, \texttt{role}) &= \texttt{Bill}\\
\sigma(e_1, \texttt{ref}) &= \texttt{IMDb}\\
\sigma(e_3, \texttt{role}) &= \texttt{Delilah}\\
\sigma(e_3, \texttt{ref}) &= \texttt{IMDb}
\end{aligned}$$

Traversal Navigation

We define the graph traversal pattern as: “the ability to rapidly traverse structures to an arbitrary depth (e.g., tree structures, cyclic structures) and with an arbitrary path description (e.g. friends that work together, roads below a certain congestion threshold)” [Marko Rodriguez]

Totally opposite to set theory (on which relational databases are based on)

- Sets of elements are operated by means of the relational algebra

Traversing Data in a RDBMS

In the relational theory, it is equivalent to joining data (schema level) and select data (based on a value)

User	Address	Phone	Email	Alternate
Alice	123 Foo St.	12345678	alice@example.org	alice@neo4j.org
Bob	456 Bar Ave		bob@example.org	
…	…	…	…	…
Zach	99 South St.		zach@example.org	

User	OrderId
Alice	1234
Alice	5678
Alice	9012
…	…
Zach	5588

OrderId	LineItemId
1234	abcd
1234	efab
…	…
5588	bafd

LineItemId	Description	Handling
abcd	strawberry ice cream	freezer
efab	potatoes	
…	…	
bafd	dried spaghetti	

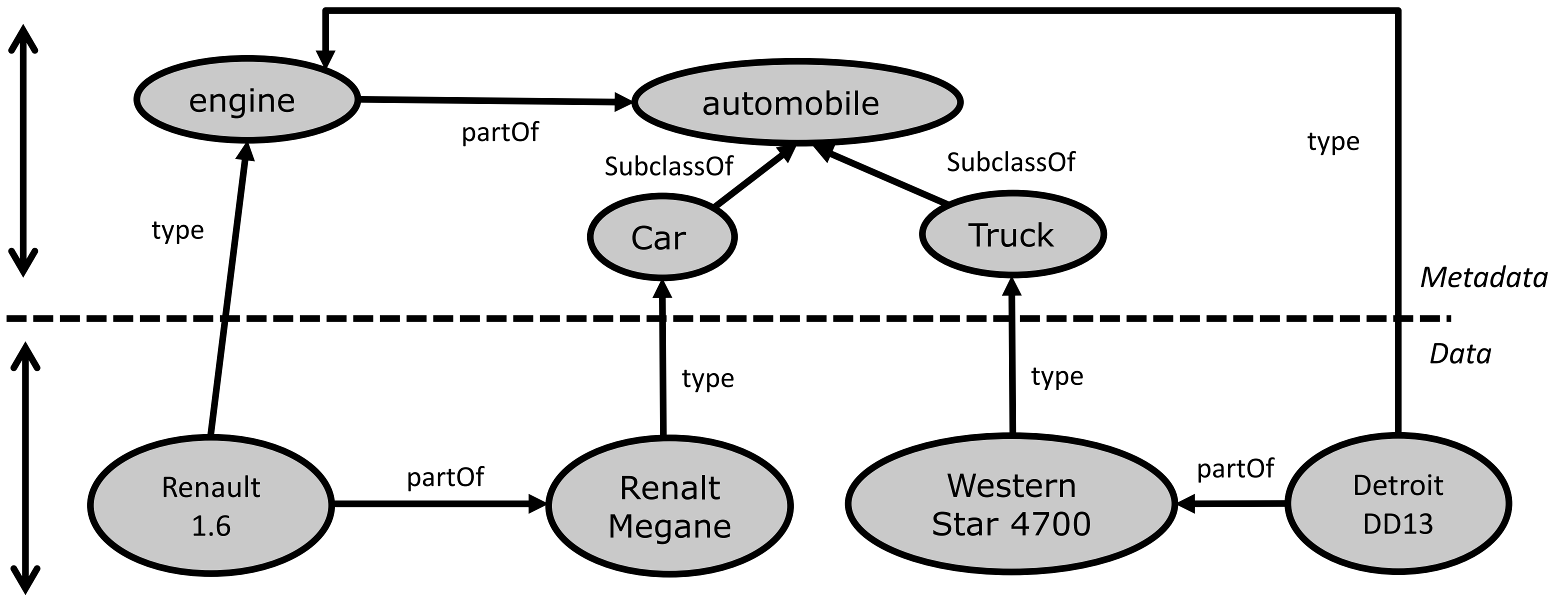
```
SELECT *
FROM user u, user_order uo,
orders o, items i
WHERE u.user = uo.user AND
uo.orderId = o.orderId AND
i.lineItemId = i.LineItemId
AND u.user = 'Alice'
```

