Dresden Database Systems Group

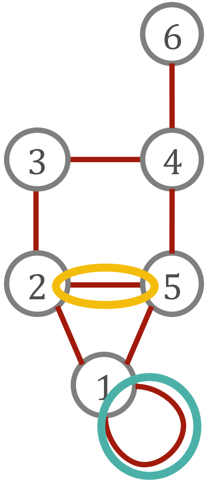
Graph Database System Neo4j

Hannes Voigt

Some Bachground

Adjacency Matrix

Undirected graph without labels

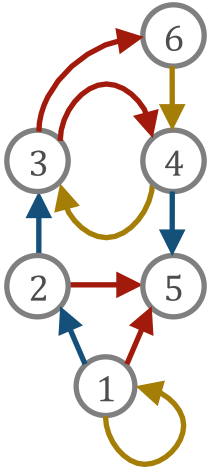


target

$$\text{source}\quad \begin{array}{c|cccccc} & 1 & 2 & 3 & 4 & 5 & 6 \\ \hline 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 2 & 1 & 0 & 1 & 0 & 1 & 0 \\ 3 & 0 & 1 & 0 & 1 & 0 & 0 \\ 4 & 0 & 0 & 1 & 0 & 1 & 1 \\ 5 & 1 & 1 & 0 & 1 & 0 & 0 \\ 6 & 0 & 0 & 0 & 1 & 0 & 0 \end{array}$$

symmetric

Directed graph with edge labels



target

$$\text{source}\quad \begin{array}{c|cccccc} & 1 & 2 & 3 & 4 & 5 & 6 \\ \hline 1 & 1 & 3 & 0 & 0 & 2 & 0 \\ 2 & 0 & 0 & 3 & 0 & 2 & 0 \\ 3 & 0 & 0 & 0 & 2 & 0 & 2 \\ 4 & 0 & 0 & 1 & 0 & 3 & 0 \\ 5 & 0 & 0 & 0 & 0 & 0 & 0 \\ 6 & 0 & 0 & 0 & 1 & 0 & 0 \end{array}$$

not symmetric

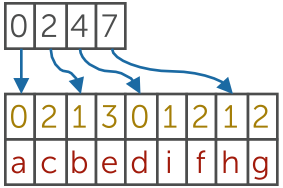
Adjacency Lists

Compression of adjacency matrix

- Compression scheme:

$$\begin{array}{c|cccc} & 0 & 1 & 2 & 3 \\ \hline 0 & a & & c & \\ 1 & & b & & e \\ 2 & d & i & f & \\ 3 & & h & g & \end{array}$$

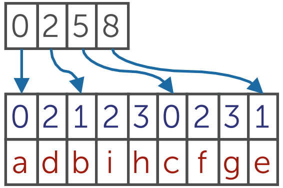
Compressed Sparse Row



0	2	4	7					
0	2	1	3	0	1	2	1	2
a	c	b	e	d	i	f	h	g

source-oriented

Compressed Sparse Column



0	2	5	8					
0	2	1	2	3	0	2	3	1
a	d	b	i	h	c	f	g	e

target-oriented

Coordinate list

$(0,0,a)$
$(0,2,c)$
$(1,1,b)$
$\vdots$
$(2,2,f)$
$(3,1,h)$
$(3,2,g)$

The same!!!

Almost the same!!!

Adjacency list

- Source vertex together outgoing edges

Without edge labels

0 -> (0,2)
1 -> (1,3)
2 -> (0,1,2)
3 -> (1,2)

With edge labels

0 -> ([0,a],[2,c])
1 -> ([1,b],[3,e])
2 -> ([0,d],[1,i],[2,f])
3 -> ([1,h],[2,g])

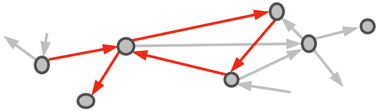
With edge properties

0 -> ([0,a,(weight=4)],[2,c,(weight=3)])
1 -> ([1,b,(weight=3)],[3,e,(weight=2)])
2 -> ([0,d,(weight=5)],[1,i,(weight=2)],…)
3 -> ([1,h,(weight=9)],[2,g,(weight=7)])

Basic Terminology

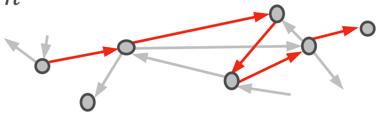
Walk

- Sequence of edges connecting vertices
  - $w_{v_1,v_n} = v_1 e_1 v_2 \dots v_{n-1} e_{n-1} v_n$ with $e_i = (v_i, v_{i+1}) \in E, 1 \le i < n$
  - n is the length of the walk



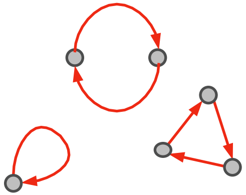
Path

- Walk connecting distinct vertices
  - $p_{v_1 v_n} = v_1 e_1 v_2 \dots v_{n-1} e_{n-1} v_n$ with $\forall v_i, v_j : i \neq j \rightarrow v_i \neq v_j$ and $e_i = (v_i, v_{i+1}) \in E, 1 \le i < n$
  - Length is number of edges or sum of edge weights



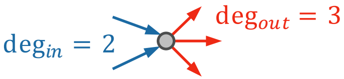
Cycle

- Walk with $v_1 = v_n$ is a cycle
- Graph is acyclic iff $\nexists v \in V : w_{v,v} \in G$



Degree (or valency)

- In/out degree of a vertex: Number of incoming/outgoing edges of that vertex
  - $\deg_{out}(v) = |\{(v,u) \in E\}|$
  - $\deg_{in}(v) = |\{(u,v) \in E\}|$
  - $\deg(v) = \deg_{out}(v) + \deg_{in}(v)$

