Query Languages for Graph Databases

Susan B. Davidson
CIS 700: Advanced Topics in Databases
MW 1:30-3
Towne 309

http://www.cis.upenn.edu/~susan/cis700/homepage.html
Graph Databases

• Offer a more intuitive representation for many modern applications
  • Social networks
  • Transportation networks
  • Biological pathways
  • Citation networks
  • ...

• A number of graph database engines, data models and query languages have been released over the past few years.
Outline

• Graph data models: edge-labeled and property graphs
• Graph patterns
  • Basic and complex
  • How expressed in SPARQL and Cypher
• Navigational queries
  • Regular path queries, integrated into graph pattern queries
  • How expressed in SPARQL and Cypher
Graph data models

• Edge-labeled graph, e.g. used in RDF

• Property graph, e.g. used in Neo4j
Components of property graphs

\[ V = \{n_1, n_2, n_3\} \quad E = \{e_1, e_2, e_3\} \]

\[ \rho(e_1) = (n_1, n_2) \quad \rho(e_2) = (n_1, n_2) \]

\[ \rho(e_3) = (n_3, n_2) \]

\[ \lambda(n_1) = \text{Person} \quad \lambda(n_2) = \text{Movie} \]

\[ \lambda(n_3) = \text{Person} \quad \lambda(e_1) = \text{acts_in} \quad \lambda(e_2) = \text{acts_in} \quad \lambda(e_3) = \text{acts_in} \]

\[ \sigma(n_1, \text{name}) = \text{Clint Eastwood} \quad \sigma(n_1, \text{gender}) = \text{male} \]

\[ \sigma(n_2, \text{title}) = \text{Unforgiven} \quad \sigma(n_3, \text{name}) = \text{Anna Levine} \quad \sigma(n_3, \text{gender}) = \text{female} \]

\[ \sigma(e_1, \text{role}) = \text{Bill} \quad \sigma(e_1, \text{ref}) = \text{IMDb} \]

\[ \sigma(e_3, \text{role}) = \text{Delilah} \quad \sigma(e_3, \text{ref}) = \text{IMDb} \]
Graph query languages: Core features

• Pattern matching
  • Basic graph patterns (bgp)
  • Complex graph patterns (cgp): bgp extended to include operators such as projection, union, options, etc.

• Navigation
  • Use paths as a core, e.g. regular path queries (RPQs)
  • Navigational graph patterns (ngps): paths incorporated into bgps
  • Complex navigational graph patterns (cngps): ngps extended with operators
• Graph data models: edge-labeled and property graphs
• Graph patterns
  • Basic and complex
  • How expressed in SPARQL and Cypher
• Navigational queries
  • Regular path queries, integrated into graph pattern queries
  • How expressed in SPARQL and Cypher
Basic graph patterns (edge-labeled graph)

- Find all co-stars.

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clint Eastwood</td>
<td>Anna Levine</td>
<td>Unforgiven</td>
</tr>
<tr>
<td>Anna Levine</td>
<td>Clint Eastwood</td>
<td>Unforgiven</td>
</tr>
<tr>
<td>Clint Eastwood</td>
<td>Clint Eastwood</td>
<td>Unforgiven</td>
</tr>
<tr>
<td>Anna Levine</td>
<td>Anna Levine</td>
<td>Unforgiven</td>
</tr>
</tbody>
</table>
Basic graph patterns (property graph)

- Things that (mutual) friends in the social network both like. Return first and last names, all details of the items they both like, and the date on which they both like the items.
Query result

\[x_{10}: Person\]
\[\text{firstName} = x_1, \text{lastName} = x_2\]

\[x_{11}: Person\]
\[\text{firstName} = x_3, \text{lastName} = x_4\]

\[x_{12}: \text{knows}\]
\[x_{13}: \text{knows}\]

\[x_{14}: \text{likes}\]
\[\text{date} = x_5\]

\[x_{15}: \text{likes}\]
\[\text{date} = x_6\]

\[x_{16}: x_7\]
\[x_8 = x_9\]

