Property Graphs

ANNA QUERALT, OSCAR ROMERO

(FACULTAT D'INFORMÀTICA DE BARCELONA)

Data Model

The Property Graph Data Model

Born in the database community
- Meant to be queried and processed
- **THERE IS NOTHING SUCH AS A STANDARD!**

Two main constructs: nodes and edges
- Nodes represent entities
- Edges relate pairs of nodes, and may represent different types of relationships

Both nodes and edges:
- May be labeled
- May have a set of properties represented as attributes (key-value pairs)**

Further considerations:
- Edges are directed
- Multi-graphs are allowed

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

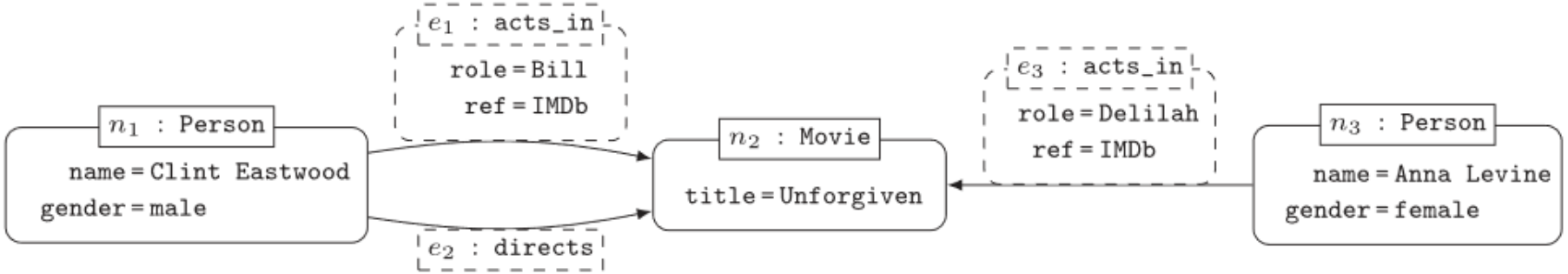
Formal Definition

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) $\rho : E \to (V \times V)$ 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) $\lambda : (V \cup E) \to Lab$ is a total function with Lab a set of labels. Intuitively, if $v \in V$ (respectively, $e \in E$) and $\lambda(v) = \ell$ (respectively, $\lambda(e) = \ell$) , then ℓ is the label of node v (respectively, edge e) in G.
(5) $\sigma : (V \cup E) \times Prop \to Val$ 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



Formal definition:

$V =$ $E =$

ρ σ

λ

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]

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

Obtaining all the items that Alice has ordered

User	Address	Phone	Email
Alice			
Bob	456 Bar Ave.		bob@example.org
…	…	…	…
Zach	99 South St.		zach@example.org

Order	Date	Status	UserId
1234	20120808	delivered	Alice
5678	20120816	dispatched	Alice
…	…	…	…
5588	20120613	delivered	Zach

Id	Description	Handling
abcd	Strawberry ice-cream	freezer
efab	Brussels sprouts	
cdef	Espresso beans	
…	…	…

