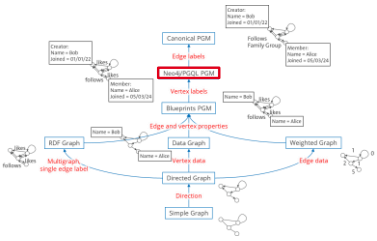
Summary

Quiz

- Architecture of Database Systems
- Transaction Management
- Modern Database Technology
- Data Warehouses and OLAP
- Data Mining
- Big Data Analytics

Graph Data Models

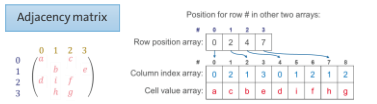


Property Graphs with Neo4j

```
(wb:Person { name: 'William Blake', yearOfBirth: 1757 })
(lo:City {name: 'London'})
(wb)-[:livedIn {since: 1757}]->(lo)
```

Graph Representations

Adjacency matrix



Adjacency lists

...without edge labels

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

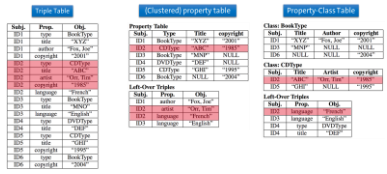
...with edge labels

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)])

Relational Representation



PGQs in SQL

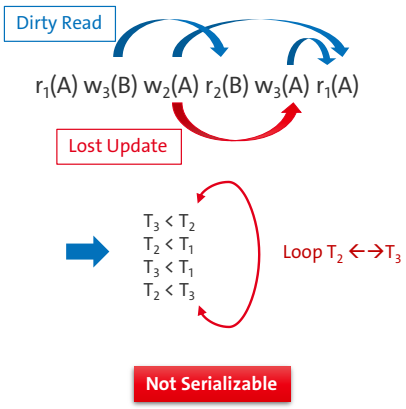
```
CREATE PROPERTY GRAPH bank_sql_pg
    VERTEX TABLES (
        bank_accounts
        KEY (id)
        LABEL account
        PROPERTIES (owner, balance)
    )
    EDGE TABLES (
        bank_txns
        KEY (txn_id)
        SOURCE KEY (from_acct_id) REFERENCES bank_accounts (id)
        DESTINATION KEY (to_acct_id) REFERENCES bank_accounts (id)
        LABEL transfer
        PROPERTIES (amount, reference)
    );
```

```
SELECT * FROM GRAPH_TABLE (bank_sql_pg MATCH
 (a IS account WHERE a.id = 42) -[e IS transfer]-> (b IS account)
COLUMNS (a.id AS acc_a, e.amount AS amount, b.id AS acc_b)
);
```

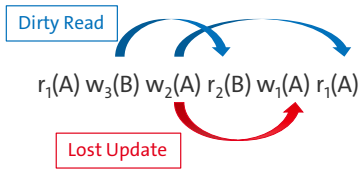
ACC_A	AMOUNT	ACC_B
42	15.00	15
42	5.00	15

Which of the following schedules are serializable?