<table>
<thead>
<tr>
<th>(x_1)</th>
<th>(x_2)</th>
<th>(x_3)</th>
<th>(x_4)</th>
<th>(x_5)</th>
<th>(x_6)</th>
<th>(x_7)</th>
<th>(x_8)</th>
<th>(x_9)</th>
<th>(x_{10})</th>
</tr>
</thead>
<tbody>
<tr>
<td>Julie</td>
<td>Freud</td>
<td>John</td>
<td>Cook</td>
<td>14-09-15</td>
<td>23-10-15</td>
<td>Post</td>
<td>content</td>
<td>I love U2</td>
<td>...</td>
</tr>
<tr>
<td>John</td>
<td>Cook</td>
<td>Julie</td>
<td>Freud</td>
<td>23-10-15</td>
<td>14-09-15</td>
<td>Post</td>
<td>content</td>
<td>I love U2</td>
<td>...</td>
</tr>
<tr>
<td>Julie</td>
<td>Freud</td>
<td>John</td>
<td>Cook</td>
<td>14-09-15</td>
<td>23-10-15</td>
<td>Post</td>
<td>language</td>
<td>en</td>
<td>...</td>
</tr>
<tr>
<td>John</td>
<td>Cook</td>
<td>Julie</td>
<td>Freud</td>
<td>23-10-15</td>
<td>14-09-15</td>
<td>Post</td>
<td>language</td>
<td>en</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>
**Def. 3.5 (Match):** Given an edge-labeled graph $G = (V, E)$ and a bgp $Q = (V', E')$ a *match* of $Q$ in $G$ is a mapping from the set of constants and variables in $Q$ to constants in $G$ such that:

- Constants are mapped to themselves: $h(a) = a$
- Each edge of $Q$ is mapped to an edge of $G$ which preserves the structure of $Q$ in its image under $h$ in $G$: for each $(b, l, c)$ in $E'$ it holds that $(h(b), h(l), h(c))$ is in $E$

This leads to three different semantics for evaluation:

- **Homomorphism-based semantics** – currently used in SPARQL (RDF)
- **Isomorphism-based semantics**
  - No-repeated anything: no two variables can be bound to the same term
  - No-repeated node
  - No-repeated edge – currently used in Cypher (Neo4j)
• Consider the following query:
Effect of semantics: query result

- Unrestricted semantics:

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$x_4$</th>
<th>$x_5$</th>
<th>$x_6$</th>
<th>$x_7$</th>
<th>$x_8$</th>
<th>$x_9$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$e_2$</td>
<td>$n_2$</td>
<td>directs</td>
<td>$n_1$</td>
<td>Clint Eastwood</td>
<td>$e_3$</td>
<td>acts_in</td>
<td>$n_3$</td>
<td>Anna Levine</td>
</tr>
<tr>
<td>$e_3$</td>
<td>$n_2$</td>
<td>acts_in</td>
<td>$n_3$</td>
<td>Anna Levine</td>
<td>$e_2$</td>
<td>directs</td>
<td>$n_1$</td>
<td>Clint Eastwood</td>
</tr>
<tr>
<td>$e_1$</td>
<td>$n_2$</td>
<td>acts_in</td>
<td>$n_1$</td>
<td>Clint Eastwood</td>
<td>$e_3$</td>
<td>acts_in</td>
<td>$n_3$</td>
<td>Anna Levine</td>
</tr>
<tr>
<td>$e_3$</td>
<td>$n_2$</td>
<td>acts_in</td>
<td>$n_3$</td>
<td>Anna Levine</td>
<td>$e_1$</td>
<td>acts_in</td>
<td>$n_1$</td>
<td>Clint Eastwood</td>
</tr>
<tr>
<td>$e_2$</td>
<td>$n_2$</td>
<td>directs</td>
<td>$n_1$</td>
<td>Clint Eastwood</td>
<td>$e_1$</td>
<td>acts_in</td>
<td>$n_1$</td>
<td>Clint Eastwood</td>
</tr>
<tr>
<td>$e_1$</td>
<td>$n_2$</td>
<td>acts_in</td>
<td>$n_1$</td>
<td>Clint Eastwood</td>
<td>$e_2$</td>
<td>directs</td>
<td>$n_1$</td>
<td>Clint Eastwood</td>
</tr>
<tr>
<td>$e_1$</td>
<td>$n_2$</td>
<td>acts_in</td>
<td>$n_1$</td>
<td>Clint Eastwood</td>
<td>$e_1$</td>
<td>acts_in</td>
<td>$n_1$</td>
<td>Clint Eastwood</td>
</tr>
<tr>
<td>$e_2$</td>
<td>$n_2$</td>
<td>directs</td>
<td>$n_1$</td>
<td>Clint Eastwood</td>
<td>$e_2$</td>
<td>directs</td>
<td>$n_1$</td>
<td>Clint Eastwood</td>
</tr>
<tr>
<td>$e_3$</td>
<td>$n_2$</td>
<td>acts_in</td>
<td>$n_1$</td>
<td>Anna Levine</td>
<td>$e_3$</td>
<td>acts_in</td>
<td>$n_1$</td>
<td>Anna Levine</td>
</tr>
</tbody>
</table>
Effect of semantics: query result