Knowledge graphs

Every node is represented with a unique identifier and can be universally referred

Metadata is represented as nodes and edges (not using special constructs; e.g., labels)

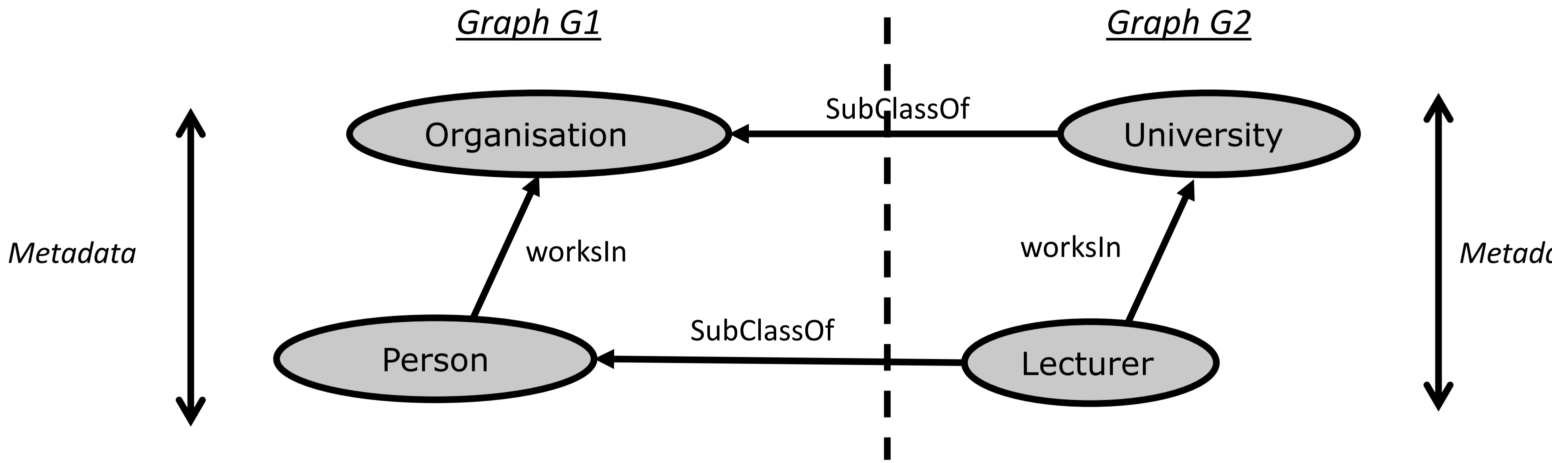
Predefined vocabularies embedding popular semantics



KGs as Canonical Data Model

Knowledge graphs facilitate linking data

- Linking via their metadata is way more powerful and it is a unique feature of their own



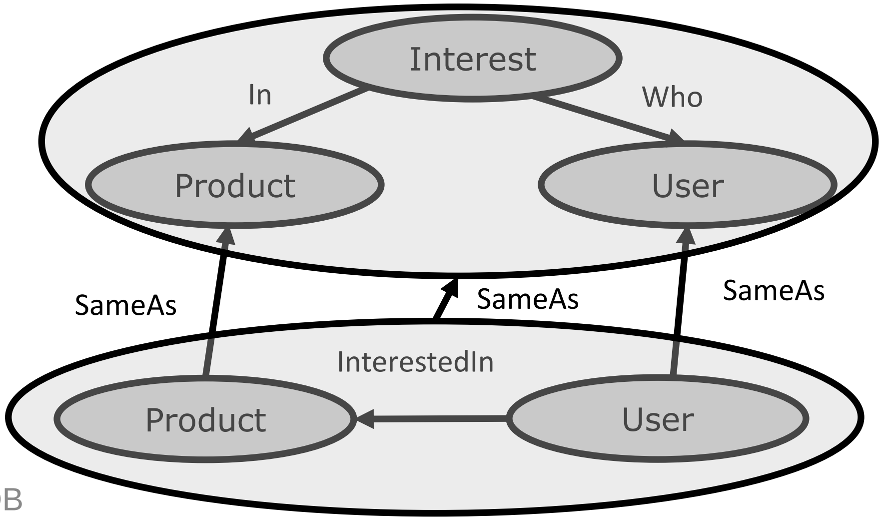
How to do this in a property graph?