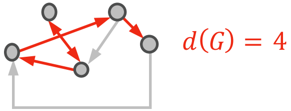


Distance

- Distance between two vertices in a graph is number of edges in a shortest path connecting them
- $d(v,u) = \min_{p_{v,u} \in G} |p_{v,u}|$

Diameter

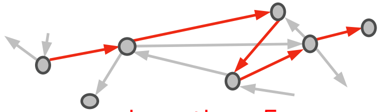
- Maximum eccentricity of any vertex in the graph
- $d(G) = \max_{v \in V} \epsilon(v)$



Finding Shortest Paths

Unweighted shortest paths

- Length of path its number of edges
- Restriction to simple paths (w/o cycles)

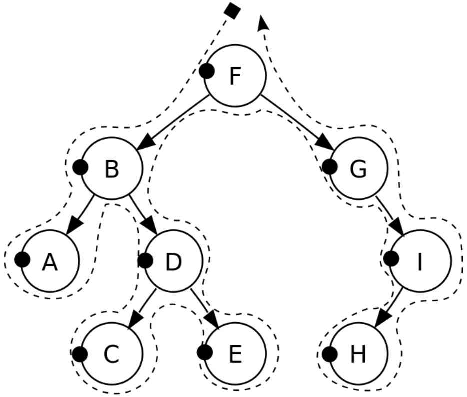


length = 5

- Two main ways of path search
  - Depth-first search (DFS)
  - Breadth-first search (BFS)

Depth-first search (DFS)

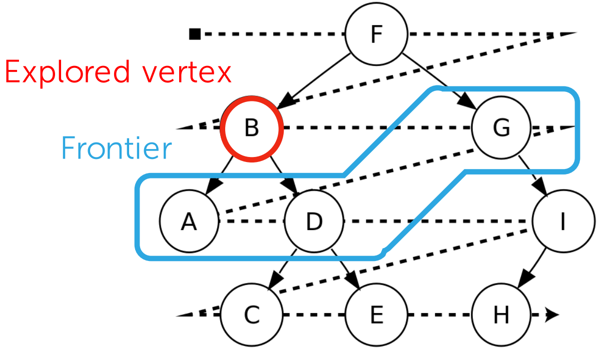
- Search tree is deepened as much as possible on each child before going to the next sibling
- Lower space complexity
- Has to examine whole graph to find shortest path between two nodes



[https://commons.wikimedia.org/wiki/File:Sorted_binary_tree_preorder.svg]

Breadth-first search (BFS)

- Search tree is broadened as much as possible on each depth before going to the next depth
- Potential large space required
- Find shortest path between two nodes first (before finding a longer one)



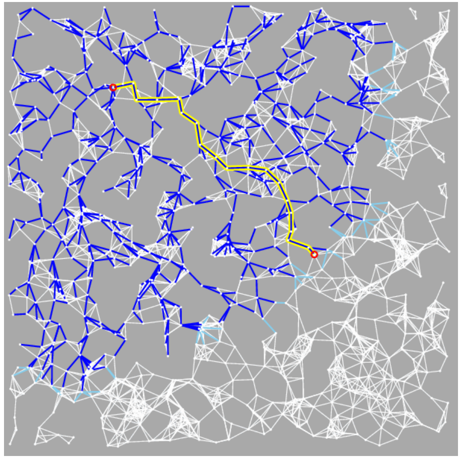
[https://commons.wikimedia.org/wiki/File:Sorted_binary_tree_breadth-first_traversal.svg]

Bidirectional BFS

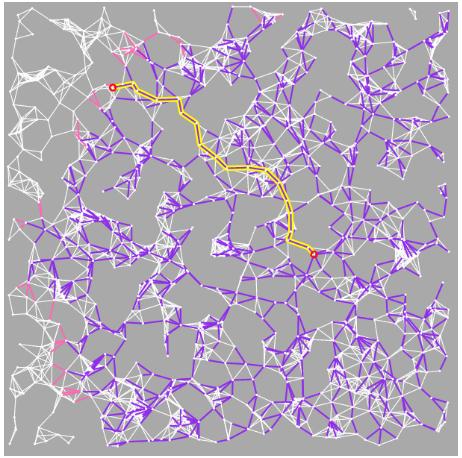
Idea: search from start and end vertex

- Alternatingly explore vertices on both sides
  - Optimization: explore vertices on the side with smaller frontier
- Algorithm stops when both BFS meet
  - When discovering a new vertex, each BFS check if that vertex is in frontier of other side

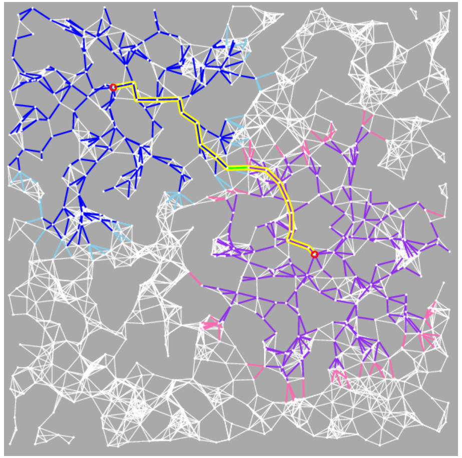
Forward (598 vertices explored)



Backward (860 vertices explored)



Bidirectional (448 vertices explored)



[http://euler.slu.edu/~goldwasser/class/slu/csci462/2010_Spring/assignments/asgn03/]