- No-repeated anything:

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$x_4$</th>
<th>$x_5$</th>
<th>$x_6$</th>
<th>$x_7$</th>
<th>$x_8$</th>
<th>$x_9$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n_2$</td>
<td>$e_2$</td>
<td>directs</td>
<td>$n_1$</td>
<td>Clint Eastwood</td>
<td>$e_3$</td>
<td>acts_in</td>
<td>$n_3$</td>
<td>Anna Levine</td>
</tr>
<tr>
<td>$n_2$</td>
<td>$e_3$</td>
<td>acts_in</td>
<td>$n_3$</td>
<td>Anna Levine</td>
<td>$e_2$</td>
<td>directs</td>
<td>$n_1$</td>
<td>Clint Eastwood</td>
</tr>
<tr>
<td>$n_2$</td>
<td>$e_1$</td>
<td>acts_in</td>
<td>$n_1$</td>
<td>Clint Eastwood</td>
<td>$e_3$</td>
<td>acts_in</td>
<td>$n_3$</td>
<td>Anna Levine</td>
</tr>
<tr>
<td>$n_2$</td>
<td>$e_3$</td>
<td>acts_in</td>
<td>$n_3$</td>
<td>Anna Levine</td>
<td>$e_1$</td>
<td>acts_in</td>
<td>$n_1$</td>
<td>Clint Eastwood</td>
</tr>
<tr>
<td>$n_2$</td>
<td>$e_2$</td>
<td>directs</td>
<td>$n_1$</td>
<td>Clint Eastwood</td>
<td>$e_1$</td>
<td>acts_in</td>
<td>$n_1$</td>
<td>Clint Eastwood</td>
</tr>
<tr>
<td>$n_2$</td>
<td>$e_1$</td>
<td>acts_in</td>
<td>$n_1$</td>
<td>Clint Eastwood</td>
<td>$e_2$</td>
<td>directs</td>
<td>$n_1$</td>
<td>Clint Eastwood</td>
</tr>
<tr>
<td>$n_2$</td>
<td>$e_1$</td>
<td>acts_in</td>
<td>$n_1$</td>
<td>Clint Eastwood</td>
<td>$e_1$</td>
<td>acts_in</td>
<td>$n_1$</td>
<td>Clint Eastwood</td>
</tr>
<tr>
<td>$n_2$</td>
<td>$e_2$</td>
<td>directs</td>
<td>$n_1$</td>
<td>Clint Eastwood</td>
<td>$e_2$</td>
<td>directs</td>
<td>$n_1$</td>
<td>Clint Eastwood</td>
</tr>
<tr>
<td>$n_2$</td>
<td>$e_3$</td>
<td>acts_in</td>
<td>$n_3$</td>
<td>Anna Levine</td>
<td>$e_3$</td>
<td>acts_in</td>
<td>$n_3$</td>
<td>Anna Levine</td>
</tr>
</tbody>
</table>
Effect of semantics: query result