Quiz Recap

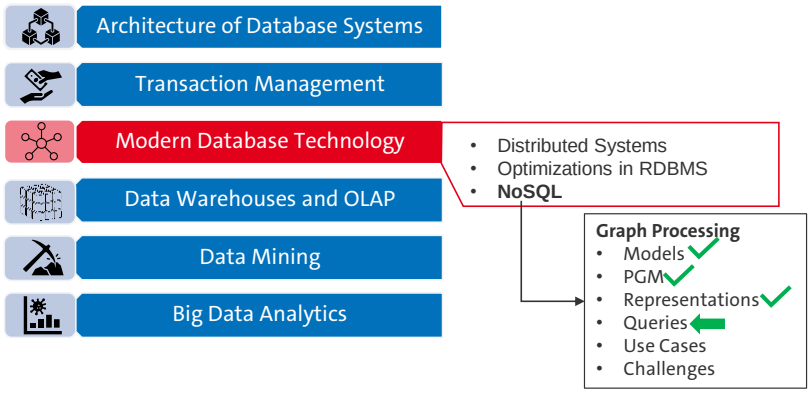


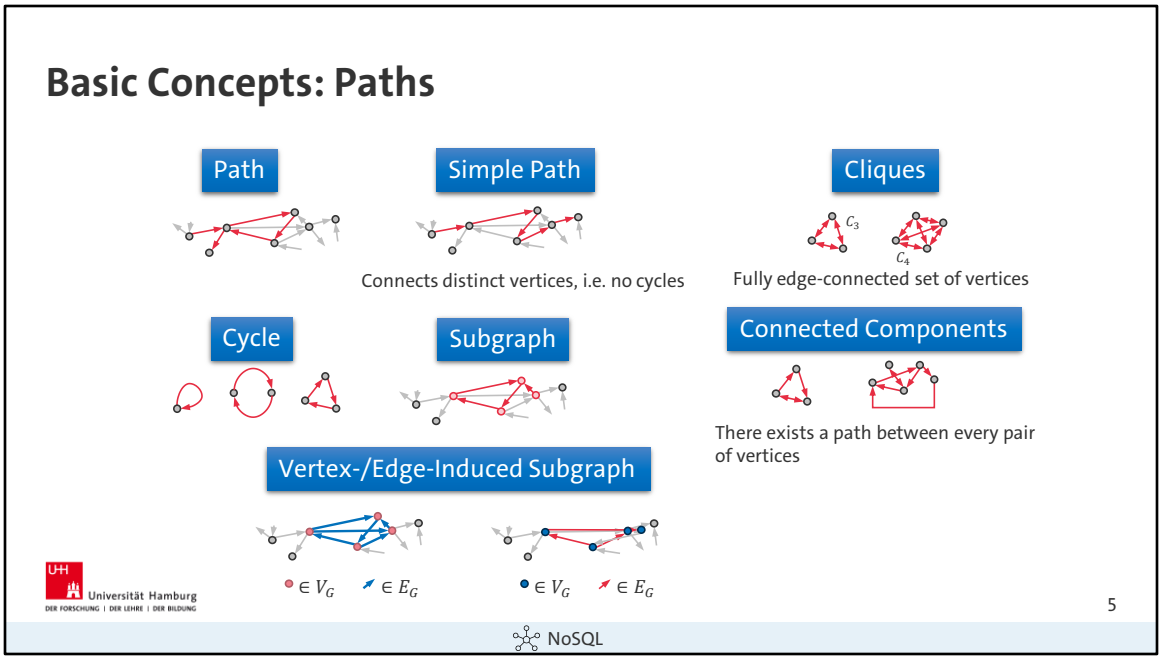
Not Serializable



Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Course Outline





Path
- 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 $e_i = (v_i, v_{i+1}) \in E, 1 \leq i < n$
- n is the length of the path

Simple Path
- $p_{v_1,v_n} = v_1 e_1 v_2 \dots v_{n-1} e_{n-1} v_n$
  with $\forall v_i, v_j: i \neq j \rightarrow v_i \neq v_j$ and $e_i = (v_i, v_{i+1}) \in E, 1 \leq i < n$

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

Subgraph
- A graph whose vertices and edges are fully part of a larger graph is a subgraph of that larger graph
- $G(V_G, E_G)$ is a subgraph of $H(V_H, E_H)$ if $V_G \subseteq V_H$ and $E_G \subseteq E_H$
- G is a subgraph of H denoted as $G \subseteq H$

Vertex-/Edge-Induced Subgraph
- Subgraph defined by a subset of vertices / edges of a given graphs
- A subgraph of $H(V_H, E_H)$ induced by
  vertex set $V_G \subseteq V_H$ / $E_G \subseteq E_H$
  is a graph $G(V_G, E_G)$ with $E_G = \{(x, y) \in E_H | x, y \in V_G\}$ / $V_G = \{v \in V_H | (v, \cdot) \in E_G \wedge (\cdot, v) \in E_G\}$

Cliques
- Fully edge-connected set of vertices
- $C \subseteq V$ with $\forall v_i, v_j \in C: (v_i, v_j) \in E$

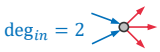
Connected Components
- Fully path-connected set of vertices, i.e. there exists a path between every pair of vertices

- $C \subseteq V$ with $\forall v_i, v_j \in C \; \exists p \in G: p = v_i, \dots, v_j$
- With considering direction: strongly connected
- Without considering direction: weakly connected
    - Directed edges are treated like undirected ones

Basic Concepts: Measures

Degree (Valency)