Centrality Measures

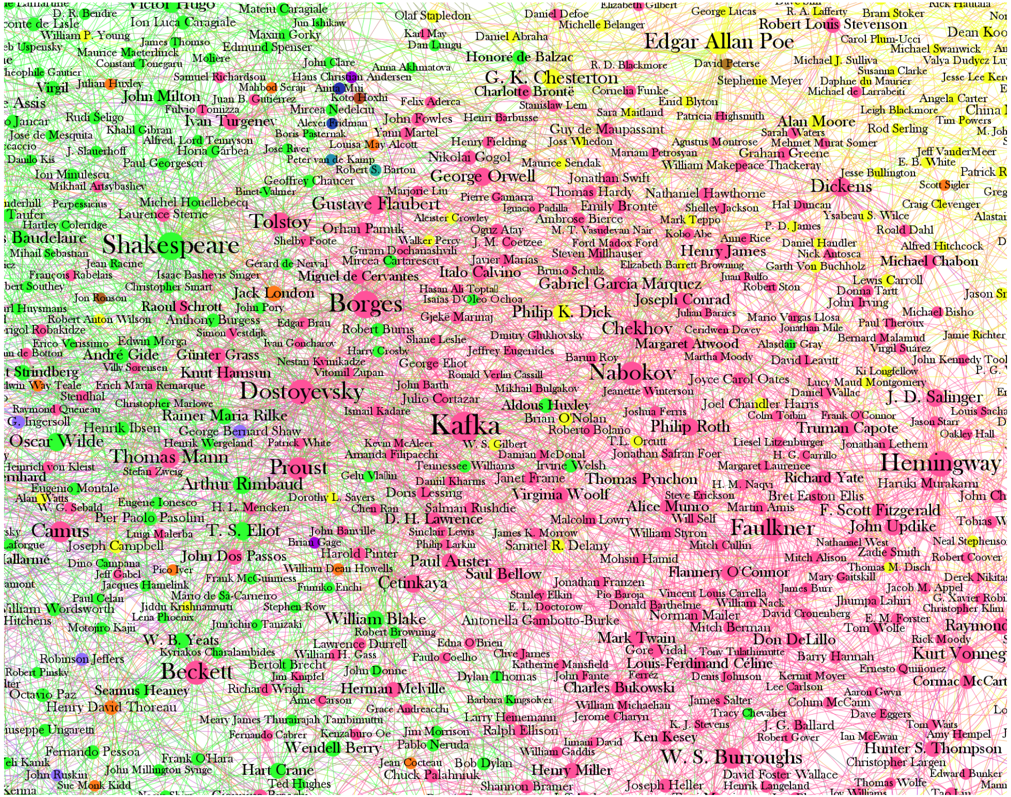
Question: Who are the key players in a graph

- Most social contacts (vaccination schedules)
- Most influential thinkers/papers (reading lists)
- Most important website (web search)
- Most important distributers (supply network)
- etc.

- Can we measure that?

Yes! With centrality Measures!

- Centrality measures identify the most important vertices within a graph

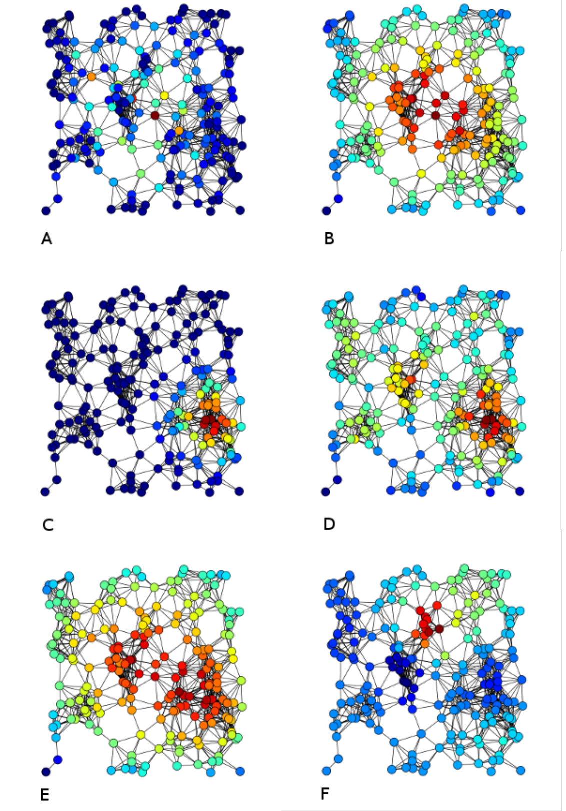


[http://brendangriffen.com/blog/gow-influential-thinkers]

Centrality Measures

Various centrality measure have been defined

- Betweenness centrality (A)
  - Number of shortest paths between all other vertices that pass through that vertex
- Closeness centrality (B)
  - Reciprocal of the sum of distances to all other vertices
  - Harmonic centrality (E) uses the sum of reciprocal of distances instead
- Eigenvector centrality/Eigencentrality (C)
  - Score of a vertex contributes to score of neighboring vertices
  - Page rank is variant of eigenvector centrality
- Katz centrality (F)
  - Number of all vertices that can be connected through a path
  - Contributions of distant nodes are penalized
  - Degree centrality (D) only considers direct neighbors

