Querying Graphs

TIW2
Interoperability 2021-2022

Prof. Angela Bonifati

Lyon 1 University

8 November 2021

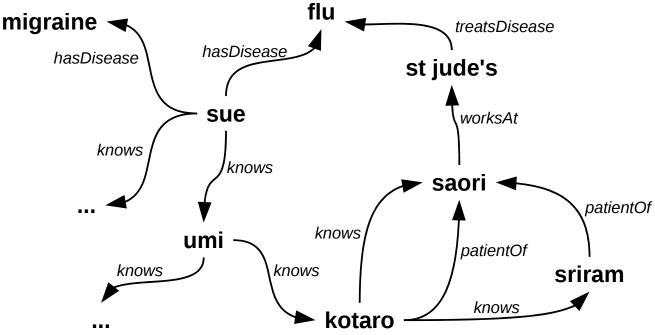
Agenda

1. Querying over graph data
   - query languages for graphs
   - openCypher

Graph queries

A small graph

Let's consider a graph with edge labels: knows, worksAt, patientOf, hasDisease, and treatsDisease.



Query language capabilities

Graph query languages typically feature one or both of the following basic capabilities

- subgraph matching
- finding nodes connected by paths

and possibly additional advanced features such as approximate matching, aggregation, and comparing paths.

Subgraph matching

Subgraph matching is the core basic capability of most graph query languages

Subgraph matching

Subgraph matching is the core basic capability of most graph query languages

Essentially, this consists of conjunctive queries on graphs

- an edge pattern is a triple (s, ℓ, t) where s and t can be either constants (in node set N) or variables, and $\ell \in \mathcal{L}$ is an edge label

Subgraph matching

Subgraph matching is the core basic capability of most graph query languages

Essentially, this consists of conjunctive queries on graphs

- an edge pattern is a triple (s, ℓ, t) where s and t can be either constants (in node set N) or variables, and $\ell \in \mathcal{L}$ is an edge label
- a query rule is then a pattern

$$head \quad \leftarrow \quad body$$

  where *head* and *body* are sets of edge patterns such that every variable occurring in *head* occurs in *body*

Subgraph matching

Subgraph matching is the core basic capability of most graph query languages

Essentially, this consists of conjunctive queries on graphs

- an edge pattern is a triple (s, ℓ, t) where s and t can be either constants (in node set N) or variables, and $\ell \in \mathcal{L}$ is an edge label
- a query rule is then a pattern

$$\textit{head} \quad \leftarrow \quad \textit{body}$$

  where *head* and *body* are sets of edge patterns such that every variable occurring in *head* occurs in *body*
  - alternatively, *head* is a list of zero or more of the variables (possibly with repetition) appearing in *body*

Subgraph matching

Subgraph matching is the core basic capability of most graph query languages

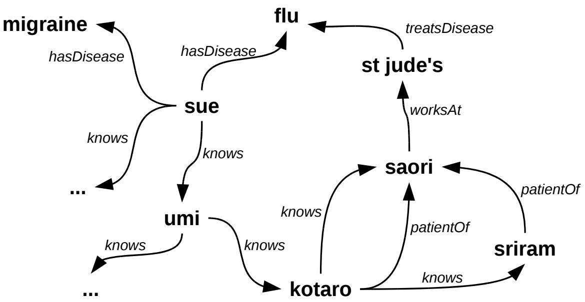
Essentially, this consists of conjunctive queries on graphs

- an edge pattern is a triple (s, ℓ, t) where s and t can be either constants (in node set N) or variables, and $\ell \in \mathcal{L}$ is an edge label
- a query rule is then a pattern

$$head \quad \leftarrow \quad body$$

  where *head* and *body* are sets of edge patterns such that every variable occurring in *head* occurs in *body*
  - alternatively, *head* is a list of zero or more of the variables (possibly with repetition) appearing in *body*
- A query is then a finite set of rules (of the same arity).