KGs as Canonical Data Model

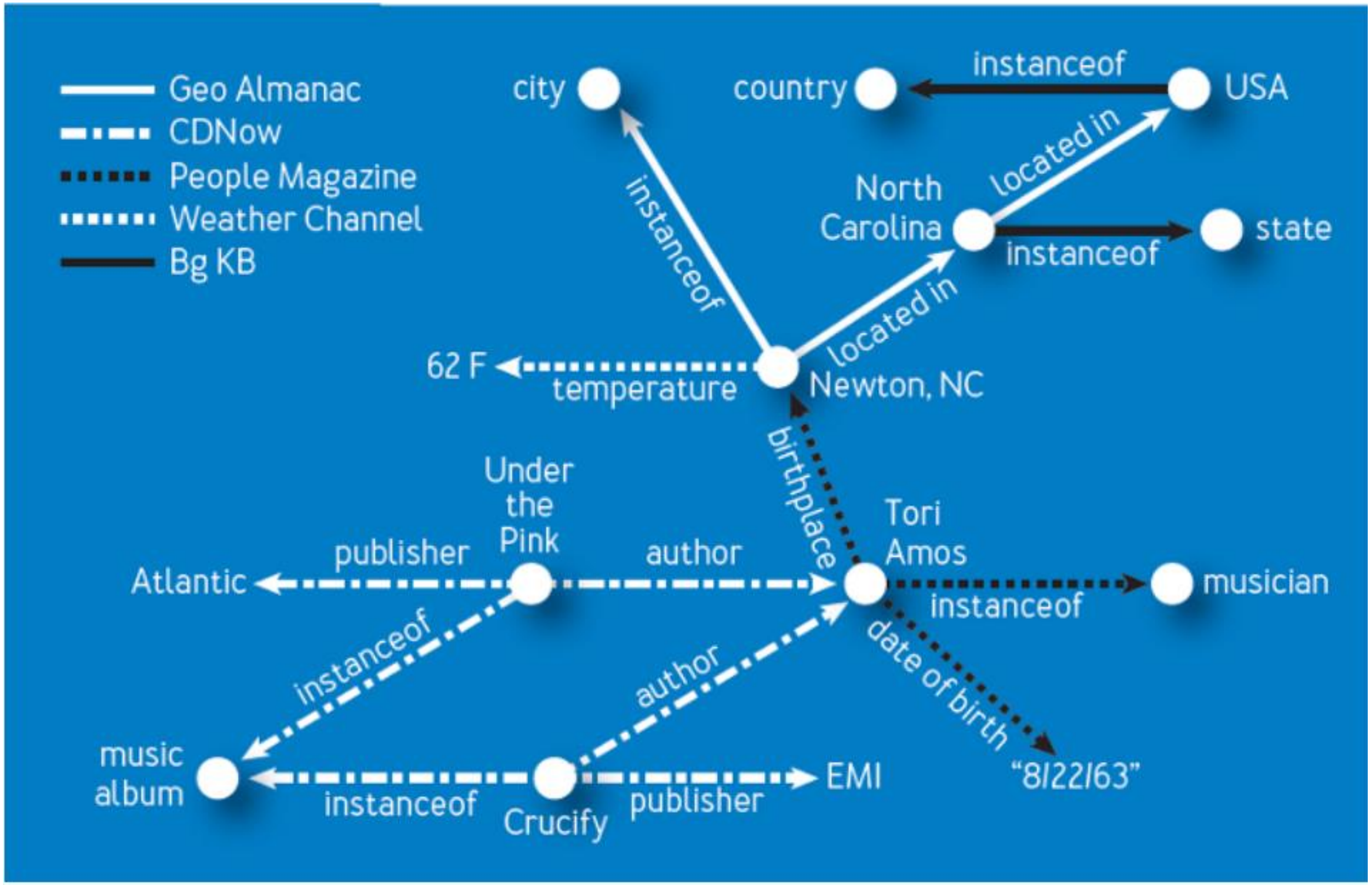
Knowledge graphs facilitate linking data

