Inhaltsverzeichnis: Graphdatenbanken

Anfragesprachen

Imperative Sprache

- Neo4j Core API

```
for (Relationship rel : d.getRelationships(Direction.INCOMING, COMPANION_OF)) {
        Node n = rel.getStartNode();
        if (n.hasRelationship(Direction.OUTGOING, IS_A)) {
                Relationship r = n.getSingleRelationship(IS_A, Direction.OUTGOING);
                if (r.getEndNode().equals(human) ) { /* Found a human */  }}}
```

Deklarative Sprache

- SPARQL (RDF)
- **Cypher**

```
MATCH (d) <-[:COMPANION_OF]- (n) -[:IS_A]->
(human) RETURN n;
```

Cypher

- Deklarative, SQL-ähnliche und leicht verständliche Sprache zur Definition, Manipulation und Abfrage eines Graphen
- Frage nach bestimmten **Mustern**
- Beschreibung der Muster über ASCII-Art
- Beispiel: (emil) <-- (jim) --> (ian) --> (emil)
  - Knoten: ( )
  - Gerichtete Kante: -->
  - Bezeichner: emil, jim, ian
- Filter auf Kantenlabel: [:<Label>]

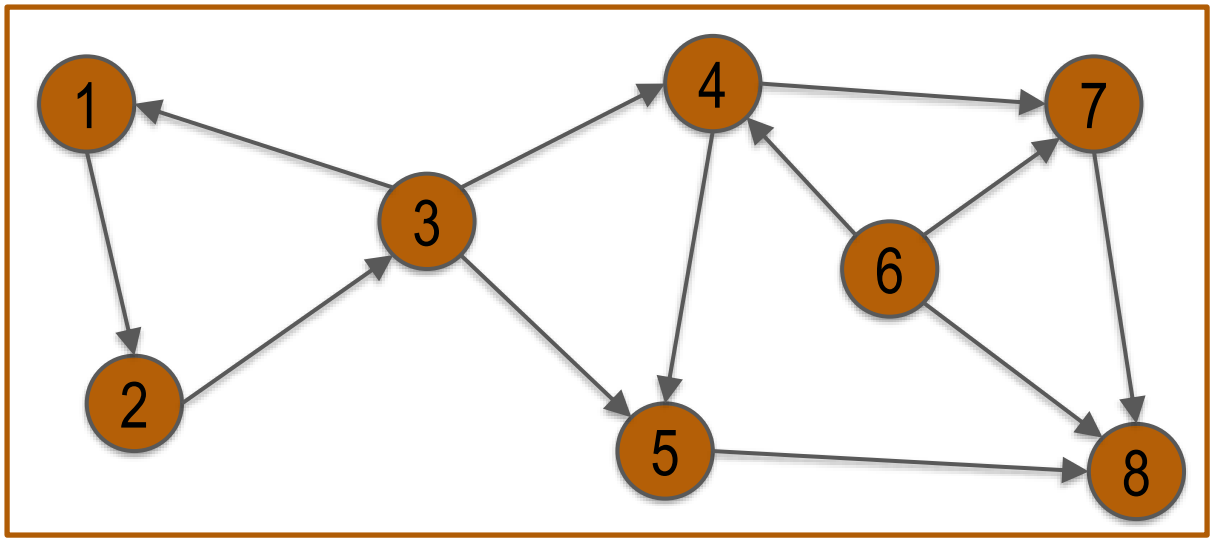
  (emil) <-[:KNOWS]- (jim) -[:KNOWS]-> (ian) -[:KNOWS]-> (emil)
- Filter auf Knotenlabel und -attribute:

  (emil:Person {name:'Emil'}) <-[:KNOWS]- (jim:Person {name:'Jim'}) -[:KNOWS]-> (ian:Person {name:'Ian'}) -[:KNOWS]-> (emil)

Cypher: Definition

- **CREATE:** Erzeugen von Knoten, Kanten, Properties und Labels

```
CREATE (n1:Node {name:1}), (n2:Node {name:2}),
       // ...
       (n1)-[:LINK]->(n2), (n2)-[:LINK]->(n3),
       // ...
```