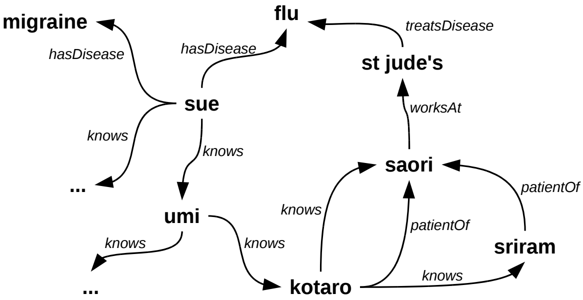
Subgraph matching



Example: People and the doctors of their friends

$$Q \quad = \quad (?p, \mathit{friendDoctor}, ?d) \leftarrow (?p, \mathit{knows}, ?f), (?f, \mathit{patientOf}, ?d)$$

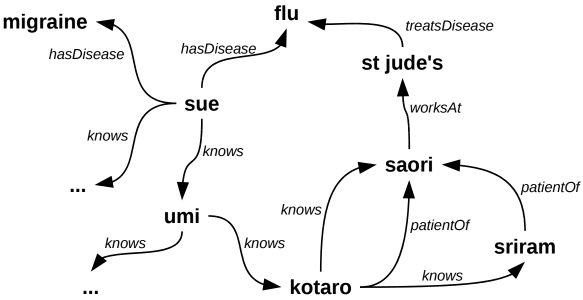
Subgraph matching



Example: People who know someone who knows a doctor.

$$Q \quad = \quad \langle ?p \rangle \leftarrow (?p, knows, ?f), (?f, knows, ?d), (?po, patientOf, ?d)$$

Subgraph matching



Example: Patients and their friends

$$Q \quad = \quad \langle ?p, ?f\rangle \leftarrow (?p, \mathit{knows}, ?f), (?p, \mathit{patientOf}, ?d)$$

Subgraph matching

The semantics $Q(G)$ of evaluating query a Q on graph G is based on embeddings of the rule *body*'s of Q in G:

$$Q(G) \;=\; \bigcup_{head \leftarrow body \in Q} \{h(head) \mid h(body) \subseteq G\}$$

where h is a homomorphism, i.e., a function with domain $N \cup \mathit{Variables}$ and range N that is the identity on N.

Subgraph matching

The semantics $Q(G)$ of evaluating query a Q on graph G is based on embeddings of the rule *body*'s of Q in G:

$$Q(G) \;=\; \bigcup_{head \leftarrow body \in Q} \{h(head) \mid h(body) \subseteq G\}$$

where h is a homomorphism, i.e., a function with domain $N \cup \mathit{Variables}$ and range N that is the identity on N.

Alternatively, some graph DBs adopt a stricter isomorphism semantics, i.e., homomorphisms that are injective.

- In other words, distinct variables in *body* must map to distinct nodes in G.

Subgraph matching

The semantics $Q(G)$ of evaluating query a Q on graph G is based on embeddings of the rule *body*'s of Q in G:

$$Q(G) \;=\; \bigcup_{head \leftarrow body \in Q} \{h(head) \mid h(body) \subseteq G\}$$

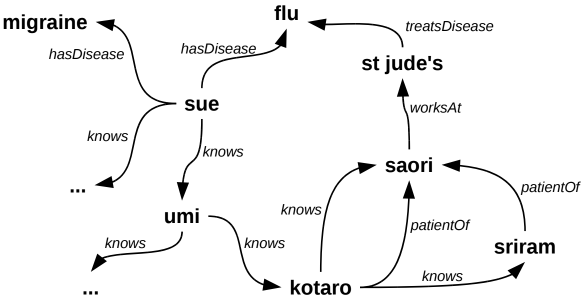
where h is a homomorphism, i.e., a function with domain $N \cup \mathit{Variables}$ and range N that is the identity on N.

Alternatively, some graph DBs adopt a stricter isomorphism semantics, i.e., homomorphisms that are injective.

- In other words, distinct variables in *body* must map to distinct nodes in G.

In the property graph model, a distinction is also sometimes made between node-isomorphism (i.e., our notion here) and edge-isomorphism (see Angles et al. 2016).