Degree of a vertex *v*



$\deg(v) = \deg_{out}(v) + \deg_{in}(v)$

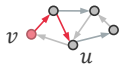
Degree of a graph *G*

$\deg(G) = \max_{v \in V} \deg(v)$

Degree distribution

→ Probability distribution of vertex degree in a graph

Distance



→ Number of edges in a shortest path connecting two vertices

Eccentricity

→ Longest shortest path starting from v

Average Distance

→ Average shortest path
→ Average eccentricity of any vertex in the graph

Radius

→ Minimum eccentricity of any vertex in the graph

$r(G) = 2$

Diameter

→ Maximum eccentricity of any vertex in the graph

$d(G) = 4$

Degree (Valency)

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

Distance

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

Eccentricity

- Greatest distance of vertex to any other vertex
- $\epsilon(v) = \max_{u \in V} d(v,u)$

Radius

- $r(G) = \min_{v \in V} \epsilon(v) = \min_{v \in V} \left(\max_{u \in V} d(v,u)\right)$

Average distance

- $L(G) = \frac{\sum_{v \in V} \epsilon(v)}{|V|}$

Diameter

- $d(G) = \max_{v \in V} \epsilon(v) = \max_{v \in V} \left(\max_{u \in V} d(v,u)\right)$

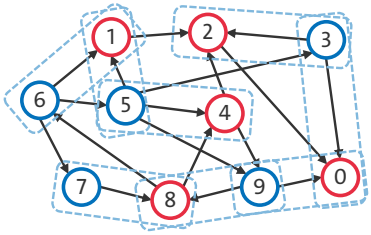
Queries: Graph Pattern Matching

Subgraph pattern p

Graph with place holders x and y

Matching p on G

Finds all subgraphs in G that fit to p



Intuitively, how many matches does p have on G?

Number of Results	
A	5
B	8
C	10
D	14

?

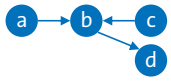
Pattern Syntax in neo4j: Vertex Pattern

- () 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*

These lists are not exhaustive, refer to the manual for more options.
http://neo4j.com/docs/developer-manual/current/cypher/syntax/patterns/

Pattern Syntax in neo4j : Edge Pattern

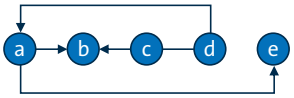
• --	unidentified undirected edge (matches either direction)
• -->	unidentified directed edge
• -[role]->	directed edge identified by variable *role*
• -[:ACTED_IN]->	unidentified directed edge with label *ACTED_IN*
• -[role:ACTED_IN]->	directed edge with label *ACTED_IN* identified by variable *role*
• -[role:ACTED_IN {roles: ["Neo"]}]->	+ property *roles* contains the string “Neo”
• (a)-->(b)<--(c), (b)-->(d)	branches



Pattern Syntax: Path pattern and Graph pattern

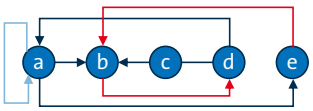
Path patterns

- (a)-->(b)<--(c)--(d)-->(a)-->(e)
- (keanu:Person:Actor {name: "Keanu Reeves"})
  -[role:ACTED_IN {roles: ["Neo"]}]-> (matrix:Movie {title: "The Matrix"})



Graph patterns

- Should have at least one shared variable
- (a)-->(b)<--(c)--(d)-->(a)-->(e), (e)-->(b)-->(d), (a)-->(a)



Path patterns

- String of alternating vertex pattern and edge pattern
- Starting and ending with a vertex pattern

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

Exercise: Pattern Syntax I

Which of these are valid patterns in Neo4j?

A: `(a,b:Movie)-[:SHOWN_IN]->(e),(f)`

B: `(a:Movie)-[:SHOWN_IN]->()`

C: `(:Movie)-[:SHOWN_IN]->`

D: `()<--(a:Movie)`

Exercise: Pattern Syntax II

Which pattern specifies a loop?

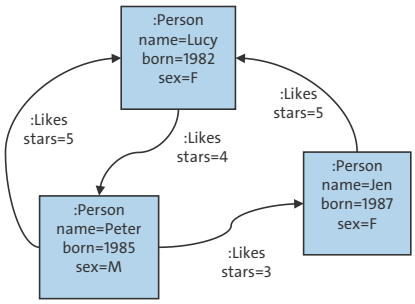
A: `(a:Movie {name: "Matrix"})-->(a)`

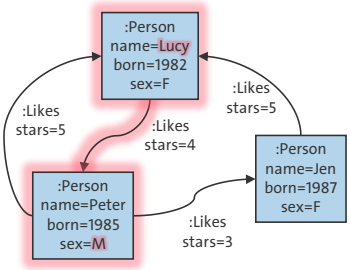
B: `(a:Movie {name: "Matrix"})-->(b:Movie {name: "Matrix"})`

C: `(a:Movie {name: "Matrix"})-->(a:Movie {name: "Matrix"})`

D: `(a:Movie {name: "Matrix"})-->({name: "Matrix"})`

Query Syntax: Matching






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

Number of Results	
A	2
B	3
C	4
D	5

?