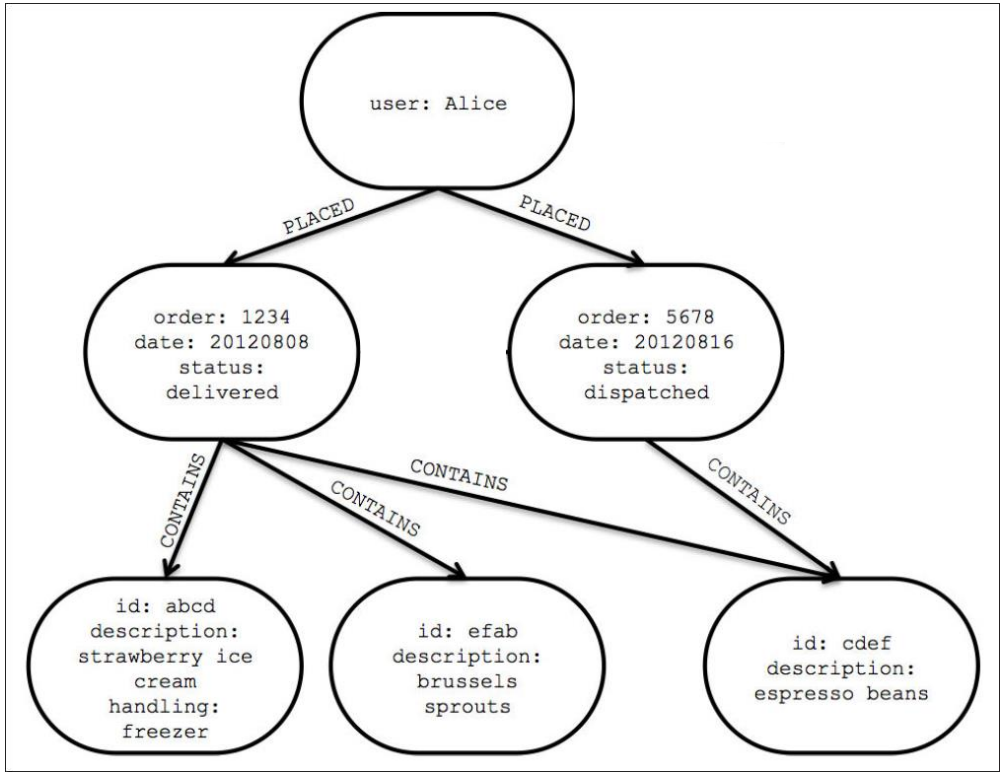
OrderId	ItemId
1234	abcd
1234	efab
1234	cdef
5678	cdef
…	
5588	hijk

```
SELECT i.id, i.description
FROM users u, orders o,
  order_items oi, items i
WHERE u.user = 'Alice' AND
  u.user = o.userId AND
  o.order = oi.orderId AND
  oi.itemId = i.id
```

Cardinalities:

|Users|: 5.000.000
|Orders|: 1.000.000.000
|OrderItems|: 3.000.000.000
|Items|: 35.000

Traversing Data in a Graph Database

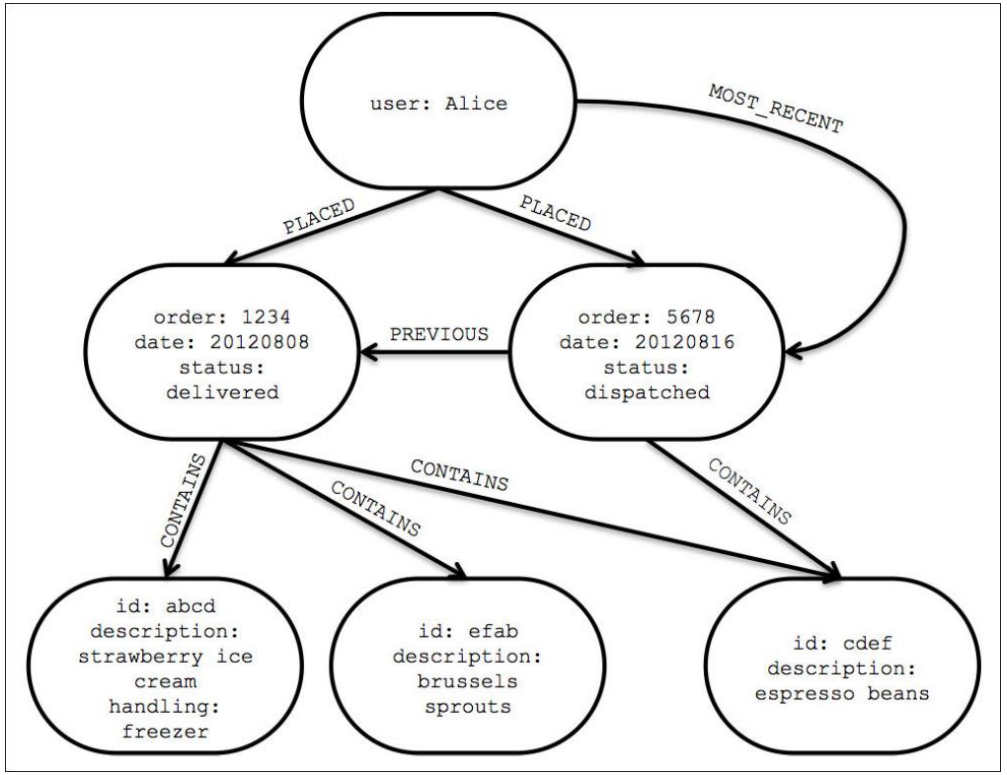


- Relationships are explicit
  - No FKs
  - No need to add nodes for "artificial" concepts
  - JOINs are "hard-wired"
- Traversing from one node to another is a constant time operation

Activity

What would be the cost of the same query in a graph database?