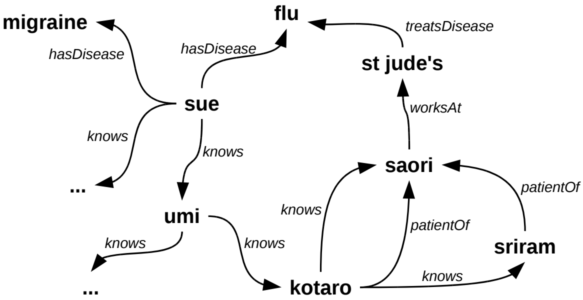
Subgraph matching



Example: People and the doctors of their friends

$$\begin{aligned} Q &= (?p, \mathit{friendDoctor}, ?d) \leftarrow (?p, \mathit{knows}, ?f), (?f, \mathit{patientOf}, ?d) \\ Q(G) &= \{(\mathit{umi}, \mathit{friendDoctor}, \mathit{saori}), (\mathit{kotaro}, \mathit{friendDoctor}, \mathit{saori}), \ldots\} \end{aligned}$$

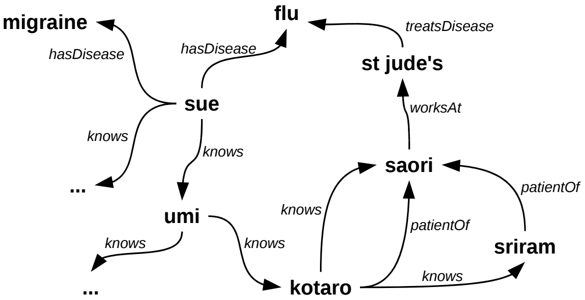
Subgraph matching



Example: People who know someone who knows a doctor.

$$Q = \langle ?p \rangle \leftarrow (?p, \mathit{knows}, ?f), (?f, \mathit{knows}, ?d), (?po, \mathit{patientOf}, ?d)$$
$$Q(G) = \{\langle \mathit{umi} \rangle, \ldots\}$$

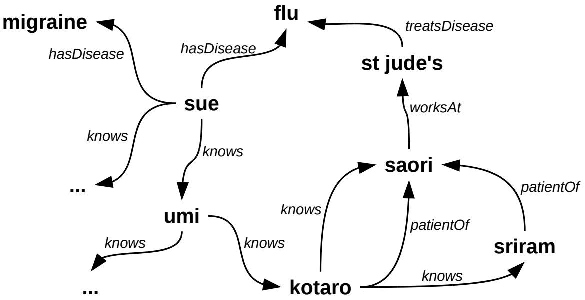
Subgraph matching



Example: Patients and their friends (homomorphisms)

$$Q \;=\; \langle ?p, ?f \rangle \leftarrow (?p, \mathit{knows}, ?f), (?p, \mathit{patientOf}, ?d)$$
$$Q(G) \;=\; \{\langle \mathit{kotaro}, \mathit{saori} \rangle, \langle \mathit{kotaro}, \mathit{sriram} \rangle, \ldots\}$$

Subgraph matching



Example: Patients and their friends (isomorphisms)

$$Q = \langle ?p, ?f \rangle \leftarrow (?p, knows, ?f), (?p, patientOf, ?d)$$
$$Q(G) = \{\cancel{\langle kotaro, saori \rangle}, \langle kotaro, sriram \rangle, \ldots\}$$

Subgraph matching

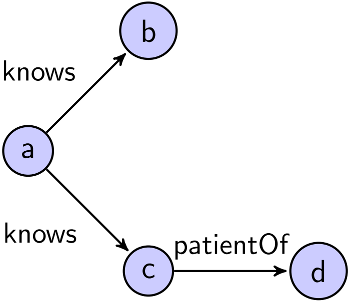
Evaluation of subgraph matching queries is NP-complete in combined complexity (i.e., in the size of Q and G) and logspace in data complexity (i.e., for a fixed query, in the size of G). This follows from the intractability of the subgraph homomorphism problem.

