TECHNISCHE UNIVERSITÄT DRESDEN

Dresden Database Systems Group

Virtual Graphs & Graph Views in Cypher

Sascha Peukert[1], Hannes Voigt[1], Michael Hunger[2]

[1]TU Dresden

[2]Neo Technology

Concept Chasm

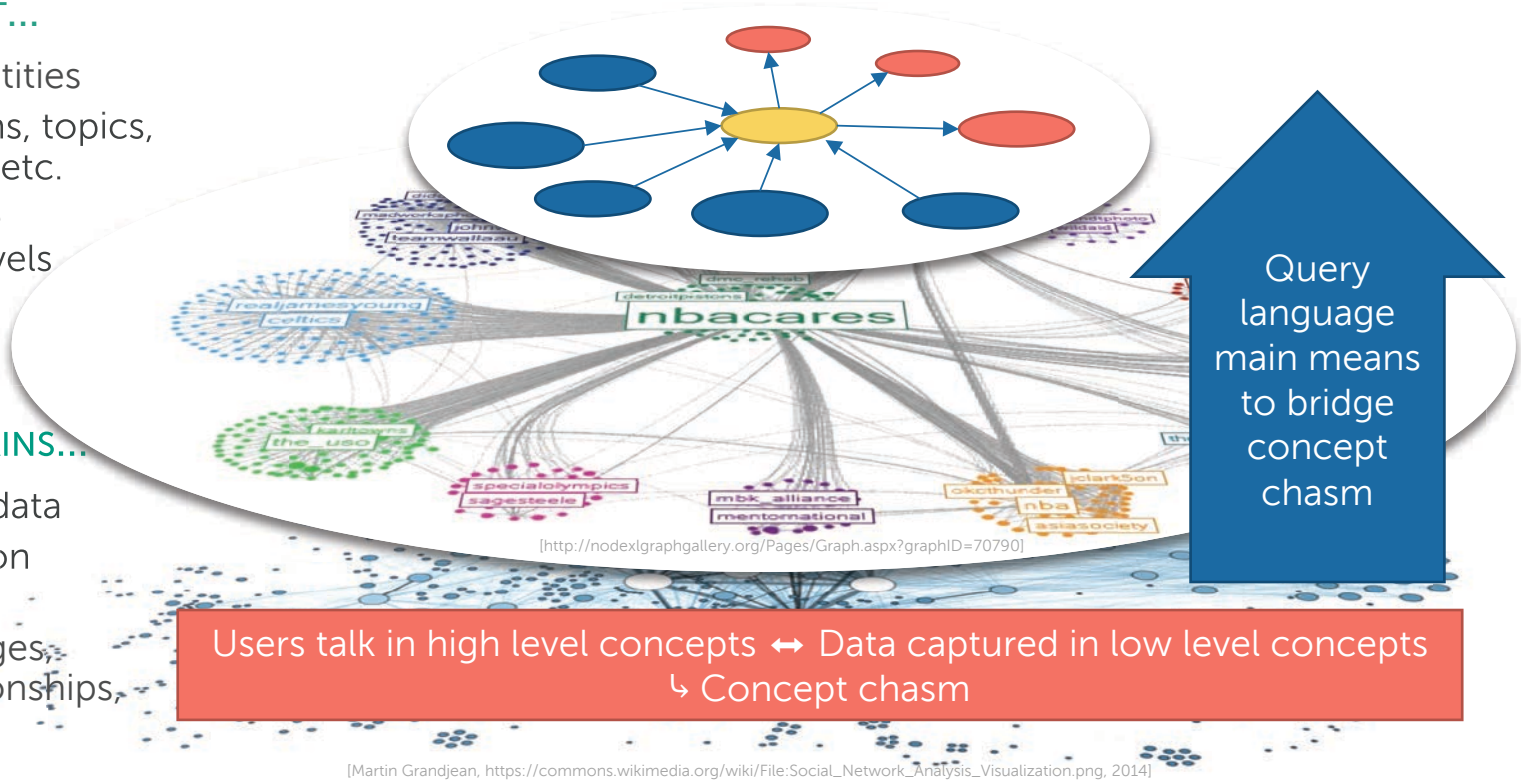
Dresden Database Systems Group

USERS TALK ABOUT...