- **MERGE:** Zusammenführen existierender und nicht existierender Teile eines Mustergraphen (ermöglicht Erzeugen einzigartiger Knoten/Kanten)

Cypher: Manipulation

- **SET:** Update von Properties und Labels
- **DELETE:** Entfernen von Knoten und Kanten
- **REMOVE:** Entfernen von Properties und Labels
- **WITH:** Verknüpfen mehrerer Anfragen (Pipe-Konzept)

Cypher: Anfragen

- **MATCH:** Beschreibung eines Mustergraphen

- **WHERE**
  - Filter auf Knotenattribute (Alternative zu „{name:'Emil'}“)
  - Bindung von Elementen im Mustergraphen an Instanzen

- **RETURN**
  - Auswahl der Ergebnisse (Knoten, Kanten, Attribute) (Projektion)
  - Anwenden von Gruppierung, Aggregation und weiterer Funktionen

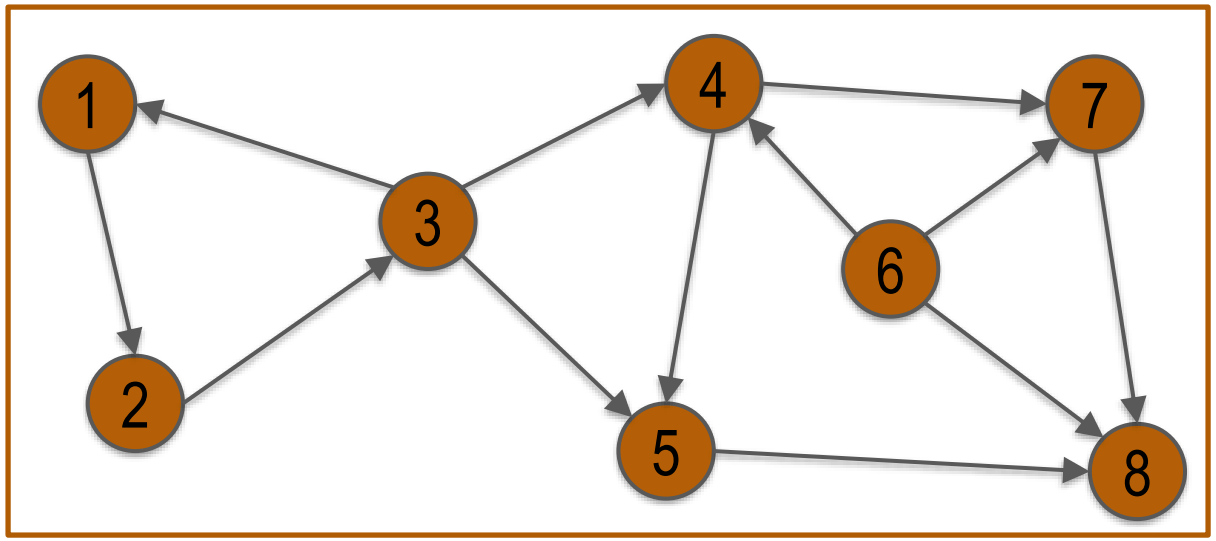
- **ORDER BY:** Sortierung

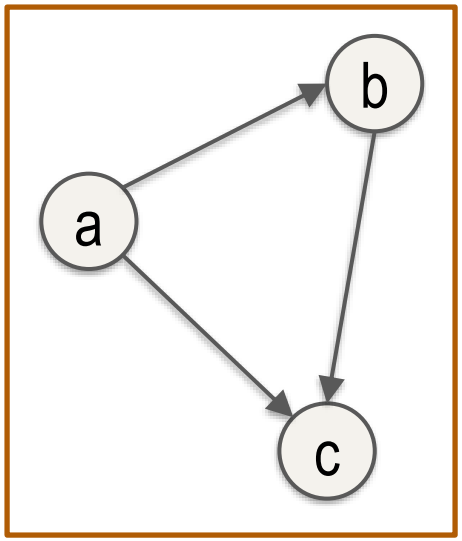
- **LIMIT, SKIP**

Cypher: Beispiel (1)

```
MATCH  (a)-->(b)-->(c)<--(a)
RETURN a.name, b.name, c.name
```