Subgraph matching: simulations

That is, instead of homomorphisms embedding Q in G, we look for a binary relation S between the nodes and variables of (a given) body of Q and the nodes of G such that

1. for each constant n in the body of Q, n is a node of G and $(n, n) \in S$;
2. for each variable v in the body of Q there exists a node n of G such that $(v, n) \in S$; and,
3. for each $(x, n) \in S$ and each edge pattern $(x, \ell, x') \in Q$, there is an edge $n \xrightarrow{\ell} n' \in G$ such that $(x', n') \in S$.

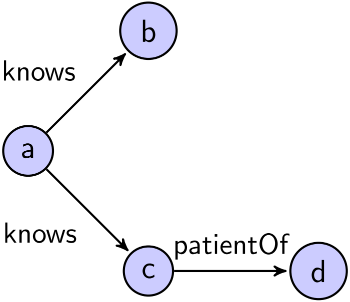
Subgraph matching: simulations



Example. The following boolean query is simulated in the graph above, but evaluates to "false" under standard query semantics

$$\langle\rangle \leftarrow (?x, \mathit{knows}, ?y), (?x, \mathit{knows}, ?z), (?z, \mathit{patientOf}, ?y)$$

Subgraph matching: simulations



Example. The following boolean query is simulated in the graph above, but evaluates to "false" under standard query semantics

$$\langle\rangle \leftarrow (?x, \mathit{knows}, ?y), (?x, \mathit{knows}, ?z), (?z, \mathit{patientOf}, ?y)$$

Here a simulation is $S = \{(?x, a), (?z, c), (?y, b), (?y, d)\}$

Path navigation: reachability

The simplest form of path matching is reachability, namely, computing

$$G^* \;=\; \{(s,t) \mid \text{there is a path in } G \text{ from } s \text{ to } t\}$$

or, given $x, y \in N$, determining whether or not $(x, y) \in G^*$.

Extensively studied in the DB community since the 80's (see the survey of Yu et al.).

Path navigation: label-constrained reachability

Generalizing reachability, we have the label-constrained reachability queries: given $x, y \in N$ and a set of labels $L \subseteq \mathcal{L}$, determining whether or not (x, y) is in the set

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

Path navigation: label-constrained reachability

Generalizing reachability, we have the label-constrained reachability queries: given $x, y \in N$ and a set of labels $L \subseteq \mathcal{L}$, determining whether or not (x, y) is in the set

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

Note that this is equivalent to the following problem

- determine whether or not there is a path in G from x to y 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 $L = \{\ell_1, \ldots, \ell_n\}$, $\cup$ is disjunction, and $*$ is the Kleene star ...

Path navigation: regular path queries

Regular path queries return all paths (i.e., pairs of nodes) connected by some regular expression over edge labels

- queries of the form

$$\langle ?x, ?y \rangle \leftarrow (?x, r, ?y)$$

  where r is a regular expression over $\mathcal{L}$
- semantics is the set of all node pairs (s, t) such that there is a path from s to t in G and the sequence of edge labels along the path forms a word in the language of r.
- query evaluation: $\mathcal{O}(|G||r|)$ time complexity

Path navigation: regular path queries

Regular path queries return all paths (i.e., pairs of nodes) connected by some regular expression over edge labels

- queries of the form

$$\langle ?x, ?y \rangle \leftarrow (?x, r, ?y)$$

  where r is a regular expression over $\mathcal{L}$
- semantics is the set of all node pairs (s, t) such that there is a path from s to t in G and the sequence of edge labels along the path forms a word in the language of r.
- query evaluation: $\mathcal{O}(|G||r|)$ time complexity

For example, the "knowing" social network is

$$\langle ?x, ?y \rangle \leftarrow (?x, \text{knows}^+, ?y)$$

and the general social network is

$$\langle ?x, ?y \rangle \leftarrow (?x, (\text{knows} \cup \text{patientOf})^+, ?y)$$

Path navigation: regular path queries

Example. Co-authorship network

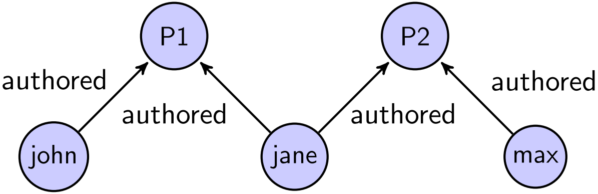
$$(?s, ?t) \;\leftarrow\; (?s, (authored/authored^{-1})^*, ?t)$$