- Application entities
- e.g. discussions, topics, communities, etc.
- Likely multiple abstraction levels

BASE DATA CONTAINS...

- Fine granular data
- Low abstraction
- E.g. individual twitter messages, retweet relationships, etc.



Query language main means to bridge concept chasm

Users talk in high level concepts ↔ Data captured in low level concepts
↳ Concept chasm

[Martin Grandjean, https://commons.wikimedia.org/wiki/File:Social_Network_Analysis_Visualization.png, 2014]

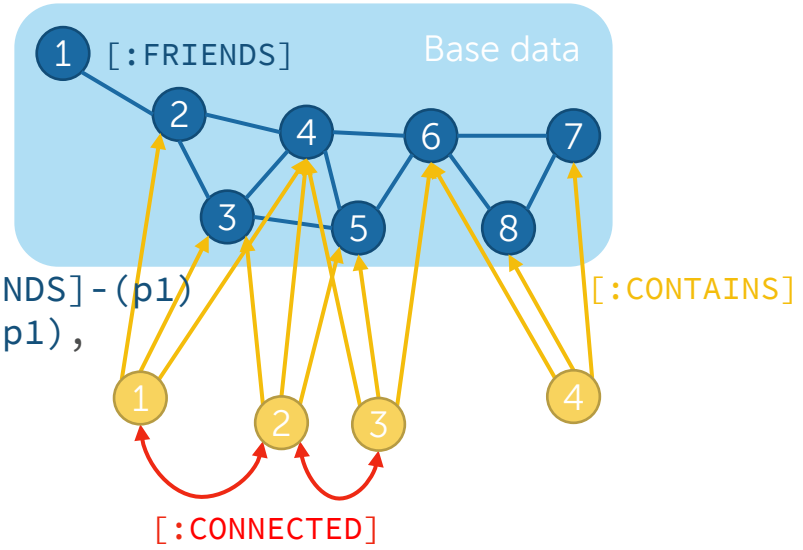
What do you need?

Dresden Database Systems Group

1. Let user create abstract graphs

- Nodes and relationships not present in the base data but derived from base data

```
MATCH (p1)-[:FRIENDS]-(p2)-[:FRIENDS]-(p3)-[:FRIENDS]-(p1)
CREATE VIRTUAL (t:FriendsTriangle)-[:CONTAINS]->(p1),
               (t)-[:CONTAINS]->(p2),
               (t)-[:CONTAINS]->(p3)
MATCH (t1)-[:CONTAINS]->()<-[:CONTAINS]-(t2)
          -[:CONTAINS]->()<-[:CONTAINS]-(t1)
CREATE VIRTUAL (t1)-[:CONNECTED]->(t2)
```



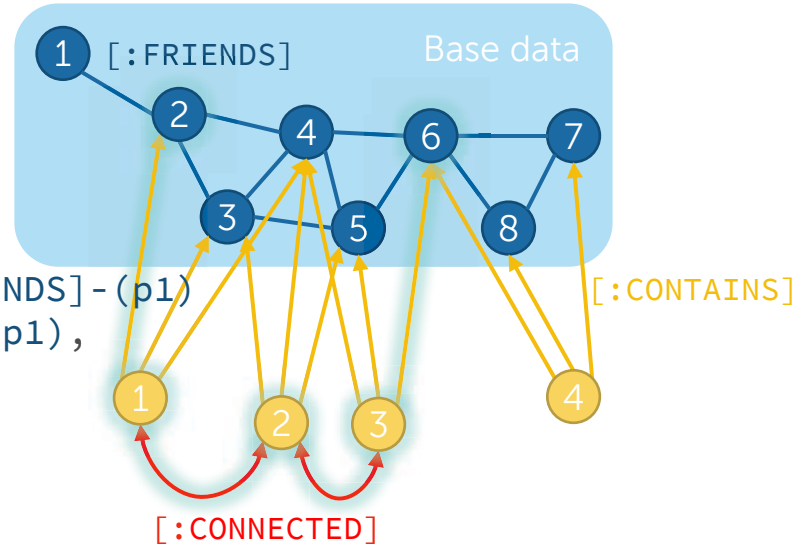
What do you need?