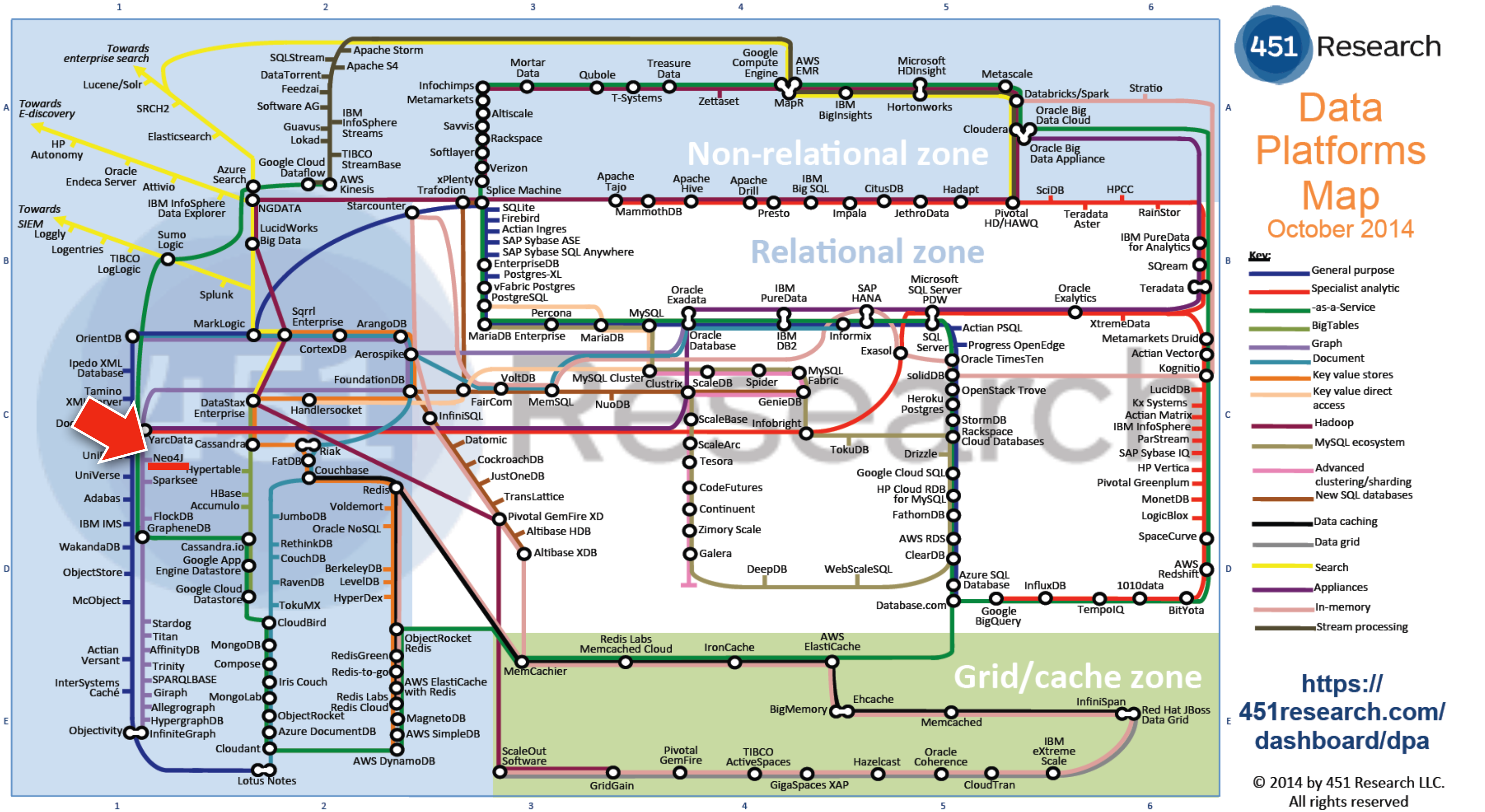


[https://commons.wikimedia.org/wiki/File:6_centrality_measures.png]