Path navigation: regular path queries

Example. Co-authorship network

$$(?s, ?t) \leftarrow (?s, (authored/authored^{-1})^*, ?t)$$

On the graph

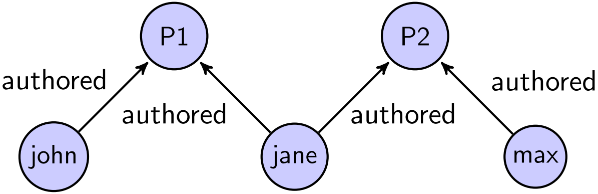


Path navigation: regular path queries

Example. Co-authorship network

$$(?s, ?t) \leftarrow (?s, (authored/authored^{-1})^*, ?t)$$

On the graph



this query evaluates to

$$\{(john, john), (john, jane), (john, max), (jane, jane), (jane, john), (jane, max), (max, max), (max, jane), (max, john)\}.$$

Unions of conjunctions of RPQs

It is natural to combine the functionalities of subgraph matching and RPQs, in the shape of unions of conjunctions of RPQs (UCRPQs):

- an edge pattern is a triple (s, r, t) where s and t can be either constants (in node set N) or variables, and r is a regular expression over $\mathcal{L}$

Unions of conjunctions of RPQs

It is natural to combine the functionalities of subgraph matching and RPQs, in the shape of unions of conjunctions of RPQs (UCRPQs):

- an edge pattern is a triple (s, r, t) where s and t can be either constants (in node set N) or variables, and r is a regular expression over $\mathcal{L}$
- a query rule is a pattern

$$head \quad \leftarrow \quad body$$

  where *body* is a set of edge patterns and *head* is a list of zero or more of the variables (possibly with repetition) appearing in *body*

Unions of conjunctions of RPQs

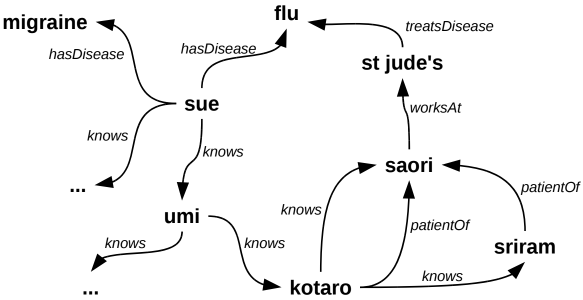
It is natural to combine the functionalities of subgraph matching and RPQs, in the shape of unions of conjunctions of RPQs (UCRPQs):

- an edge pattern is a triple (s, r, t) where s and t can be either constants (in node set N) or variables, and r is a regular expression over $\mathcal{L}$
- a query rule is a pattern

  $$\mathit{head} \quad \leftarrow \quad \mathit{body}$$

  where *body* is a set of edge patterns and *head* is a list of zero or more of the variables (possibly with repetition) appearing in *body*
- A query is a finite set of rules, each of the same arity.

Unions of conjunctions of RPQs



Example: Doctors and the patients in both their social and treatment networks

$$Q \quad = \quad \langle ?d, ?p \rangle \leftarrow (?d, \mathit{knows}^*, ?p), (?p, \mathit{patientOf}^*, ?d)$$

Regular queries

Note that all recursion in UCRPQs is captured in the Kleene star operation, R^*.

[1] http://drops.dagstuhl.de/opus/volltexte/2015/4984/pdf/11.pdf

Regular queries

Note that all recursion in UCRPQs is captured in the Kleene star operation, R^*.

... which is just the transitive closure of the binary relation defined by R ...

[1] http://drops.dagstuhl.de/opus/volltexte/2015/4984/pdf/11.pdf

Regular queries

Note that all recursion in UCRPQs is captured in the Kleene star operation, R^*.

... which is just the transitive closure of the binary relation defined by R ...