a.name	b.name	c.name
3	4	5
6	4	7
6	7	8



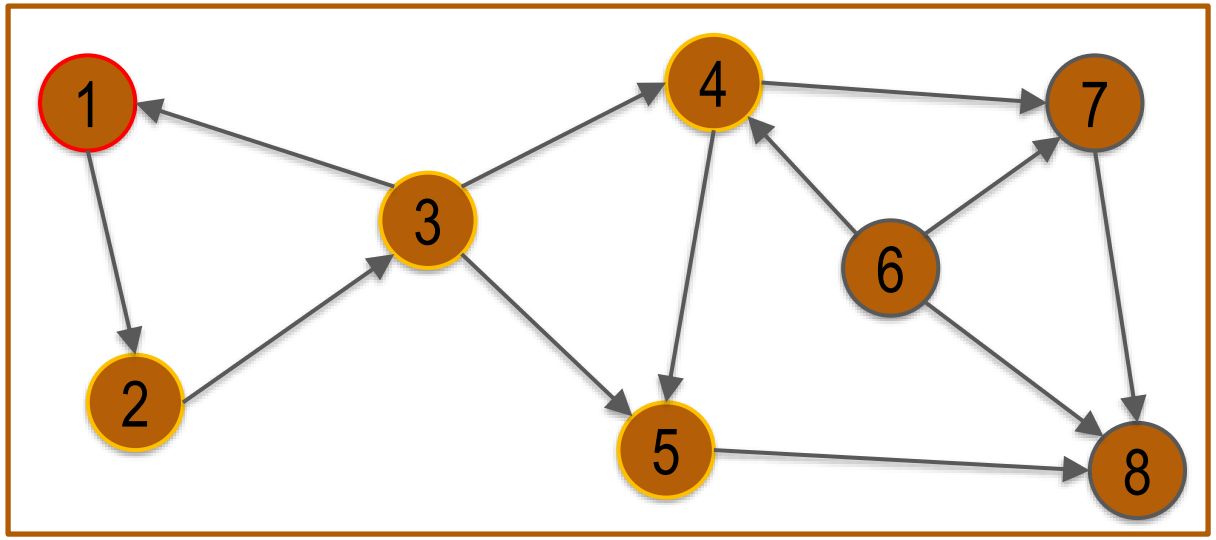


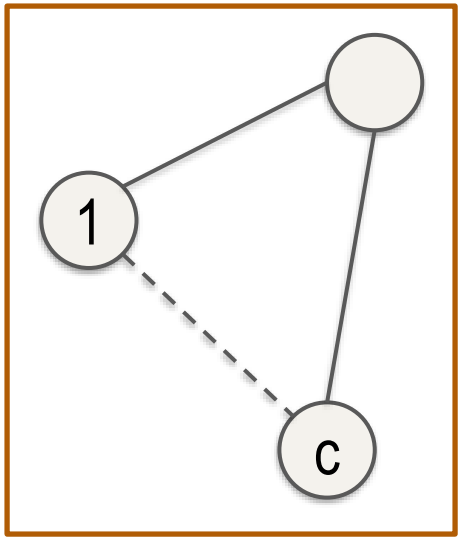
Cypher: Beispiel (2)

```
MATCH (a)--()--(c) WHERE a.name = 1
RETURN c.name AS c
```

```
MATCH (a {name:1})--()--(c)
RETURN c.name AS c
```