- Linking via their metadata is way more powerful and it is a unique feature of their own

But KGs are even more flexible than that…

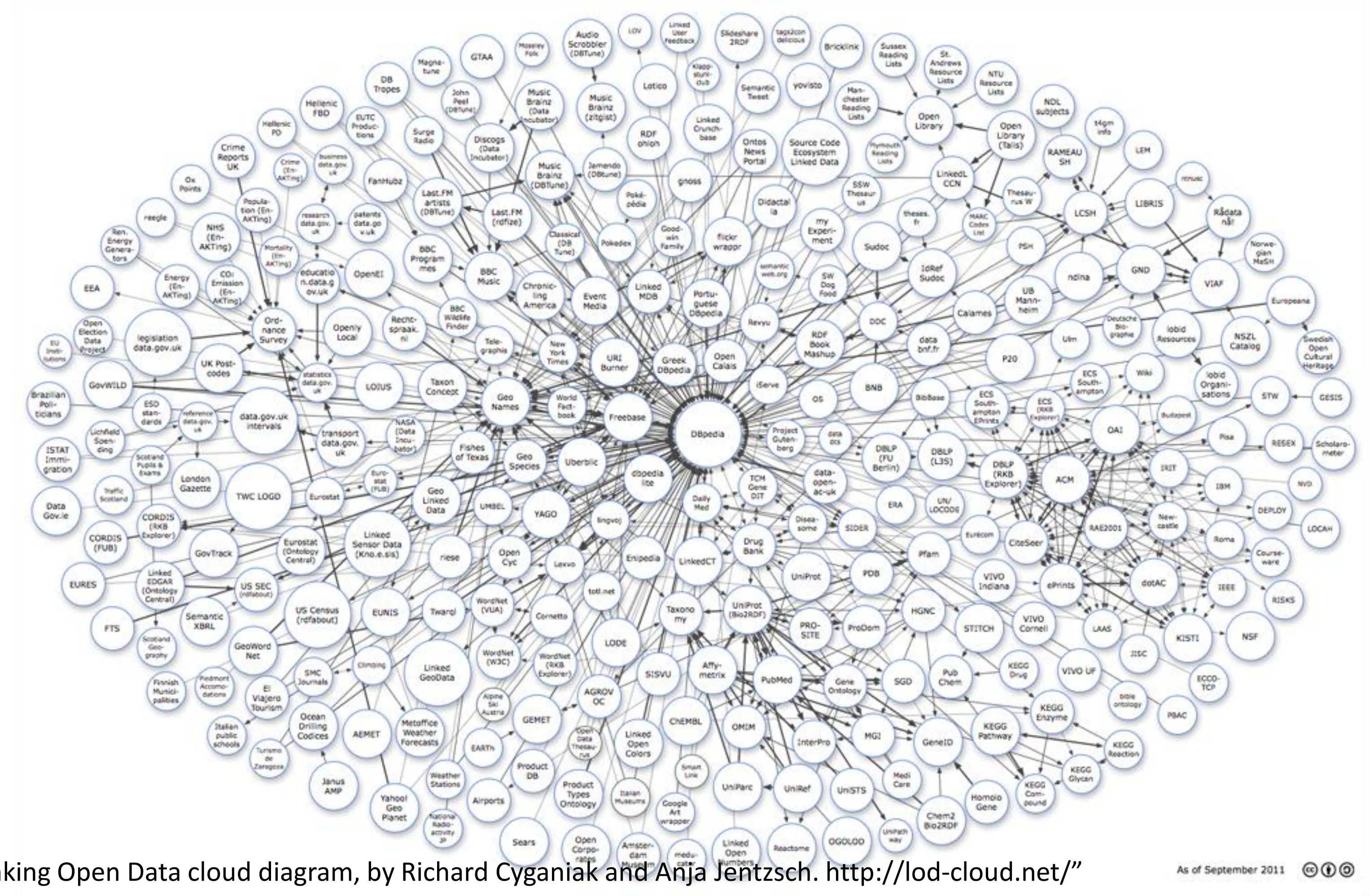


The Envisioned Idea: Distributed Knowledge



Extracted from: https://queue.acm.org/detail.cfm?id=2857276

The Linking Open Data Project



“Linking Open Data cloud diagram, by Richard Cyganiak and Anja Jentzsch. http://lod-cloud.net/”