Dresden Database Systems Group

1. Let user create abstract graphs

- Nodes and relationships not present in the base data but derived from base data

```
MATCH (p1)-[:FRIENDS]-(p2)-[:FRIENDS]-(p3)-[:FRIENDS]-(p1)
CREATE VIRTUAL (t:FriendsTriangle)-[:CONTAINS]->(p1),
               (t)-[:CONTAINS]->(p2),
               (t)-[:CONTAINS]->(p3)
MATCH (t1)-[:CONTAINS]->()<-[:CONTAINS]-(t2)
          -[:CONTAINS]->()<-[:CONTAINS]-(t1)
CREATE VIRTUAL (t1)-[:CONTAINS]->(t2)
MATCH (pa)<-[:CONTAINS]-(ta),
      tp=shortestPath((ta)-[:CONNECTED*]->(tb)),
      (tb)-[:CONTAINS]->(pb)
RETURN pa, pb, length(tp)+1 AS triangleDist
```



pa	pb	triangleDist
2	6	3
⋮	⋮	⋮

What do you need?

Dresden Database Systems Group

2. Let user modularize and reuse (sub)-queries with views

- Create view

```
CREATE VIEW friendsTriangles {
  MATCH (p1)-[:FRIENDS]-(p2)-[:FRIENDS]-(p3)-[:FRIENDS]-(p1)
  CREATE VIRTUAL (t:FriendsTriangle)-[e1:CONTAINS]->(p1),
                 (t)-[e2:CONTAINS]->(p2),
                 (t)-[e3:CONTAINS]->(p3)
  SAVE t,e1,e2,e3,p1,p2,p3
}
```

- Use view

```
IN VIEW friendsTriangles { MATCH (p)<-[:CONTAINS]-(t) }
RETURN p, count(t) AS numTriangles
ORDER BY numTriangles DESC LIMIT 10
```

- Use multiple views

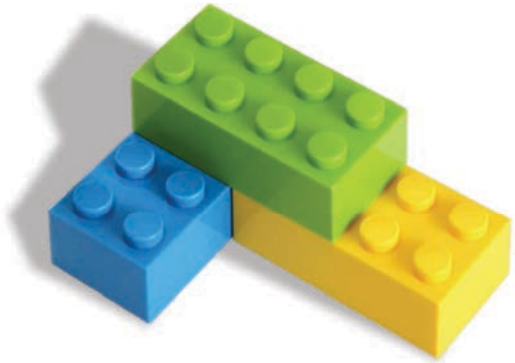
```
IN VIEW friendsTriangles, connected, ALL {  //ALL is base data
  MATCH (t1)-[:CONTAINS]->(p)<-[:CONTAINS]-(t2)
        -[:CONTAINS]->(q1)-[:KNOWS]-(q2)<-[:CONTAINS]-(t2)
  WHERE NOT (t1)-[:CONNECTED]-(t2)
}
RETURN t1,t2,q1,q2
```

Summary

Dresden Database Systems Group

Views are awesome for

- Query modularization
- Reuse of query concepts
- Employing *Need to Know* for applications (fine-grained access control)
- Performance tuning (with materialized views)



Our current implementation in Neo4j

- Two specializations of the OperationsFacade for handling virtual entities and views
- VirtualEntitiesFacade create entities internally and discards them after end of transaction
- ViewFacade
  - Keeps list of node and relationship id that are part of the view
  - Filter queries on views are execution on base data for results that are part of the view

Goal: True cypher query rewrite