This leads us to the Regular Queries of Reutter et al., properly generalizing UCRPQs while maintaining all of their nice algorithmic properties

- equivalence is decidable; query evaluation is tractable.[1]

[1] http://drops.dagstuhl.de/opus/volltexte/2015/4984/pdf/11.pdf

Regular queries

Regular Queries. Non-recursive Datalog programs, where:

- All rules, except perhaps the output rule, are binary.
- We can take the transitive closure of any predicate in a rule body.

Regular queries

Regular Queries. Non-recursive Datalog programs, where:

- All rules, except perhaps the output rule, are binary.
- We can take the transitive closure of any predicate in a rule body.

For our co-authorship example, we have the following equivalent regular query:

$$\begin{aligned} \mathit{coAuthored}(S,T) &\leftarrow \mathit{authored}(S,X), \mathit{authored}(T,X) \\ \mathit{answer}(S,T) &\leftarrow \mathit{coAuthored}^*(S,T). \end{aligned}$$

Practical syntaxes

openCypher

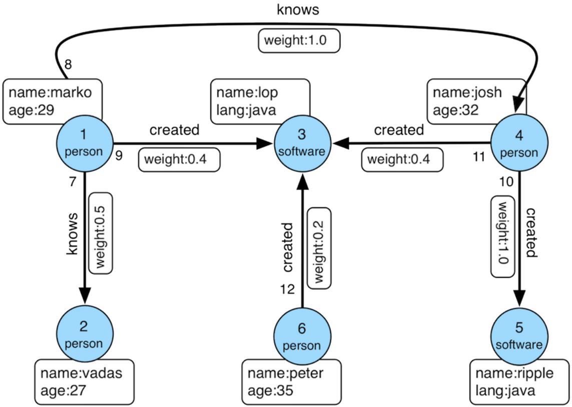
openCypher.

- Declarative graph query language of the popular open-source Neo4j graph database.

  `http://neo4j.com/developer/cypher/`

- Property graph model (cf. Angles et al. 2016)
  - directed node- and edge-labeled graph
  - nodes and edges have ID
  - nodes and edges carry sets of property-value pairs

openCypher: property graphs



(image credit: `http://tinkerpop.apache.org`)

openCypher: subgraph matching

The basic building block of queries is subgraph matching, via a MATCH clause, with isomorphic matching.

```
MATCH (n:Person)-[:Created]->(m),
      (m)<-[:Created]-(p)
WHERE n.age = 29 AND p.age < 35
RETURN p
```

openCypher: subgraph matching

The basic building block of queries is subgraph matching, via a MATCH clause, with isomorphic matching.

```
MATCH (n:Person)-[:Created]->(m),
      (m)<-[:Created]-(p)
WHERE n.age = 29 AND p.age < 35
RETURN p
```

$$\langle ?p \rangle \leftarrow (?n, created, ?m), (?p, created, ?m), \\ n.age = 29, p.age < 35, n.label = Person$$

openCypher: subgraph matching

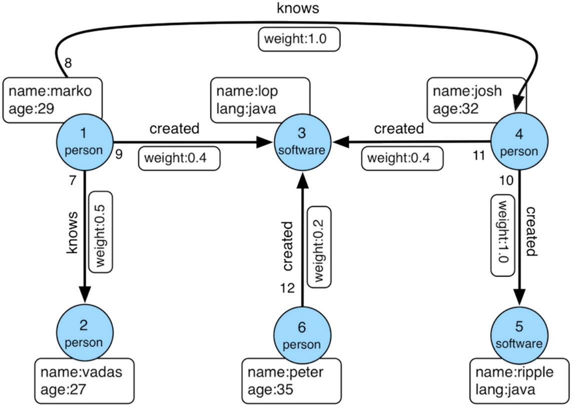
The basic building block of queries is subgraph matching, via a MATCH clause, with isomorphic matching.

```
MATCH (n:Person)-[:Created]->(m),
      (m)<-[:Created]-(p)
WHERE n.age = 29 AND p.age < 35
RETURN p
```

$$\langle ?p \rangle \leftarrow (?n, created, ?m), (?p, created, ?m),$$
$$n.age = 29, p.age < 35, n.label = Person$$