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

15

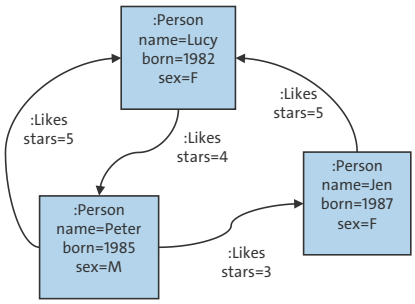
NoSQL

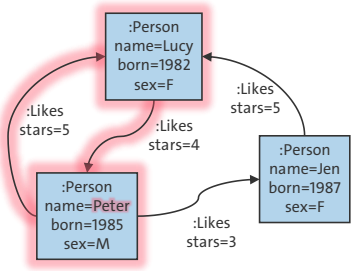
Documentation:
http://neo4j.com/docs/developer-manual/current/cypher/clauses/match/
http://neo4j.com/docs/developer-manual/current/cypher/clauses/return/

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

Query Syntax: Matching






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

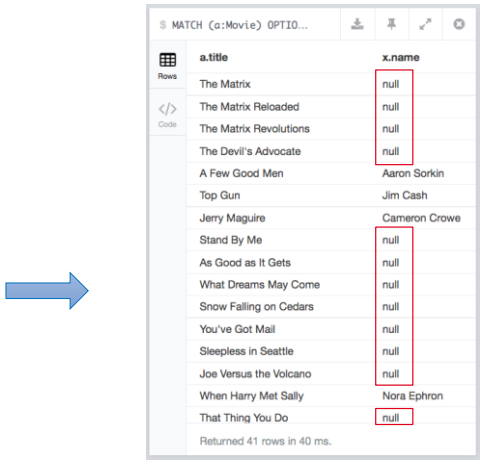
Number of Results	
A	2
B	3
C	4
D	5

?

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

Query Syntax: Optional Match

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

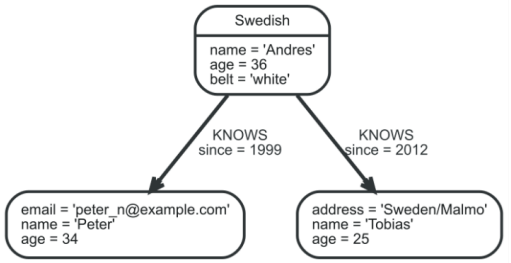


Documentation:
http://neo4j.com/docs/developer-manual/current/cypher/clauses/optional-match/

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

Query Syntax: Filtering




```
MATCH (n)
WHERE n.name = 'Peter' XOR (n.age < 30 AND n.name = "Tobias")
OR NOT (n.name = "Tobias" OR  n.name="Peter")
RETURN n
```

n
Node[0]{name:"Andres",age:36,belt:"white"}
Node[1]{address:"Sweden/Malmo",name:"Tobias",age:25}
Node[2]{email:"peter_n@example.com",name:"Peter",age:34}

18

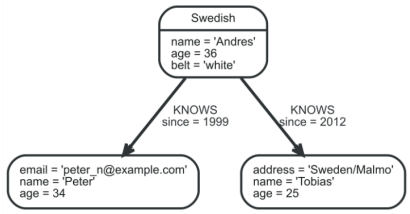
NoSQL

Documentation:
http://neo4j.com/docs/developer-manual/current/cypher/clauses/where/

WHERE clause

- 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>

Query Syntax: Filtering



- Filter on node label

```
MATCH (n) WHERE n:Swedish RETURN n
```

n
Node[0]{name:"Andres",age:36,belt:"white"}

- ... on node property

```
MATCH (n) WHERE n.age < 30 RETURN n
```

n
Node[1]{address:"Sweden/Malmo",name:"Tobias",age:25}

- ... on relationship type

```
MATCH (n)-[k]->(f) WHERE type(k) = 'KNOWS'
                     AND k.since < 2000 RETURN f
```

n
Node[2]{email:"peter_n@example.com",name:"Peter",age:34}

NoSQL

More filtering options

On Lists

```
MATCH (n) WHERE a.name IN ["Peter", "Tobias"] RETURN n
```

n

Node[1]{address:"Sweden/Malmo",name:"Tobias",age:25}
Node[2]{email:"peter_n@example.com",name:"Peter",age:34}