Neo4j

Graph database

- https://neo4j.com/download/other-releases/

GUI di base

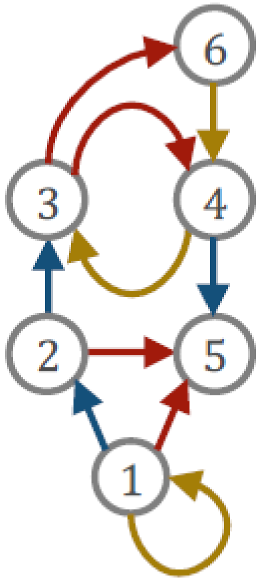
- Start docker and databases
- http://127.0.0.1:7474/
    - Database: neo4j
    - User: neo4j
    - Pwd: fitstic

Comes with tutorials to start using it

Graph Database - Concepts

There are three fundamental concepts in a graph database:

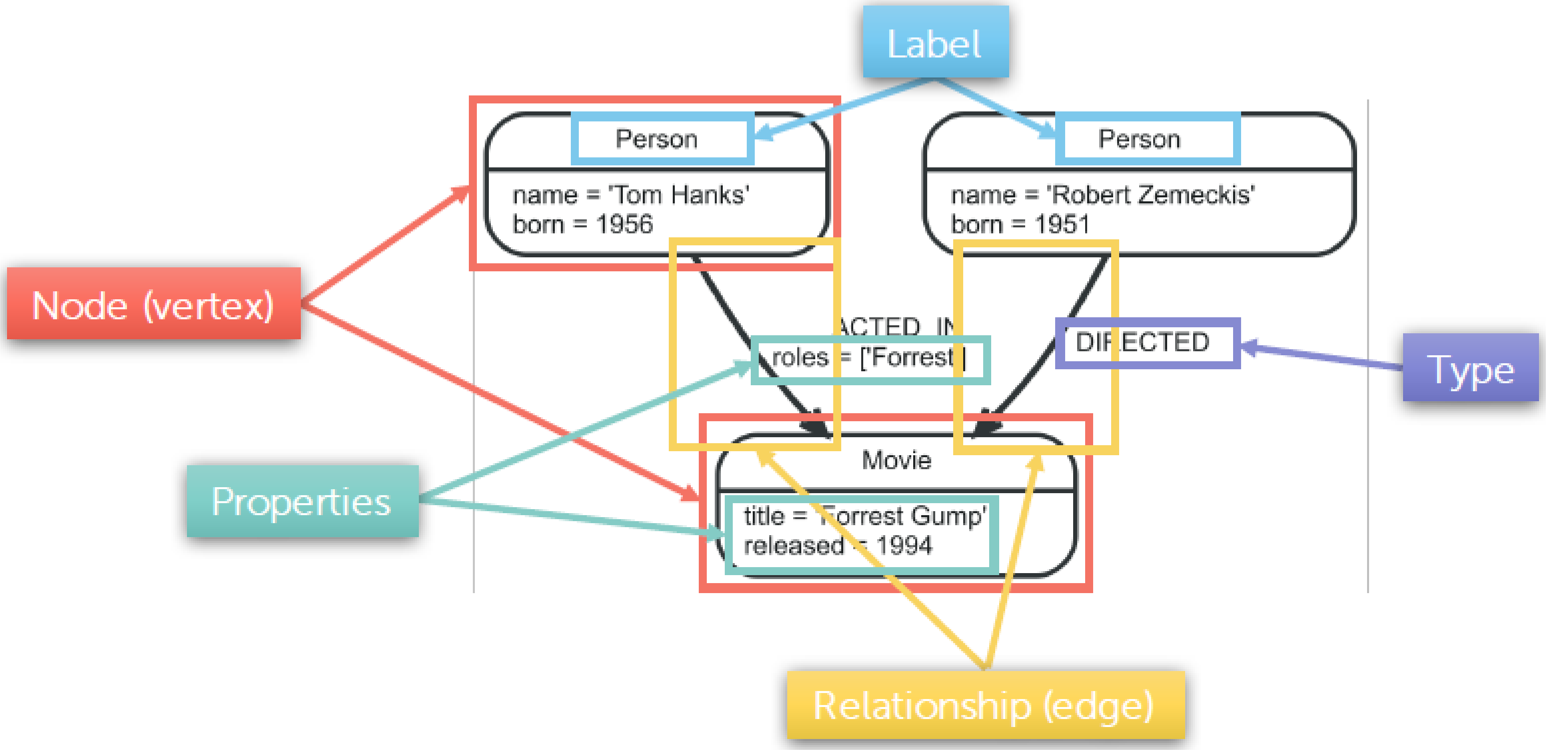
- Nodes: records, data units
- Relationships (or arcs): Directed links between nodes
- Properties: values (with a certain label) associated with a node or relation