Neo4j

Database Systems Landscape

Dresden Database Systems Group

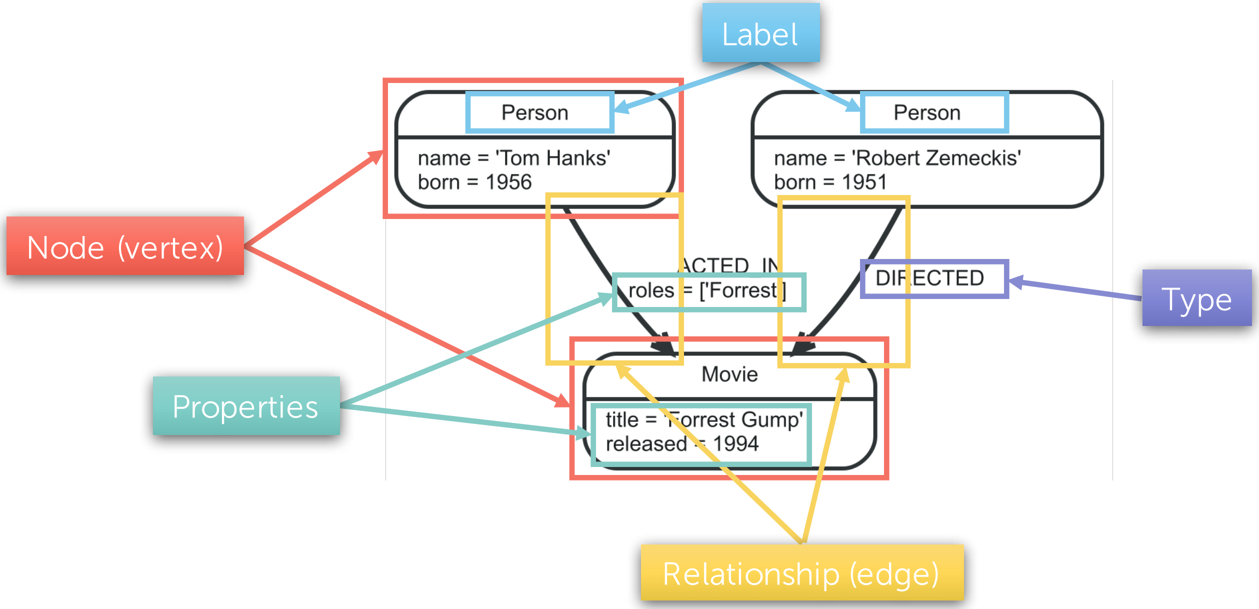


https://451research.com/dashboard/dpa

© 2014 by 451 Research LLC.
All rights reserved

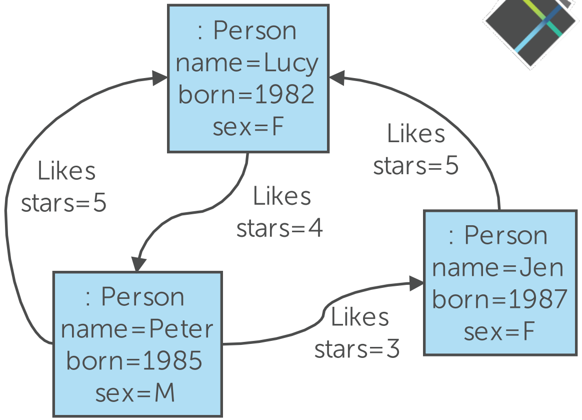
Neo4j Terminology

[http://neo4j.com/docs/developer-manual/current/#graphdb-concepts]



Match

[http://neo4j.com/docs/developer-manual/current/#query-match]
[http://neo4j.com/docs/developer-manual/current/#query-return]



MATCH-CLAUSE

- Primary way of getting data from a Neo4j database
- Allows you to specify the patterns
- Named pattern element, e.g. (p:Person), will be bound to the match instance
- Query can have multiple MATCH-clauses

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

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

RETURN-CLAUSE

- Projects to the result set
- Allows projection to nodes, edges, and properties

```
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

Pattern Syntax

[http://neo4j.com/docs/developer-manual/current/#cypher-intro-patterns]

Vertex pattern

()	unidentified vertex
(matrix)	vertex identified by variable *matrix*
(:Movie)	unidentified vertex with label *Movie*
(matrix:Movie:Action)	vertex with labels *Movie* and *Action* identified by variable *matrix*
(matrix:Movie {title: "The Matrix"})	+ property *title* equal the string “The Matrix”
(matrix:Movie {title: "The Matrix", released: 1997})	+ property *released* equal the integer *1997*

Edge pattern

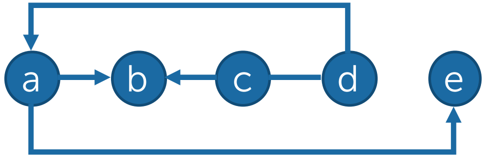
-->	unidentified edge
-[role]->	edge identified by variable *role*
-[:ACTED_IN]->	unidentified edge with label *ACTED_IN*
-[role:ACTED_IN]->	edge with label *ACTED_IN* identified by variable *role*
-[role:ACTED_IN {roles: ["Neo"]}]->	+ property *roles* contains the string “Neo”

Pattern Syntax

[http://neo4j.com/docs/developer-manual/current/#cypher-intro-patterns]

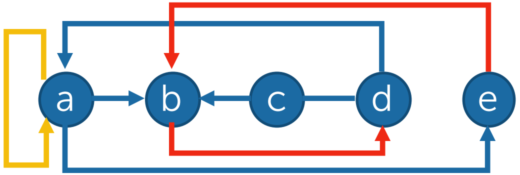
PATH PATTERNS

- String of alternating vertex pattern and edge pattern
- Starting and ending with a vertex pattern
- (a)-->(b)<--(c)--(d)-->(a)-->(e)
- (keanu:Person:Actor {name: "Keanu Reeves"}) -[role:ACTED_IN {roles: ["Neo"]}]-> (matrix:Movie {title: "The Matrix"})



GRAPH PATTERNS

- One or multiple path patterns
- Path patterns should have at least one shared variable
- Without shared variable graph pattern is disconnected
  - Results in a cross-product of the results for connected sub patterns
  - Quadrating blow up in result size and computational complexity
- (a)-->(b)<--(c)--(d)-->(a)-->(e), (e)-->(b)-->(d), (a)-->(a)



Return

[http://neo4j.com/docs/developer-manual/current/#query-return]
[http://neo4j.com/docs/developer-manual/current/#cypher-expressions]

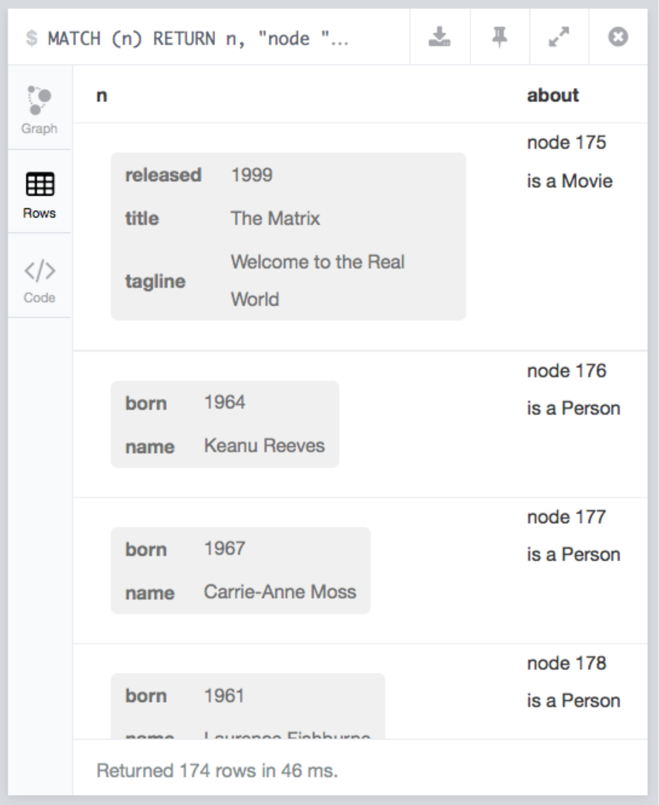
RETURN-CLAUSE

- Defines what to include in the query result set
- Comparable with relational projection
- Only once per query
- Allows to return nodes, edges, properties, or any expressions
- Column can be rename using AS <new name>

EXAMPLE

- MATCH (n)
  RETURN n, "node " + id(n) +" is " +