• **No-repeated node:**

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$x_4$</th>
<th>$x_5$</th>
<th>$x_6$</th>
<th>$x_7$</th>
<th>$x_8$</th>
<th>$x_9$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n_2$</td>
<td>$e_2$</td>
<td>directs</td>
<td>$n_1$</td>
<td>Clint Eastwood</td>
<td>$e_3$</td>
<td>acts_in</td>
<td>$n_3$</td>
<td>Anna Levine</td>
</tr>
<tr>
<td>$n_2$</td>
<td>$e_3$</td>
<td>acts_in</td>
<td>$n_3$</td>
<td>Anna Levine</td>
<td>$e_2$</td>
<td>directs</td>
<td>$n_1$</td>
<td>Clint Eastwood</td>
</tr>
<tr>
<td>$n_2$</td>
<td>$e_1$</td>
<td>acts_in</td>
<td>$n_1$</td>
<td>Clint Eastwood</td>
<td>$e_3$</td>
<td>acts_in</td>
<td>$n_3$</td>
<td>Anna Levine</td>
</tr>
<tr>
<td>$n_2$</td>
<td>$e_3$</td>
<td>acts_in</td>
<td>$n_3$</td>
<td>Anna Levine</td>
<td>$e_1$</td>
<td>acts_in</td>
<td>$n_1$</td>
<td>Clint Eastwood</td>
</tr>
<tr>
<td>$n_2$</td>
<td>$e_2$</td>
<td>directs</td>
<td>$n_1$</td>
<td>Clint Eastwood</td>
<td>$e_1$</td>
<td>acts_in</td>
<td>$n_1$</td>
<td>Clint Eastwood</td>
</tr>
<tr>
<td>$n_2$</td>
<td>$e_1$</td>
<td>acts_in</td>
<td>$n_1$</td>
<td>Clint Eastwood</td>
<td>$e_2$</td>
<td>directs</td>
<td>$n_1$</td>
<td>Clint Eastwood</td>
</tr>
<tr>
<td>$n_2$</td>
<td>$e_1$</td>
<td>acts_in</td>
<td>$n_1$</td>
<td>Clint Eastwood</td>
<td>$e_1$</td>
<td>acts_in</td>
<td>$n_1$</td>
<td>Clint Eastwood</td>
</tr>
<tr>
<td>$n_2$</td>
<td>$e_2$</td>
<td>directs</td>
<td>$n_1$</td>
<td>Clint Eastwood</td>
<td>$e_2$</td>
<td>directs</td>
<td>$n_1$</td>
<td>Clint Eastwood</td>
</tr>
<tr>
<td>$n_2$</td>
<td>$e_3$</td>
<td>acts_in</td>
<td>$n_1$</td>
<td>Anna Levine</td>
<td>$e_3$</td>
<td>acts_in</td>
<td>$n_1$</td>
<td>Anna Levine</td>
</tr>
</tbody>
</table>

16
Complex graph patterns

• Basic graph patterns cover natural join and selection based on equality

• Complex graph patterns add further traditional relational operators:
  • Projection, union, difference, optional (aka left-outer-join), filter
SPARQL

• W3C standard for query RDF graphs
• Based on triple patterns (subject, predicate, object), where variables are indicated by “?”
• Supports all complex graph pattern features
• Uses a homomorphism-based semantics
Sample RDF graph

[Diagram showing RDF graph with nodes and edges labeled]

- :Clint_Eastwood
  - :type :Person
  - :acts_in :Unforgiven
  - :directs :Unforgiven

- :Unforgiven
  - :type :Movie
  - :title "Unforgiven"
  - :acts_in :Anna_Levine

- :Anna_Levine
  - :type :Person
SPARQL query: projection and filter

```sparql
SELECT ?x1 ?x2
WHERE {
  FILTER(?x1 != ?x2)
}
```

- **Result:**

<table>
<thead>
<tr>
<th>?x1</th>
<th>?x2</th>
</tr>
</thead>
<tbody>
<tr>
<td>:Clint_Eastwood</td>
<td>:Anna_Levine</td>
</tr>
<tr>
<td>:Anna_Levine</td>
<td>:Clint_Eastwood</td>
</tr>
</tbody>
</table>
SPARQL queries: union, difference

SELECT ?x
WHERE {{ :Clint_Eastwood :acts_in ?x . } UNION { :Clint_Eastwood :directs ?x . }}

- **Result:** :Unforgiven

SELECT ?x
WHERE {{ ?x :acts_in :Unforgiven . } MINUS { ?x :directs :Unforgiven . }}