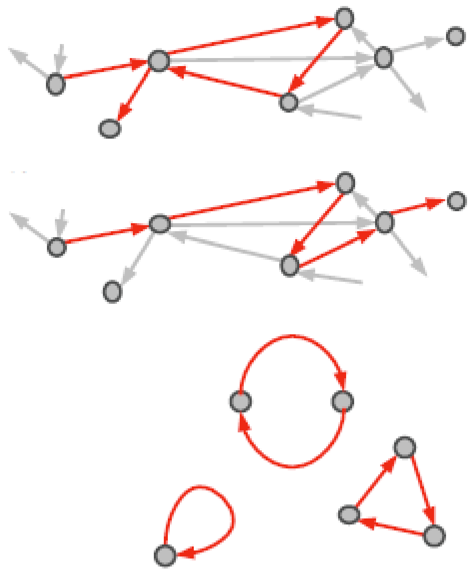
Relations are pointers contained in a node and pointing to another node

- Very different mechanism from foreign key in RDBMS
- Much more efficient for certain types of queries

Graph Database - Concepts



Graph Database - Concepts



Path

- Sequence of distinct arcs connecting two nodes

Way

- Path passing through distinct nodes

Cycle

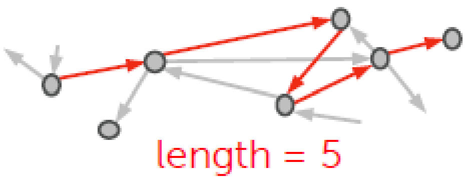
- Path that begins and ends in the same node

Distance between two nodes

- Minimum number of arcs connecting two nodes

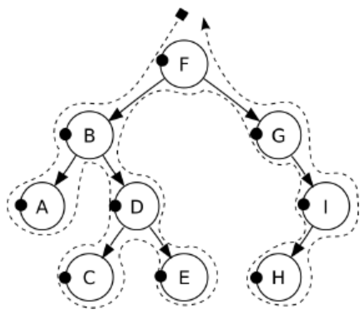
Graph Database - Concepts

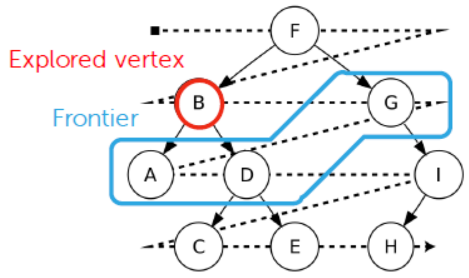
One of the most known queries is finding the shortest path between two nodes



Two main methods:

- Depth-first search
  - Examine all child nodes before examining sibling nodes
  - Requires fewer resources
  - Examine the whole graph to find the right solution
- Breadth-first search
  - Examine all sister nodes before examining child nodes
  - Requires more resources
  - The first solution he finds is the right one





Graph Database - Concepts

Betweenness centrality (A)

- Number of shorter paths between two nodes passing through a certain node

Closeness centrality (B)

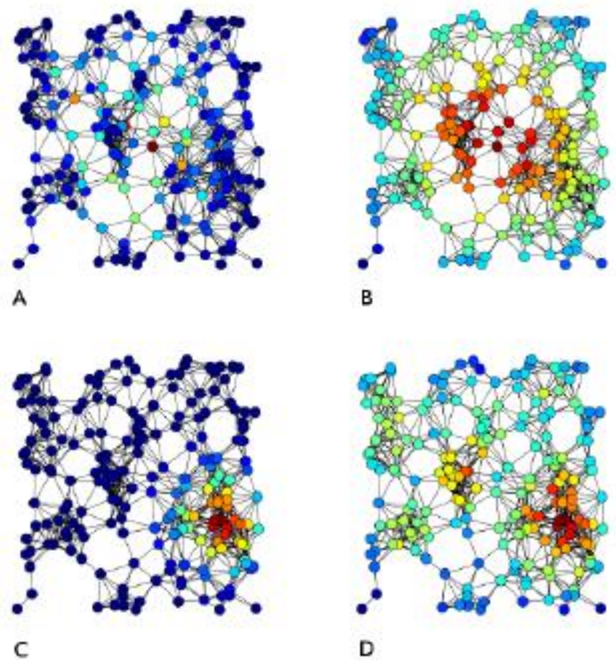
- Sum of distances from all other nodes

Eigenvector centrality (C)