  ```
        CASE    WHEN n.title IS NOT NULL THEN "a Movie”
                WHEN EXISTS(n.name)  THEN "a Person”
                ELSE "something unknown”
        END AS about
  ```



```
$ MATCH (n) RETURN n, "node "...
```

n	about
released 1999 title The Matrix tagline Welcome to the Real World	node 175 is a Movie
born 1964 name Keanu Reeves	node 176 is a Person
born 1967 name Carrie-Anne Moss	node 177 is a Person
born 1961 name Laurence Fishburne	node 178 is a Person

Returned 174 rows in 46 ms.

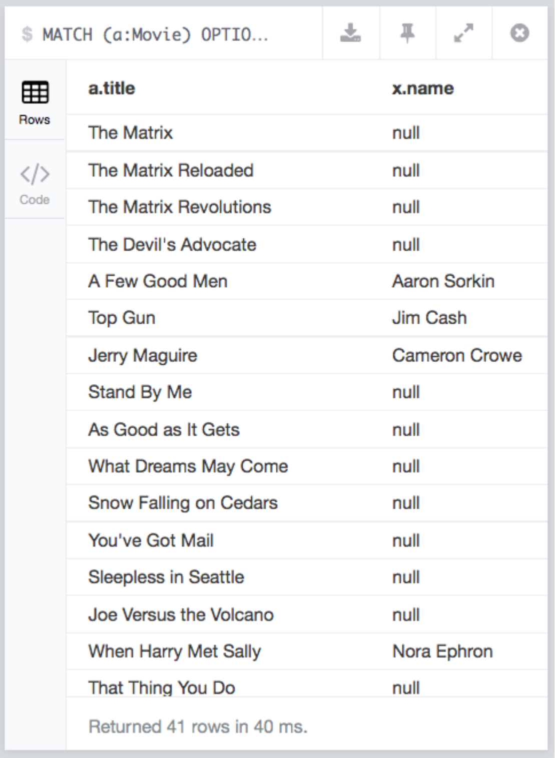
Optional Match & Where

[http://neo4j.com/docs/developer-manual/current/#query-optional-match]
[http://neo4j.com/docs/developer-manual/current/#query-where]

OPTIONAL MATCH-CLAUSE

- Matches patterns against your graph database, just like MATCH
- Matches the complete pattern or not
- If no matches are found, OPTIONAL MATCH will use NULLs as bindings
- Like relational outer join
- Example:

```
MATCH (a:Movie)
OPTIONAL MATCH (a)<-[:WROTE]-(x)
RETURN a.title, x.name
```



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
As Good as It Gets	null
What Dreams May Come	null
Snow Falling on Cedars	null
You've Got Mail	null
Sleepless in Seattle	null
Joe Versus the Volcano	null
When Harry Met Sally	Nora Ephron
That Thing You Do	null

Returned 41 rows in 40 ms.

WHERE

- After an (OPTIONAL) MATCH, it adds constraints to the (optional) match
- WHERE becomes part of the pattern
- After a WITH, it just filters the result
- Syntax: WHERE <expression>
- Example:

```
MATCH (n)
WHERE n.name = 'Peter' XOR (n.age < 30 AND n.name = 'Tobias')
            OR NOT (n.name =~ 'Tob.*' OR n.name CONTAINS 'ete')
RETURN n
```

Matching Paths

[http://neo4j.com/docs/developer-manual/current/#_variable_length]
[http://neo4j.com/docs/developer-manual/current/#_assigning_to_path_variables]

Variable length path patterns

- Repetitive edge types can be expressed by specifying a length with lower and upper bounds
- Example: (a)-[:x*2]->(b) is equal to (a)-[:x]->()-[:x]->(b)
- More examples: (a)-[*3..5]->(b)
  (a)-[*3..]->(b)
  (a)-[*..5]->(b)
  (a)-[*]->(b)
- Complete example: MATCH (me)-[:KNOWS*1..2]-(remote_friend)
  WHERE me.name = "Filipa" RETURN remote_friend.name
- Matches unique paths (relationship uniquness), not unique reachable nodes!!!
- Particularly the unbounded [*] easily matches larger numbers of paths -> exponential blowup!!!

Path variables

- Assign matched paths to variable or further processing
- Example: p = ((a)-[*3..5]->(b))

Matching Shortest Paths

[http://neo4j.com/docs/developer-manual/current/#query-shortest-path]
[http://neo4j.com/docs/developer-manual/current/#query-shortestpath-planning]

SHORTEST PATHS

- Path between two nodes with minimum number of edges
- Apply the shortestPath/allShortestPath function to a path pattern to match single/all shortest paths
- Additional filter predicates can be given with WHERE clause
  - Universal (NONE/ALL) predicates can be evaluated during shortest path search
  - Other predication can be evaluated only after shortest path has been discovered
- Fast evaluation algorithm
  - Bidirectional BFS
  - Standard for paths without additional predicates and path with universal predicates
- Slow evaluation algorithm
  - DFS
  - Fallback for paths with non-universal predicates
- Example (fast evaluation):

```
MATCH (m { name:"Martin Sheen" }),(o { name:"Oliver Stone" }), p = shortestPath((m)-[*..15]-(o))
WHERE NONE(r IN rels(p) WHERE type(r)= "FATHER") RETURN p
```

Max. 15 hops

- Example (fast evaluation):