- *Assume you can find the node "Alice" in constant time*



Cardinalities:

|Users|: 5.000.000
|Orders|: 1.000.000.000
|Items|: 35.000

Graph DBs vs Relational DBs

The following table reports result of an experiment aimed to finding friends-of-friends in a social network, to a maximum depth of 5

- For a social network containing 1,000,000 people, each with approximately 50 friends.

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

From *Neo4j in Action. Jonas Partner, Aleksa Vukotic, and Nicki Watt. MEAP. 2012*

Traversing Property Graphs

Traversing graph data depends on three main variables

- The **query** topology
  - and the traversal “**seeds**"
- The **size** of the graph (i.e., #edges)
- The **topology** of the graph

Graph Operations

Graph Operations

Content-based queries

- The value is relevant
  - Get a node, get the value of a node / edge attribute, etc.
  - A typical case are summarization queries (i.e., aggregations)

Topological queries

- Only the graph topology is considered
- Typically, several business problems (such as fraud detection, trend prediction, product recommendation, network routing, or route optimization) are solved using graph algorithms exploring the graph topology
  - Computing the betweenness centrality of a node…
    - In a social network, an analyst can detect influential people or groups for targeting a marketing campaign audience.
    - In a telecommunication operator, an analyst may detect central nodes of an antenna network and optimize the routing and load balancing across the infrastructure accordingly

Hybrid approaches

Topological Queries

Divided into three (four) main categories

- Adjacency
- Reachability
- Pattern Matching
- [Graph metrics]

Adjacency Queries

Formalized as node adjacency or edge incidence

- Node adjacency
- Edge incidence (node degree, out-degree, in-degree)
- K-neighbourhood of a node

Formal definition: $\text{Adjacency}(n) = \bar{N}$

$$n_i \in \bar{N} \iff \exists e_1 \mid \rho(e_1) = (n_i, n) \vee \rho(e_1) = (n, n_i)$$

Computational cost: linear cost on the number of edges to visit

Examples:

- Find all friends of a person
- Airports with a direct connection
- Movies watched by a person
- Products bought by a customer
- …

Reachability Queries

Formal definition:

$$\text{Reachability}(n_{or},\, n_{dest}) \text{ is } \mathbf{true} \iff \exists \text{Walk}(n_{or},\, n_{dest})$$
$$\text{Walk}(n_{or}, n_{dest}) = (e_1 \dots e_m) \mid \exists n_1 \dots n_{m-1},\, \rho(e1) = (n_{or}, n_1),\, \rho(e2) = (n_1, n_2)$$
$$\dots\ \rho(e_m) = (n_{m-1}, n_{dest})$$

Reachability queries **find a walk** between two nodes. They may define additional constraints (e.g., shortest path, fixed-length paths, etc.)

- Fixed-length paths (fixed #edges and #nodes)
- Shortest path
- Non-repeated nodes (path)
- Non-repeated edges (trail)
- Regular simple paths (restrictions as regular expressions)
  - Hybrid if the restriction is in the *content*

Reachability Queries

Computational cost: hard to compute for large graphs

- Shortest-path (Dijkstra's algorithm): $\mathbf{O(|V^2|)}$
  - Smarter implementations based on priority queues yield $\mathbf{O(|E|*|V|log|V|)}$ complexity

Examples:

- Friend-of-a-friend
- Flight connections
- Logistics (goods distribution)
- …

Label-constrained Reachability

Definition:

$$G_L^* = \{(s,t) \mid \text{there is a path in } G \text{ from } s \text{ to } t \text{ using only edges with labels in } L\}$$

It is equivalent to determine whether or not there is a path in **G** from *s* to *t* such that the concatenation of the edge labels along the path forms a string in the language denoted by the regular expression

$$(\ell_1 \cup \cdots \cup \ell_n)^*$$

where: $L = \{\ell_1, \ldots, \ell_n\}$, U is *disjunction and* * is *the Kleene star*

Typically, the allowed topology and labels involved are expressed as a regular expression

- In general, the cost of regular label-constrained queries (computing regular simple paths) is known to be NP-complete

Pattern Matching

Formalized as the graph isomorphism problem

- Input: property graph **G**, and a pattern graph **P**
- Output: all sub-graphs of **G** that are isomorphic to **P**

Computational cost: hard to compute, in general, NP-complete

Examples:

- Group of cities all of them directly connected by flights
- People who have ordered the same item
- ...