- A node's score is influenced by Adjacent node scoring (page rank)

Degree centrality (D)

- Number of adjacent nodes



Query language: Cypher

Two main clauses: **match** and **return**

Match

- Which data to be retrieved by specifying patterns
- Multiple match per query
- Similar to the combination of WHERE and JOIN in SQL

Return

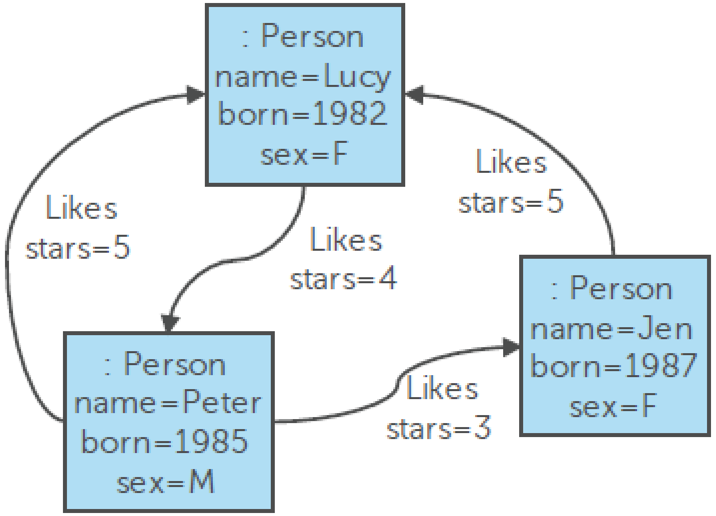
- Which data to be returned (nodes, arcs, properties, expressions)
- One clause per query
- Corresponds to SELECT in SQL

Cypher - Examples

Examples

- MATCH (p:Person)-[:Likes]->(f:Person)
  RETURN p.name, f.sex

p.name	f.sex
Lucy	M
Peter	F
Jen	F
Peter	F



- MATCH (p:Person)-[:Likes]->(:Person) -[:Likes]->(fof:Person)
  RETURN p.name, fof.name

p.name	fof.name
Lucy	Jen
Peter	Lucy
Peter	Peter
Jen	Peter
Lucy	Lucy

Cypher – Pattern syntax

Matching **nodes**

()	unidentified node
(matrix)	node identified by the matrix variable
(:Movie)	unidentified node of class “Movie”
(matrix:Movie:Action)	node with “Movie” and “Action” classes identified by matrix variable…
(matrix:Movie {title: "The Matrix"})	… with a title property equal to "The Matrix"
(matrix:Movie {title: "The Matrix", released: 1997})	… with a released property equal to 1997

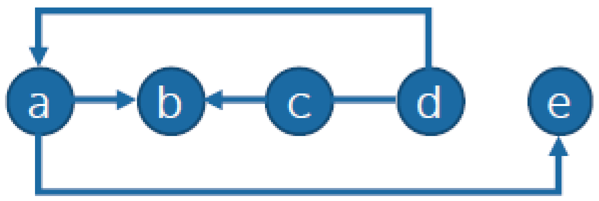
Matching **arcs**

-->	unidentified arc
--	unidentified arc without direction
-[role]->	arc identified by the role variable
-[:ACTED_IN]->	unidentified arc of class “ACTED_IN”
-[role:ACTED_IN]->	arc of class “ACTED_IN” identified by the variable role
-[role:ACTED_IN {roles: ["Neo"]}]->	… with roles properties that contains "Neo"