```
MATCH (m { name:"Martin Sheen" }),(o { name:"Oliver Stone" }), p = shortestPath((m)-[*..15]-(o))
WHERE length(p) > 1 RETURN p
```

Aggregation

[http://neo4j.com/docs/developer-manual/current/#query-aggregation]
[http://neo4j.com/docs/developer-manual/current/#_assigning_to_path_variables]

In RETURN-clause

- Implicit group by
  - Expressions without an aggregation function will grouping keys
  - Expressions with an aggregation function will produce aggregates
- DISTINCT within the aggregation function removes duplicates in a group before the aggregation
- Aggregation function: COUNT, SUM, AVG, MIN, MAX, STDEV, STDEVP, PERCENTILEDISC, PERCENTILECONT, and COLLECT – collects all the values into a list

In WITH-clause

- Like a process pipe
- Chains query parts together, piping the results from one to be used as starting points in the next
- Like RETURN, WITH defines – including aggregation – the output before it is passed on
- Allows to
  - Filter on aggregates
  - Aggregation of aggregates
  - Limit search space based on order of properties or aggregates

Aggregation

[http://neo4j.com/docs/developer-manual/current/#query-aggregation]

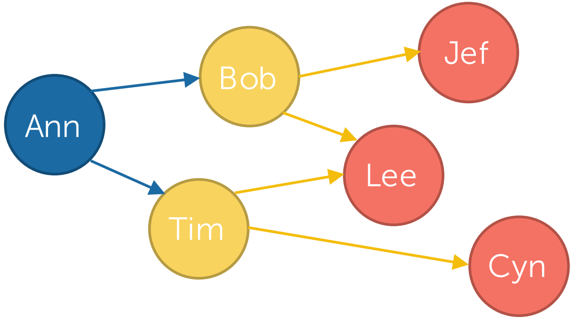
IN RETURN-CLAUSE

- Implicit group by
  - Expressions without an aggregation function will be group keys
  - Expressions with an aggregation function will produce aggregates
- DISTINCT within the aggregation function removes duplicates in a group before the aggregation
- Aggregation function: COUNT, SUM, AVG, MIN, MAX, STDEV, STDEVP, PERCENTILEDISC, PERCENTILECONT, and COLLECT – collects all the values into a list
- Example:

```
MATCH (me:Person {name:'Ann'})-->(friend:Person)-->(friend_of_friend:Person)
RETURN me.name, count(DISTINCT friend_of_friend), count(friend_of_friend)
```

group key

aggregates



Result

me	COUNT DISTINCT	COUNT
Ann	3	4

IN WITH-CLAUSE

- See next slide

Query Composition

[http://neo4j.com/docs/developer-manual/current/#query-with]

WITH-CLAUSE

- Like a process pipe
- Chains query parts together, piping the results from one to be used as starting points in the next
- Like RETURN, WITH defines – including aggregation – the output before it is passed on

- Filter on aggregates
  Example: Soccer team on average younger than 25

  ```
  MATCH (p)-[:PLAYS]->(t) WITH t, AVG(p.age) AS a WHERE a < 25 RETURN t
  ```

- Aggregation of aggregates
  Example: Average age of the youngest player in each team

  ```
  MATCH (p)-[:PLAYS]->(t) WITH t, MIN(p.age) AS a RETURN AVG(a)
  ```

- Limit search space based on order of properties or aggregates
  Example: Friends of five best friends