- **Result:** :Anna_Levine
SPARQL query: optional

```
SELECT ?x1 ?x2 ?x3
WHERE {{ ?x1 :acts_in ?x2 .} OPTIONAL { ?x1 ?x3 ?x2 . FILTER(?x3 != :acts_in) }}
```

**Result:**

<table>
<thead>
<tr>
<th>?x1</th>
<th>?x2</th>
<th>?x3</th>
</tr>
</thead>
<tbody>
<tr>
<td>:Clint_Eastwood</td>
<td>:Unforgiven</td>
<td>:directs</td>
</tr>
<tr>
<td>:Anna_Levine</td>
<td>:Unforgiven</td>
<td></td>
</tr>
</tbody>
</table>
Cypher

• Query language for Neo4j, based on patterns
• Semantics: Isomorphism-based no-repeated edges
• Syntax:
  • Nodes are written inside “( )” and edges inside of “[ ]”.
  • Filters for labels specified using “:”
  • Values for properties specified using “{ }”
• Return clause projects output variables

```cypher
MATCH (x1:Person) -[:acts_in]-> (:Movie {title:"Unforgiven"})
  <-[:acts_in]- (x2:Person)
RETURN x1, x2
```
Sample graph

$n_1 : \text{Person}$
- name = Clint Eastwood
- gender = male

$n_2 : \text{Movie}$
- title = Unforgiven
- $e_1 : \text{acts_in}$
  - role = Bill
  - ref = IMDb

$n_3 : \text{Person}$
- name = Anna Levine
- gender = female

$n_2 : \text{Movie}$
- $e_2 : \text{directs}$

$n_1 : \text{Person}$
- $e_3 : \text{acts_in}$
  - role = Delilah
  - ref = IMDb
Isomorphism-based semantics

MATCH (x1:Person) -[:acts_in]-> (:Movie {title: "Unforgiven"})
  <-[:acts_in]- (x2:Person)
RETURN x1, x2

MATCH (x1:Person) -[:acts_in]-> (x3:Movie {title: "Unforgiven"})
MATCH (x2:Person) -[:acts_in]-> (x3)
RETURN x1, x2
Cypher queries: union, difference

MATCH (:Person {name:"Clint Eastwood"}) -[:acts_in]-> (x3:Movie)
RETURN x3.title
UNION ALL MATCH (:Person {name:"Clint Eastwood"}) -[:directs]-> (x3:Movie)
RETURN x3.title

• **Result:** {“Unforgiven”, “Unforgiven”}

MATCH (x1:Person) -[:acts_in]-> (x3:Movie {title:"Unforgiven"})
WHERE NOT (x1) -[:directs]-> (x3)
RETURN x1.name

• **Result:** {“Anna Levine”}
Cypher queries: optional

```cypher
SELECT ?x1 ?x2 ?x3
WHERE {{ ?x1 :acts_in ?x2 .} OPTIONAL { ?x1 ?x3 ?x2 . FILTER(?x3 != :acts_in) }}
```

**Result:**

<table>
<thead>
<tr>
<th>?x1</th>
<th>?x2</th>
<th>?x3</th>
</tr>
</thead>
<tbody>
<tr>
<td>:Clint_Eastwood</td>
<td>:Unforgiven</td>
<td>:directs</td>
</tr>
<tr>
<td>:Anna_Levine</td>
<td>:Unforgiven</td>
<td></td>
</tr>
</tbody>
</table>
Outline

• Graph data models: edge-labeled and property graphs
• Graph patterns
  • Basic and complex
  • How expressed in SPARQL and Cypher
• Navigational queries
  • Regular path queries, integrated into graph pattern queries
  • How expressed in SPARQL and Cypher
Navigational graph queries

• Path queries are added to basic graph queries: $x \xrightarrow{\alpha} y$

• $\alpha$ is a regular expression over the set of edge labels
  • knows+
  • knows+.likes
  • knows+.\(\text{likes} \mid \text{dislikes}\)
  • *