c
3
4
5
2

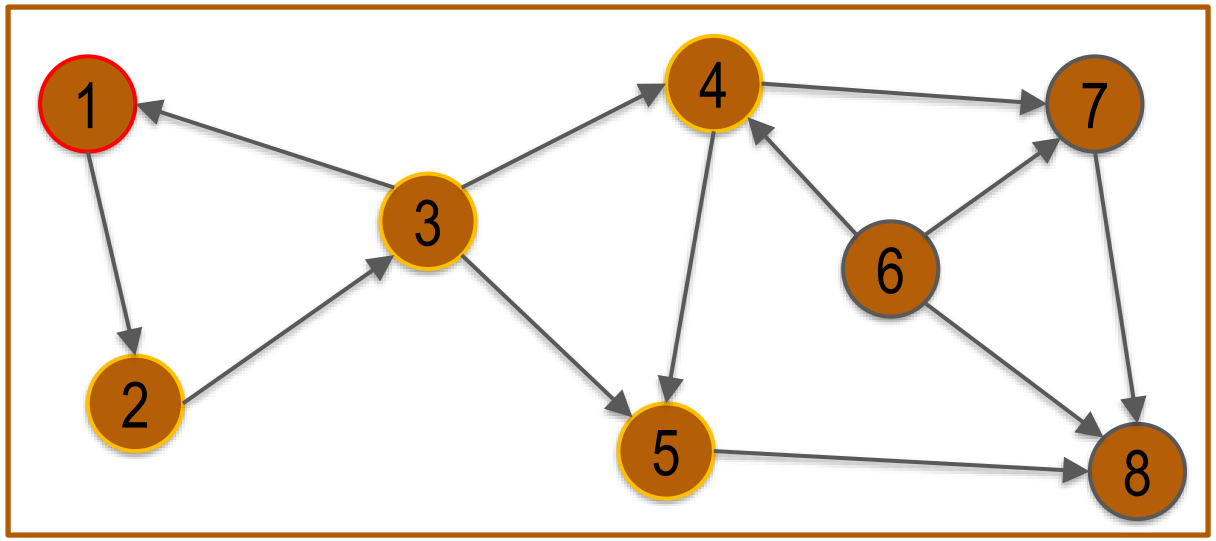


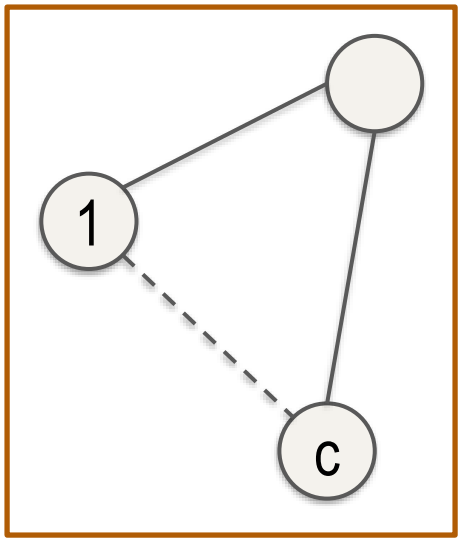


Cypher: Beispiel (3)

```
MATCH (a)--()--(c) WHERE a.name = 1
RETURN c.name AS c ORDER BY c DESC
```