  ```
  MATCH (p)-[f:FRIENDS]->(p2)
  WITH f,p2 ORDER BY f.rating DESC LIMIT 5
  MATCH (p2)-[f:FRIENDS]->(p3) RETURN DISTINCT p3
  ```

Exercise

Exercise

Preparation

- Download and install neo4j community edition:
  - For installation follow standard download procedure: http://neo4j.com/download/
  - For portable usage without installation download archive (tar/zip): https://neo4j.com/download/other-releases/ and follow OS-specific installing instructions at download page

- Import the movie database > `:play movie graph` > 2nd page > click on code > execute

- Try out query:
```
MATCH (n) WITH COUNT(n) AS numVertices
MATCH (a)-[e]->(b)
RETURN numVertices, COUNT(e) AS numEdges
```
- Try out query: `MATCH (n) RETURN n`

Expected Result

numVertices	numEdges
171	253

Add Data

- Add movie, actor (three main characters), director as vertices and ACTED_IN/DIRECTED edges for the movie *The Bridges of Madison County* http://www.imdb.com/title/tt0112579/
- Do not insert vertices that already exist in the database!!!

Exercise

Simple patterns

- Find all actors that directed a movie they also acted in and return actor and movie nodes
- Find all reviewer pairs, one following the other and both reviewing the same movie, and return entire subgraphs
- Find all reviewer pairs, one following the other, and return the two reviewers and a movie the may have reviewed both
- Restrict previous query so that the name of the followed reviewer is not 12 characters long
  - Try a different position for the where clause. Explain why this gives a different result.
- Find all actors that acted in a movie together after 2010 and return the actor names and movie node
- By extending the previous query, find all movies that the cast of the movies found before also acted in

Matching semantics of Neo4j

- Which matching semantics does Neo4j implement? Homomorphism, Isomorphism, Induced subgraph isomorphism?
- Remove duplicates for pattern (x)--(y)
- Match pattern (a1)-[:REVIEWED]->(m)<-[:REVIEWED]-(a2) as induced subgraph
- Find all actor pairs that acted in multiple movies together
- Find all pairs of actor–movie subgraphs with equal roles (on ACTS_IN edges), return actors names, roles, and movie titles

Exercise

Paths

- Match all reviewers and the one they are following directly or via another a third reviewer
- Count the number of paths of at most length 4 starting from *Clint Eastwood* ignoring edge direction
- Count the number of paths of at most length 10 starting from *Clint Eastwood* ignoring edge direction
- Count the number of paths of at most length 11 starting from *Clint Eastwood* ignoring edge direction
- Count the number of nodes reachable in at most 4 hops starting from *Clint Eastwood* ignoring edge direction
- Count the number of nodes reachable in at most 10 hops starting from *Clint Eastwood* ignoring edge direction
- Count the number of nodes reachable in at most 11 hops starting from *Clint Eastwood* ignoring edge direction

Exercise

Young and old movies

- Determine the average age of the Apollo 13 cast at the time of the movie's release
- Find the movies with the top-10 oldest cast at the time of the movie's release
  - Return movie and average age rounded to two decimal ordered by descending age
- Find average age of youngest actors in movie casts at time of release
- Find ACTED_IN subgraph of the movie with the youngest cast at the time of the movie's release
- Determine the movie with youngest and movie with oldest cast and their age difference rounded to two decimal points

Adjacency list and distributions

- Return the whole graph a simple adjacency list of vertex ids ordered by decreasing vertex degree
- Return out degree distribution ordered by ascending degree
- Return degree distribution ordered by ascending degree
- Return edge types with number of instances order by decreasing instances number

Exercise

Six degrees of Kevin Bacon [https://en.wikipedia.org/wiki/six_degrees_of_kevin_bacon]

- Determine the Bacon number of Clint Eastwood
- Count for each Bacon number the number of actor
  - Return degree and number of actors ordered by ascending degree

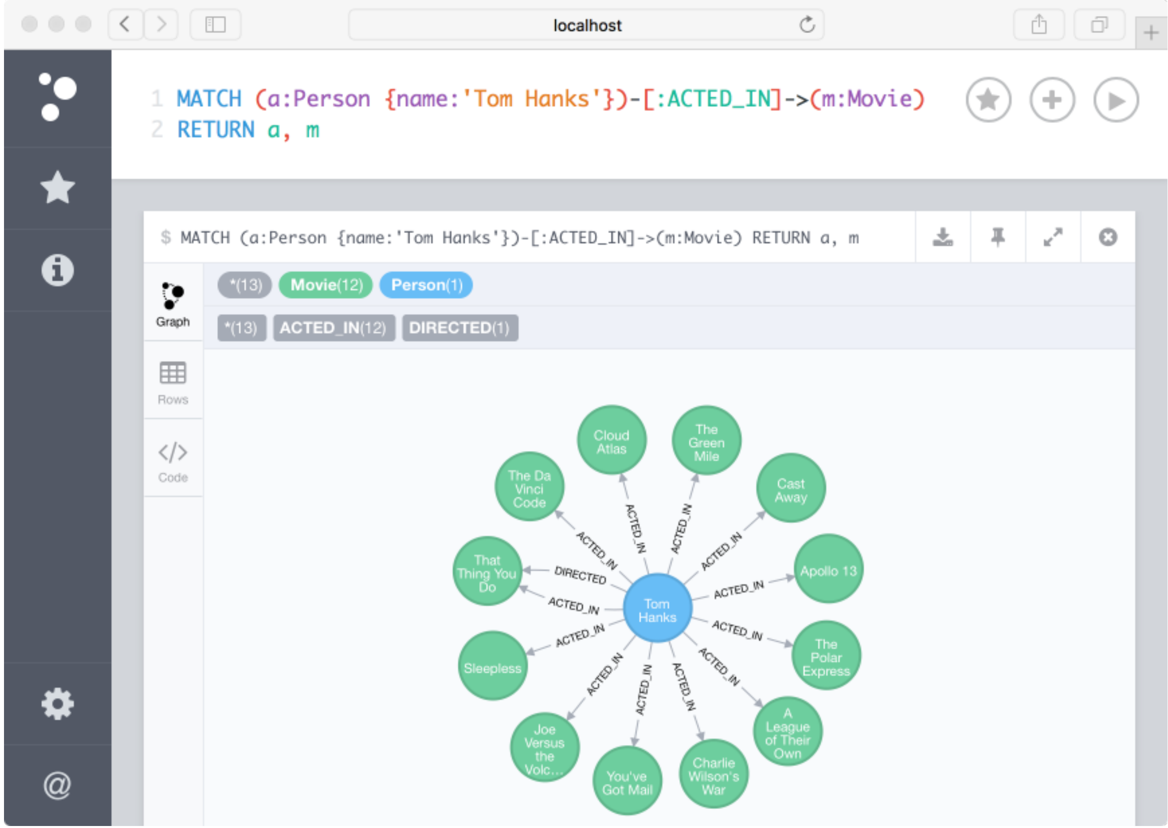
Katz centrality [https://en.wikipedia.org/wiki/Katz_centrality]

- Find actors with top 10 Katz centrality along ACTED_IN edges
  - Distance penalty is reciprocal of path length (e.g. 3-hop neighbor gets a penalty of 1/3)
  - Return actor vertex and Katz centrality

Exercise

HINTS

- Use the neo4j browser (web frontend) http://neo4j.com/docs/stable/tools-webadmin.html
- Use the Cypher documentation: http://neo4j.com/docs/stable/cypher-query-lang.html
- Use your preferred search engine
- Try out! Explorer! Have fun!!!



```
MATCH (a:Person {name:'Tom Hanks'})-[:ACTED_IN]->(m:Movie)
RETURN a, m
```

```
$ MATCH (a:Person {name:'Tom Hanks'})-[:ACTED_IN]->(m:Movie) RETURN a, m
```