Property Graph Patterns

Based on *basic graph patterns* (bgps)

- Equivalent to conjunctive queries

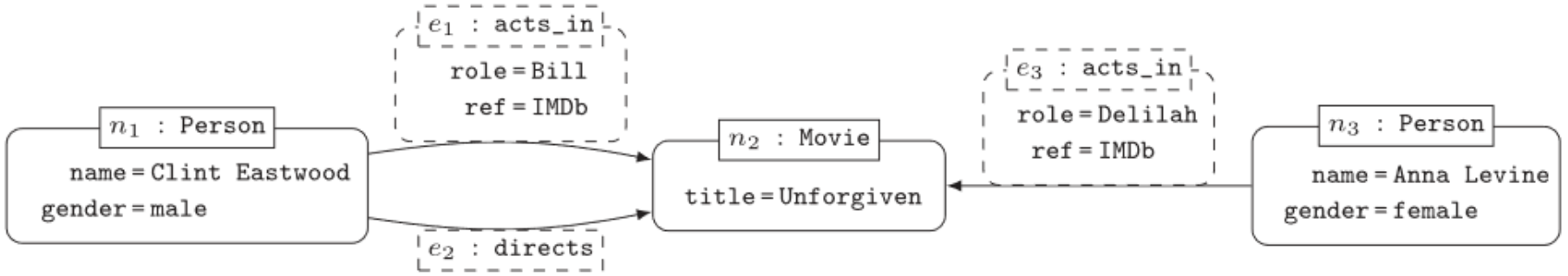
A *bgp* for querying property graphs is a property graph where variables can appear in place of any constant (ids/labels/properties)

A **match** for a *bgp* is a mapping from variables to constants such that when the mapping is applied to the *bgp*, the result is *contained* within the original graph

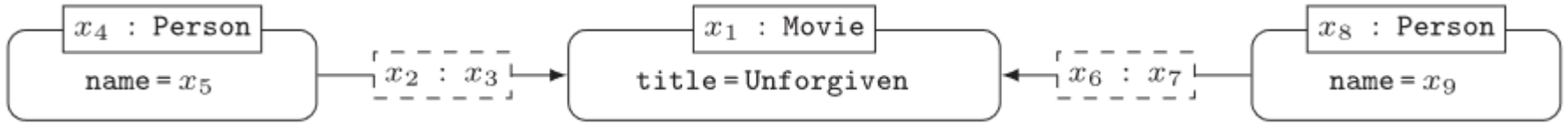
The **results** for a *bgp* are then **all mappings** from variables in the query to constants that comprise a match

Example of Graph Pattern

Graph:



BGP:



Evaluating Graph Patterns

Evaluating a *bgp* **Q** against a graph database **G** corresponds to listing **all** possible matches of **Q** with respect to **G**

Formally:

> *Definition 3.5 (Match).* Given an edge-labelled graph $G = (V, E)$ and a bgp $Q = (V', E')$, a *match* h of Q in G is a mapping from $Const \cup Var$ to $Const$ such that:
>
> (1) for each constant $a \in Const$, it is the case that $h(a) = a$; that is, the mapping maps constants to themselves; and
> (2) for each edge $(b, l, c) \in E'$, it holds that $(h(b), h(l), h(c)) \in E$; this condition imposes that (a) each edge of Q is mapped to an edge of G, and (b) the structure of Q is preserved in its image under h in G (that is, when h is applied to all the terms in Q, the result is a sub-graph of G).