On string properties

```
MATCH (n) WHERE n.name = 'Peter' RETURN n
```

- prefixes
- suffixes
- infixes
- regular expressions

```
MATCH (n) WHERE n.name STARTS WITH 'Pet' RETURN n
MATCH (n) WHERE n.name ENDS WITH 'ter' RETURN n
MATCH (n) WHERE n.name CONTAINS 'ete' RETURN n
MATCH (n) WHERE n.name =~ 'P[et]+r?' RETURN n
```

n

Node[2]{email:"peter_n@example.com",name:"Peter",age:34}

On property (non-)existence

`false` is default for missing values

```
MATCH (n) WHERE exists(n.belt) RETURN n
MATCH (n) WHERE NOT n.belt = 'white' RETURN n
```

n

Node[0]{name:"Andres",age:36,belt:"white"}

```
MATCH (n) WHERE NOT exists(n.belt) RETURN n
MATCH (n) WHERE n.belt IS NULL RETURN n
```

n

Node[1]{address:"Sweden/Malmo",name:"Tobias",age:25}

Using patterns for filtering

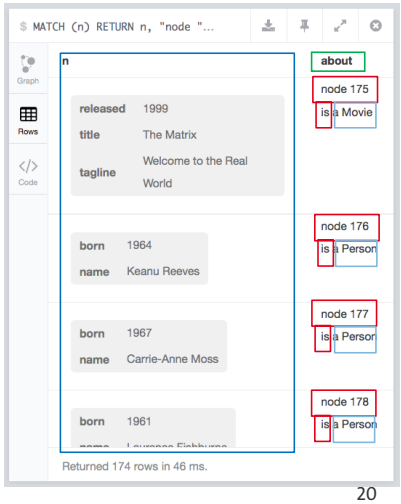
```
MATCH (tobias { name: 'Tobias' }),(others)
WHERE others.age > 30 AND (tobias)<--(others)
RETURN others
```

n

Node[0]{name:"Andres",age:36,belt:"white"}

Query Syntax: Projection

```
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
```



Documentation:
http://neo4j.com/docs/developer-manual/current/cypher/clauses/return/
http://neo4j.com/docs/developer-manual/current/cypher/syntax/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>

More clauses adapted from SQL

- ORDER BY, e.g. `MATCH (n) RETURN n ORDER BY n.name DESC`
- LIMIT, e.g. `MATCH (n) RETURN n ORDER BY n.name LIMIT 3`
- SKIP, e.g. `MATCH (n) RETURN n ORDER BY n.name SKIP 3`
- `Unique results, e.g. MATCH (a { name: "A" })-->(b) RETURN DISTINCT b`
- `Aggregations, e.g. MATCH (n) RETURN count(*)`

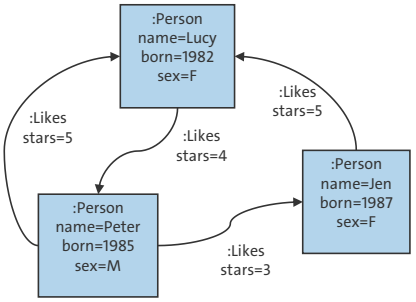
Exercise @home: neo4j Queries

a) Find all pairs of people who like each other. Return their names.

b) Return all names in the graph that are not “Peter”. Order them alphabetically.

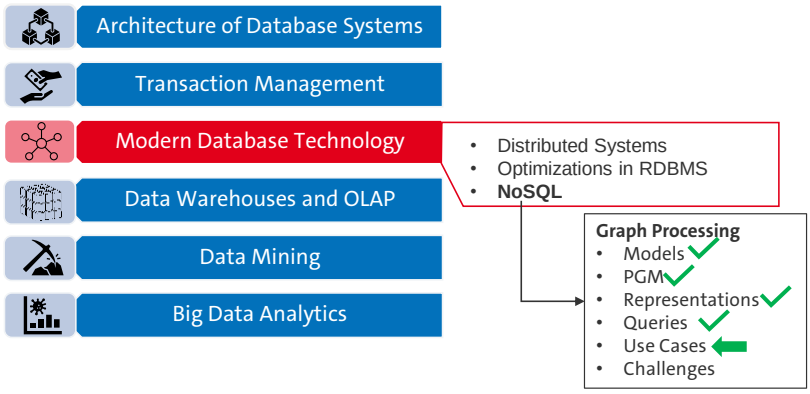
c) Return the highest star rating of a “LIKES” relationship in the graph.

d) Return the name and birth year of all people who like Lucy. The result should have the columns “GivenName” and “year”.