c
5
4
3
2

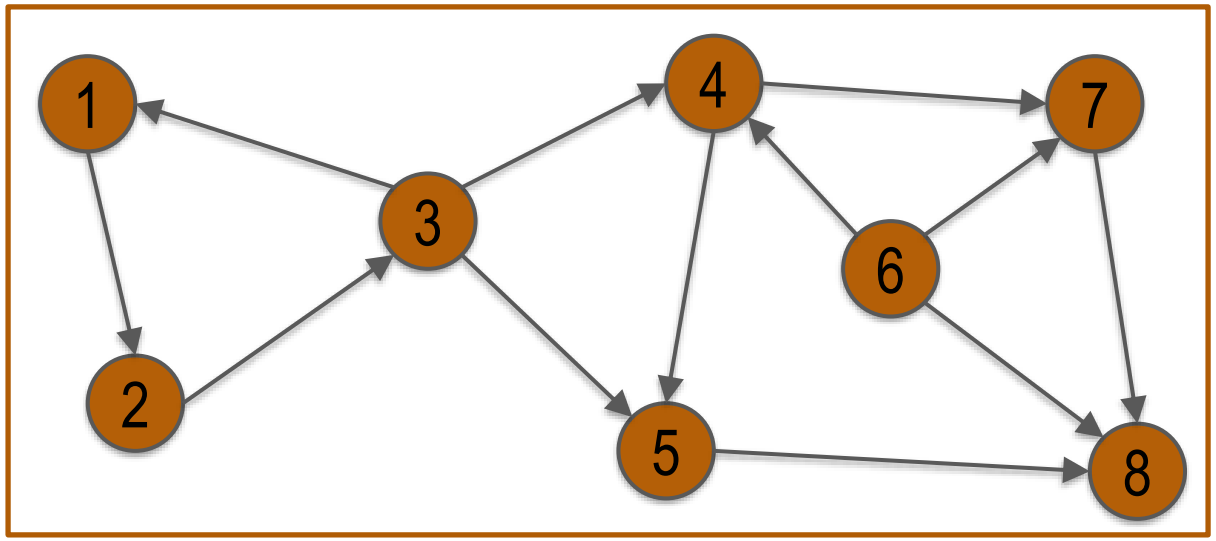


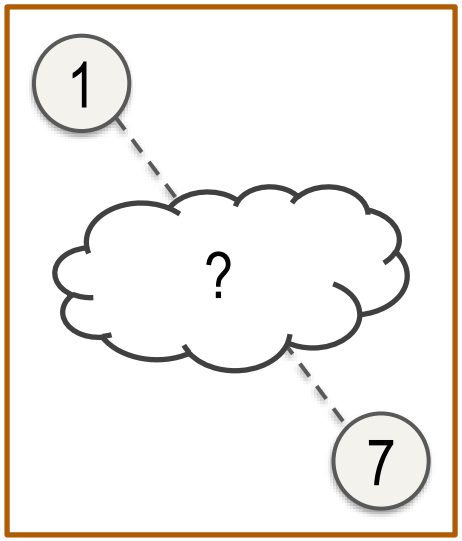


Cypher: Beispiel (4)

```
MATCH  p=(a)-[*1..4]-(b)
WHERE  a.name = 1 AND b.name = 7
RETURN nodes(p), length(p)
```

nodes(p)	length(p)
(1,2,3,4,7)	4
(1,3,4,7)	3
(1,3,4,6,7)	4
(1,3,5,8,7)	4
(1,3,5,4,7)	4