• Inverse operator to allow backward edge traversal
  • acts_in. acts_in ^
• All paths in G whose label satisfies $\alpha$
• But there may be an infinite number of such paths (cycles), so in practice:
  • Arbitrary path semantics: test existence of such a path, or pairs of nodes connected by such paths (finite)
  • Shortest path semantics
  • No-repeated node semantics (simple paths)
  • No-repeated edge semantics (used in Cypher)
• Output:
  • Boolean
  • Nodes
  • Paths
  • Graphs (compact representation)
Navigational graph patterns (ngps)

- Complex navigational graph patterns (cngps)
  - Project $x_5$: recommended posts for Julie
  - Union: recommended posts for John or Julie
  - Intersection: recommended posts for both John and Julie
  - Difference: recommended posts for Julie but not John

Fig. 10. A navigational graph pattern that characterises the friends of friends of Julie that like a post with a tag she that she follows.
SPARQL examples

• All pairs of actors who have finite collaboration distance

```
SELECT ?x ?y
WHERE  ?x (:acts_in/?acts_in)* ?y
```

• All people with a finite Erdos-Bacon number

```
SELECT ?x
```

• Posts recommended to Julie but not to John

```
SELECT ?x
WHERE {
  {
    { :Julie :knows+/:likes ?x . ?x :hasTag/:hasFollower :Julie . }
    MINUS
    { :John :knows+/:likes ?x . ?x :hasTag/:hasFollower :John . }
  }
}
```
SPARQL: limited form of negation

SELECT ?y
WHERE { :Clint_Eastwood (!{:type,:directs})* ?y }

- Will match :Unforgiven (IRI), “Unforgiven” (string) and :Movie (IRI)
- Called “negated property sets”
Cypher

• Does not support full regular expressions, but allows transitive closure over a single edge label or edge property/value pair

```
MATCH (x1:Person) -[:knows*]-> (x2:Person)
RETURN x1,x2
```

• Uses no-repeated-edge semantics for cngps and bag semantics

```
MATCH (x1) -[*]-> (x2)
RETURN x1,x2
```

What would this return?
Cypher: shortest and bounded paths

- Returns a single shortest witnessing path (could also use `allShortestPaths` to get all shortest paths)

```cypher
MATCH ( julie:Person {firstname:"Julie"} ),
p = shortestPath((julie) -[:knows*]-> (x:Person) )
RETURN p
```

- Star operator on edge label - can also specify bounds on the length (e.g. `[:knows*2..7]`

```cypher
MATCH (x1:Person firstName:"Julie") -[:knows*]-> (x2:Person)
MATCH (x2) -[:likes]-> () -> [:hasTag] -> (x3)
MATCH (x3) -[:hasFollower]-> (x1)
RETURN x2
```
Ok, but what about aggregation?

We’ll take a look at Cypher.
Cypher and aggregation

Common aggregation functions are supported: count, sum, avg, min, and max

MATCH (p:Person) 
RETURN count(*) as headcount;

MATCH (actor:Person)-[:ACTED_IN]->(movie:Movie)<-[:DIRECTED]-(director:Person) 
RETURN actor,director,count(*) AS collaborations

<table>
<thead>
<tr>
<th>&quot;actor&quot;</th>
<th>&quot;director&quot;</th>
<th>&quot;collaborations&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>{&quot;born&quot;: &quot;1946&quot;, &quot;name&quot;: &quot;Susan Sarandon&quot;}</td>
<td>{&quot;born&quot;: &quot;1965&quot;, &quot;name&quot;: &quot;Lana Wachowski&quot;}</td>
<td>&quot;1&quot;</td>
</tr>
<tr>
<td>{&quot;born&quot;: &quot;1960&quot;, &quot;name&quot;: &quot;Annabella Sciorra&quot;}</td>
<td>{&quot;born&quot;: &quot;1956&quot;, &quot;name&quot;: &quot;Vincent Ward&quot;}</td>
<td>&quot;1&quot;</td>
</tr>
<tr>
<td>{&quot;born&quot;: &quot;1956&quot;, &quot;name&quot;: &quot;Tom Hanks&quot;}</td>
<td>{&quot;born&quot;: &quot;1951&quot;, &quot;name&quot;: &quot;Robert Zemeckis&quot;}</td>
<td>&quot;2&quot;</td>
</tr>
<tr>
<td>{&quot;born&quot;: &quot;1953&quot;, &quot;name&quot;: &quot;David Morse&quot;}</td>
<td>{&quot;born&quot;: &quot;1959&quot;, &quot;name&quot;: &quot;Frank Darabont&quot;}</td>
<td>&quot;1&quot;</td>
</tr>
</tbody>
</table>
Cypher: beyond min, max, sum, count