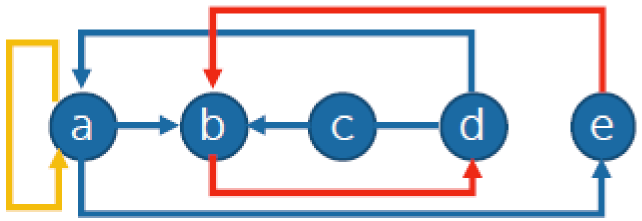
Cypher – Pattern syntax

Syntax for paths

- A path is a string in which nodes and arcs alternate
- A path always begins and ends with a node
- (a)-->(b)<--(c)--(d)-->(a)-->(e)
- (keanu:Person:Actor {name: "Keanu Reeves"})
  -[role:ACTED_IN {roles: ["Neo"]}]->
  (matrix:Movie {title: "The Matrix"})



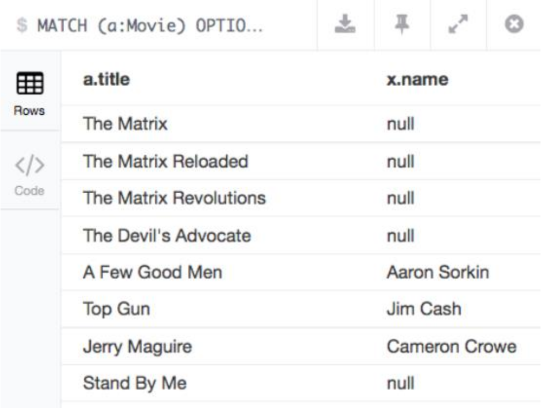
- Specify multiple paths, as long as they are connected by at least one shared variable
- (a)-->(b)<--(c)--(d)-->(a)-->(e),
  (e)-->(b)-->(d),
  (a)-->(a)



Match opzionale & Where

Optional Match clause

- Works as a left outer join
- If the pattern does not match, returns null
- MATCH (a:Movie)
  OPTIONAL MATCH (a)<-[:WROTE]-(x)
  RETURN a.title, x.name



$ MATCH (a:Movie) OPTIO...

a.title	x.name
The Matrix	null
The Matrix Reloaded	null
The Matrix Revolutions	null
The Devil's Advocate	null
A Few Good Men	Aaron Sorkin
Top Gun	Jim Cash
Jerry Maguire	Cameron Crowe
Stand By Me	null

Where clause

- Adds conditions that must be met by the pattern
- More expressive of the conditions that can be specified in Match
- MATCH (n)
  WHERE n.name = 'Matteo' XOR (n.age < 30 AND n.name = 'Enrico')
  OR NOT (n.name ~= 'Enr.*' OR n.name CONTAINS 'att')
  RETURN n

Variable-length paths

Follow the same type of arc by specifying how many "jumps" you want to do

- The * character precedes the length declaration
  - (a)-[:x*2]->(b) Exactly two jumps: (a)-[:x]->()-[:x]->(b)
  - (a)-[*3..5]->(b) Minimum 3, maximum 5
  - (a)-[*3..]->(b) Minimum 3
  - (a)-[*..5]->(b) Maximum 5
  - (a)-[*]->(b) No limits
- Un esempio completo
  - MATCH (me)-[:KNOWS*1..2]->(remote_friend)
    WHERE me.name = "Enrico" RETURN remote_friend.name
  - Returns direct friends and friends of friends
  - Attention: if a direct friend and also friend of friends, will be returned twice!

Percorsi di lunghezza variabile

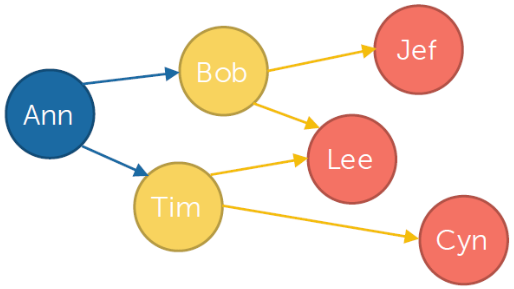
It is possible to search for the shortest path between two nodes

- ```
  MATCH (m { name:"Martin Sheen" }),
        (o { name:"Oliver Stone" }),
        p = shortestPath((m)-[*..15]-(o))
  RETURN p
  ```
  ```

Aggregation

The group-by clause is implicit

- Expressions in RETURN without aggregate functions are grouping keys
- Expressions in RETURN with aggregate functions produce aggregates
- MATCH (me:Person {name:'Ann'})-->(friend:Person)-->(friend_of_friend:Person) RETURN me.name, count(DISTINCT friend_of_friend), count(friend_of_friend)



Result

me	COUNT DISTINCT	COUNT
Ann	3	4