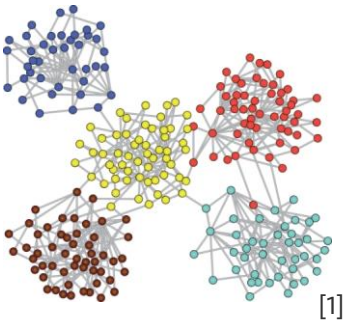
Solution: White on white

Course Outline

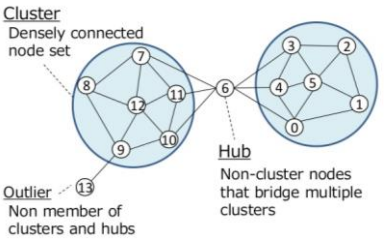


Use Cases: Clustering

- Many different approaches, e.g. min-max cut methods
- Structure-connected clusters → Extends notion of a density-based cluster to graph
- Algorithm examples: SCAN (original), LinkSCAN (uses edge sample), SCAN++ (more efficient)



[1]



[2]

Clustering
Division of a graph into a set of subgraphs such that there are a dense set of edges within every cluster and few edges between clusters

Sources:
[1] https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2542420/
[2] https://www.slideshare.net/LazyShion/scan-efficient-algorithm-for-finding-clusters-hubs-and-outliers-on-largescale-graphs-vldb-2015

The SCAN paper: Xu et al. SCAN: A Structural Clustering Algorithm for Networks. KDD. 2007

A very simplified description of SCAN

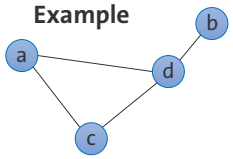
- Neigborhood of a vertex: Each vertex that is connected via an edge and the vertex itself
- Distance (specific to SCAN and different from our general definition on slide 6): Intersection of the neigborhoods of two vertices
- Normalized distance: Distance normalized by the geometric mean of the size of the two neighborhoods

$$\sigma(v,w) = \frac{|\Gamma(v) \cap \Gamma(w)|}{\sqrt{|\Gamma(v)| \cdot |\Gamma(w)|}}$$

Normalized distance between v and w
Neighborhood intersection between v and w
Sizes of the neighborhoods

- Structural Similarity: If $\sigma(v,w) > \epsilon$, v and w are considered structurally similar
- Core: Vertices with more than μ structural similar neighbors are core vertices

Example



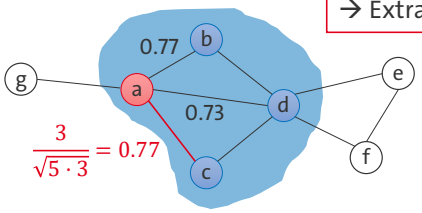
$\sigma(c,d) = \frac{3}{\sqrt{3 \cdot 4}} = 0.866$

- If a core is found, the cluster is expanded
- All structurally similar vertices that are reachable from a core, are initially a border
- If a vertex is not part of a cluster and all it's neighbors either belong to one or no cluster, it is an outlier
- If a vertex is not part of a cluster but has neighbors in two or more clusters, it is a hub

SCAN

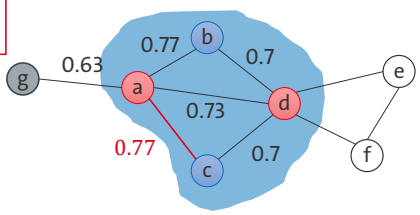
Example: $\epsilon = 0.7, \mu = 2$

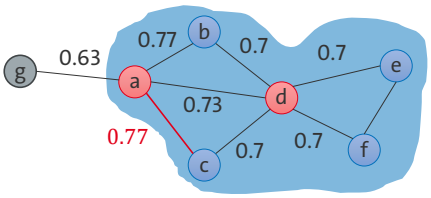
Node a is a core
b, c, and d are borders
→ Extract a cluster



$\frac{3}{\sqrt{5 \cdot 3}} = 0.77$

g is an outlier
d is a core





d is a core → Expand cluster
b, c, e, and f are borders

Exercise Clustering

$\epsilon = 0.8, \mu = 2$