- **Collect()** function collects all aggregated values into a list

```cypher
MATCH (m:Movie)<-[[:ACTED_IN]]-(a:Person)
RETURN m.title AS movie, collect(a.name) AS cast, count(*) AS actors
```

| "movie"                | "cast"                                                                 | "actors"
|------------------------|------------------------------------------------------------------------|--------
| "You've Got Mail"     | ["Dave Chappelle","Parker Posey","Steve Zahn","Meg Ryan","Tom Hanks","Greg Kinnear"] | "6"    
| "Apollo 13"           | ["Tom Hanks","Kevin Bacon","Ed Harris","Bill Paxton","Gary Sinise"]       | "5"    
| "Johnny Mnemonic"     | ["Dina Meyer","Takeshi Kitano","Ice-T","Keanu Reeves"]                | "4"    
| "Stand By Me"         | ["Marshall Bell","Kiefer Sutherland","John Cusack","Corey Feldman","Jerry O'Connell","River Phoenix","Wil Wheaton"] | "7"    
| "The Polar Express"   | ["Tom Hanks"]                                                          | "1"    

38
MATCH (person:Person)-[:ACTED_IN]-(m:Movie)
WITH person, count(*) AS appearances, collect(m.title) AS movies
WHERE appearances > 1
RETURN person.name, appearances, movies

<table>
<thead>
<tr>
<th>&quot;person.name&quot;</th>
<th>&quot;appearances&quot;</th>
<th>&quot;movies&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Cuba Gooding Jr.&quot;</td>
<td>&quot;4&quot;</td>
<td>[&quot;A Few Good Men&quot;,&quot;Jerry Maguire&quot;,&quot;As Good as It Gets&quot;,&quot;What Dreams May Come&quot;]</td>
</tr>
<tr>
<td>&quot;Oliver Platt&quot;</td>
<td>&quot;2&quot;</td>
<td>[&quot;Frost/Nixon&quot;,&quot;Bicentennial Man&quot;]</td>
</tr>
<tr>
<td>&quot;Philip Seymour Hoffman&quot;</td>
<td>&quot;2&quot;</td>
<td>[&quot;Twister&quot;,&quot;Charlie Wilson's War&quot;]</td>
</tr>
<tr>
<td>&quot;Sam Rockwell&quot;</td>
<td>&quot;2&quot;</td>
<td>[&quot;The Green Mile&quot;,&quot;Frost/Nixon&quot;]</td>
</tr>
<tr>
<td>&quot;Greg Kinnear&quot;</td>
<td>&quot;2&quot;</td>
<td>[&quot;As Good as It Gets&quot;,&quot;You've Got Mail&quot;]</td>
</tr>
<tr>
<td>&quot;Zach Grenier&quot;</td>
<td>&quot;2&quot;</td>
<td>[&quot;RescueDawn&quot;,&quot;Twister&quot;]</td>
</tr>
<tr>
<td>&quot;Rosie O'Donnell&quot;</td>
<td>&quot;2&quot;</td>
<td>[&quot;A League of Their Own&quot;,&quot;Sleepless in Seattle&quot;]</td>
</tr>
</tbody>
</table>

Source: https://neo4j.com/developer/cypher-query-language/
Summary

- SPARQL is the W3C recommended language for querying RDF graphs, and is based on an edge-labeled graph model.
  - Supports all complex graph patterns
  - Homomorphism-based bag semantics
  - Allows more than regular path queries (e.g. inverse)
  - Output can be boolean or nodes

- Cypher is the query language for Neo4j, and is based on a property graph model.
  - Supports all complex graph patterns
  - No-repeated-edge bag semantics
  - Allows only a fragment of regular path queries (repeated label)
  - Output can be boolean/nodes/paths/graphs