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

Semantics of a Match

Homomorphism-based semantics: the previous definition maps to a homomorphism from **Q** to **G**

- Multiple variables in **Q** can map to the same term in **G** (within a mapping)

Isomorphism-based semantics: an additional constraint is added, the match function *h* must be injective (within a mapping)

Still, different semantics can be applied:

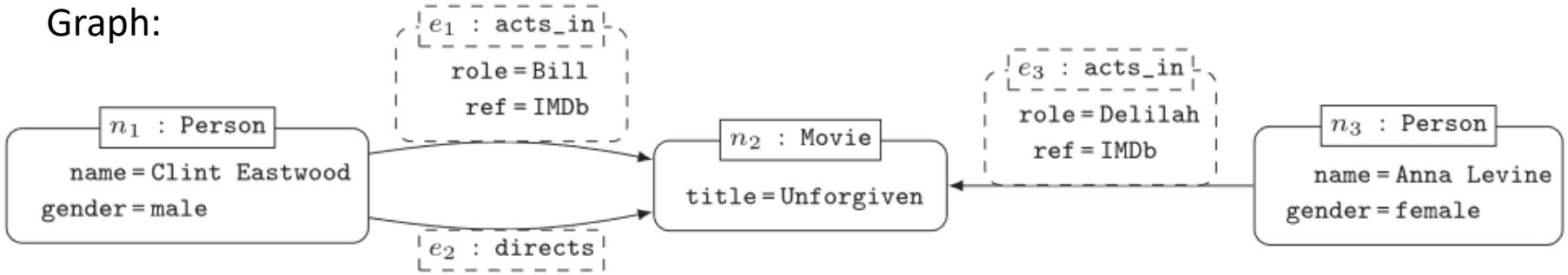
- Strict isomorphism (no-repeated-anything): *h* is injective
- No repeated-node semantics: *h* is only injective for nodes
- No repeated-edge semantics: *h* is only injective for edges

Activity

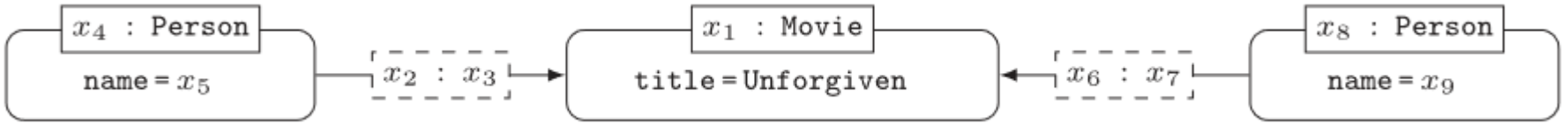
Objective: Understand the differences between graph matching isomorphism-based and homomorphism-based semantics

- *Given the following graph, bgp and potential results…*

Graph:



BGP:



Activity

Objective: Understand the differences between graph matching isomorphism-based and homomorphism-based semantics

- *Given the following graph, bgp and potential results…*

Results:

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9
1	n_2	e_2	directs	n_1	Clint Eastwood	e_3	acts_in	n_3	Anna Levine
2	n_2	e_3	acts_in	n_3	Anna Levine	e_2	directs	n_1	Clint Eastwood
3	n_2	e_1	acts_in	n_1	Clint Eastwood	e_3	acts_in	n_3	Anna Levine
4	n_2	e_3	acts_in	n_3	Anna Levine	e_1	acts_in	n_1	Clint Eastwood
5	n_2	e_2	directs	n_1	Clint Eastwood	e_1	acts_in	n_1	Clint Eastwood
6	n_2	e_1	acts_in	n_1	Clint Eastwood	e_2	directs	n_1	Clint Eastwood
7	n_2	e_1	acts_in	n_1	Clint Eastwood	e_1	acts_in	n_1	Clint Eastwood
8	n_2	e_2	directs	n_1	Clint Eastwood	e_2	directs	n_1	Clint Eastwood
9	n_2	e_3	acts_in	n_1	Anna Levine	e_3	acts_in	n_1	Anna Levine

Which results would be obtained when applying **isomorphism-based** semantics? And **homomorphism-based** semantics?

- For isomorphism, **distinguish the three isomorphism semantics** presented

Summary

Property graphs do not have a defined standard, but there is a de-facto standard:

- Directed
- Multigraph
- Nodes/edges may have labels (similar to the concept of typing) and properties (equivalent to the concept of attributes)

The basic operations on graphs are:

- Adjacency
- Reachability
- Pattern matching

The result of a Pattern Matching query depends on the semantics assumed:

- Homomorphism
- Strict Isomorphism
- No-repeated-node Isomorphism
- No-repeated-edge Isomorphism

Thanks! Any Question?