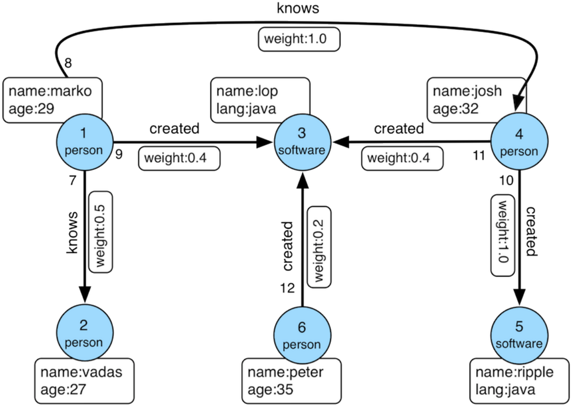
Can be further combined using UNION, applying aggregation functions, string functions, etc.

openCypher: subgraph matching




```
MATCH (n:Person)-[:Created]->(m),
      (m)<-[:Created]-(p)
WHERE n.age = 29 AND p.age < 35
RETURN p
```

openCypher: subgraph matching




```
MATCH (n:Person)-[:Created]->(m),
      (m)<-[:Created]-(p)
WHERE n.age = 29 AND p.age < 35
RETURN p
```

Result is $\{\langle 4 \rangle\}$.

openCypher: path queries

Cypher also provides support for RPQs in the MATCH clause.

```
MATCH (n:Person)-[:knows*]->(p)
WHERE n.name = "marko"
RETURN p
```

openCypher: path queries

Cypher also provides support for RPQs in the MATCH clause.

```
MATCH (n:Person)-[:knows*]->(p)
WHERE n.name = "marko"
RETURN p
```

and with bounded recursion

```
MATCH (n:Person)-[:knows*2..7]->(p)
WHERE n.name = "marko"
RETURN p
```

Can also apply * to a disjunction of symbols

Popular imperative syntaxes

Gremlin.

- Part of the Apache TinkerPop graph computing framework. `http://tinkerpop.apache.org`
- Property graph model
- Example.

```
gremlin> g.V().has('name','kotaro').
               out('knows').in('patientOf').
               values('name')
==>kotaro
==>sriram
```

Popular imperative syntaxes

Gremlin.

- Part of the Apache TinkerPop graph computing framework. http://tinkerpop.apache.org
- Property graph model
- Example.

```
gremlin> g.V().has('name','kotaro').
               out('knows').in('patientOf').
               values('name')
==>kotaro
==>sriram
```

See also the recent Sparksee API for a similar approach to graph analytics

http://sparsity-technologies.com

References

- *Querying graphs.* Angela Bonifati et al. Morgan & Claypool, 2018.
- Survey of graph database models. Renzo Angles and Claudio Gutiérrez. *ACM Comput. Surv.* 40(1), 2008.
  `http://users.dcc.uchile.cl/~cgutierr/papers/surveyGDB.pdf`
- Graph reachability queries: A survey. Jeffrey Xu Yu and Jiefeng Cheng. In *Managing and Mining Graph Data*, pages 181-215. Springer, 2010.
  `http://dx.doi.org/10.1007/978-1-4419-6045-0_6` (you must be on campus or VPN)
- Foundations of modern graph query languages. Renzo Angles et al. *arXiv* 1610.06264, 2016
  `https://arxiv.org/pdf/1610.06264.pdf`
- *Graph queries: from theory to practice*. Angela Bonifati and Stefania Dumbrava. *SIGMOD Record* 47(4): 5-16, 2018.
  `http://linkeddatabook.com/editions/1.0/`
- Querying semantic data on the web. Marcelo Arenas et al. *SIGMOD Record* 41(4): 6-17, 2012.
  `http://www.sigmod.org/publications/sigmod-record/1212/pdfs/03.principles.arenas.pdf`

Recap

1. Querying over graph data
   - query languages for graphs
   - openCypher
   - Gremlin and Sparksee