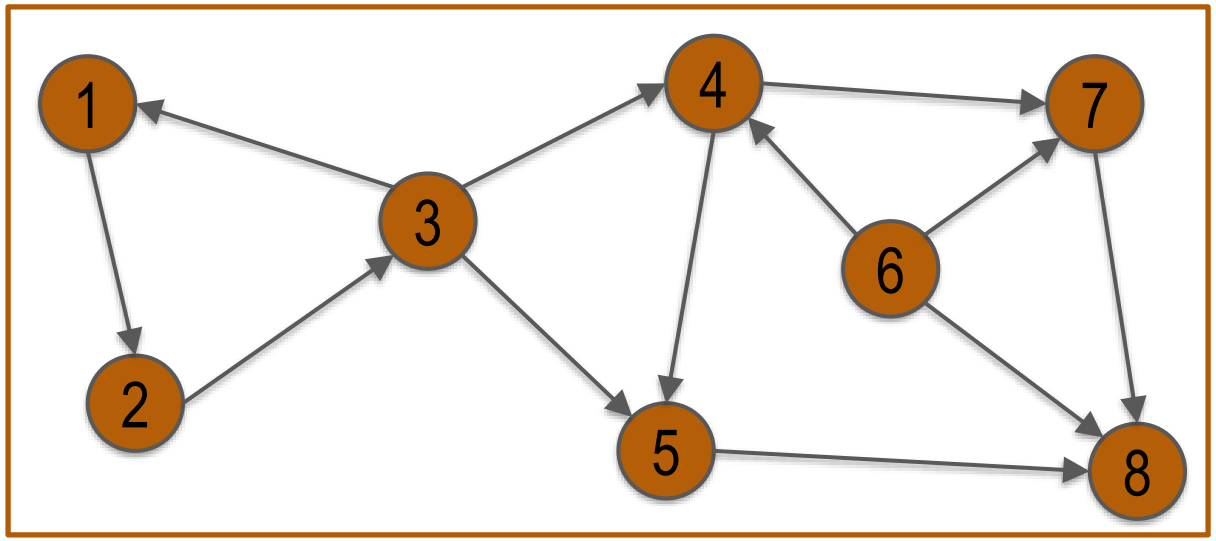


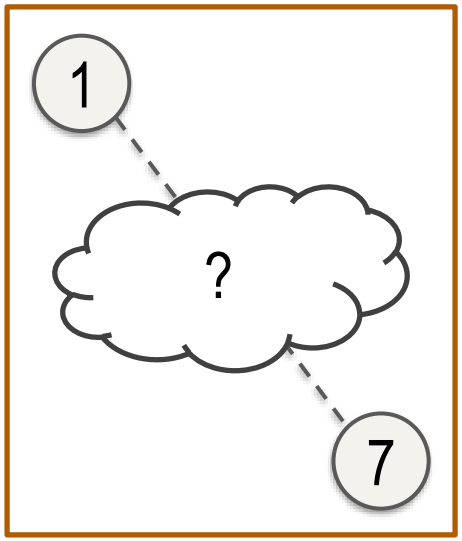


Cypher: Beispiel (5)

```
MATCH  p=(a)-[*1..4]-(b)
WHERE  a.name = 1 AND b.name = 7
RETURN length(p), count(p)
```

length(p)	count(p)
4	4
3	1

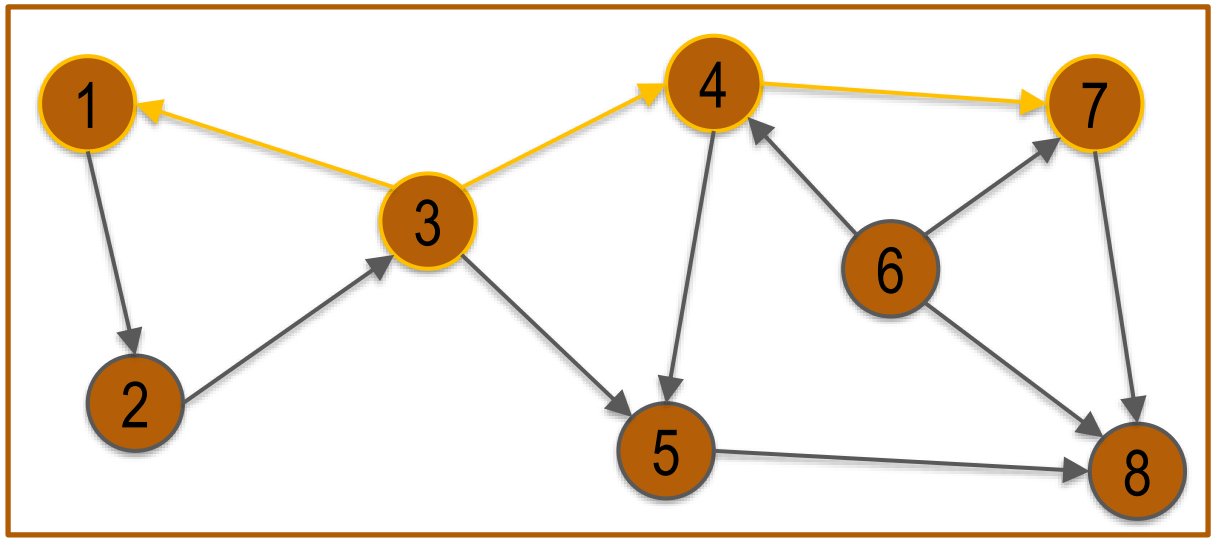


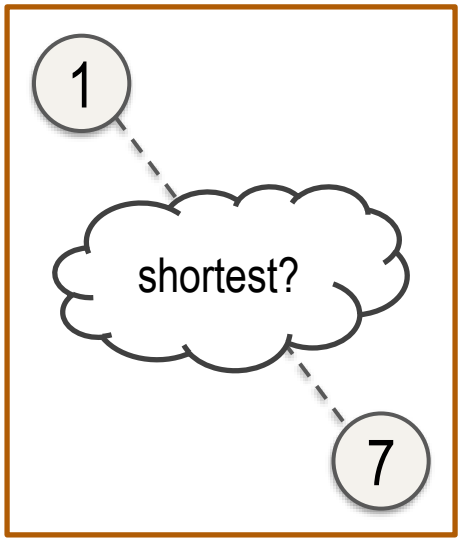


Cypher: Beispiel (6)

```
MATCH  p=shortestPath((a)-[*..4]-(b))
WHERE  a.name = 1 AND b.name = 7
RETURN nodes(p)
```

nodes(p)
(1,3,4,7)

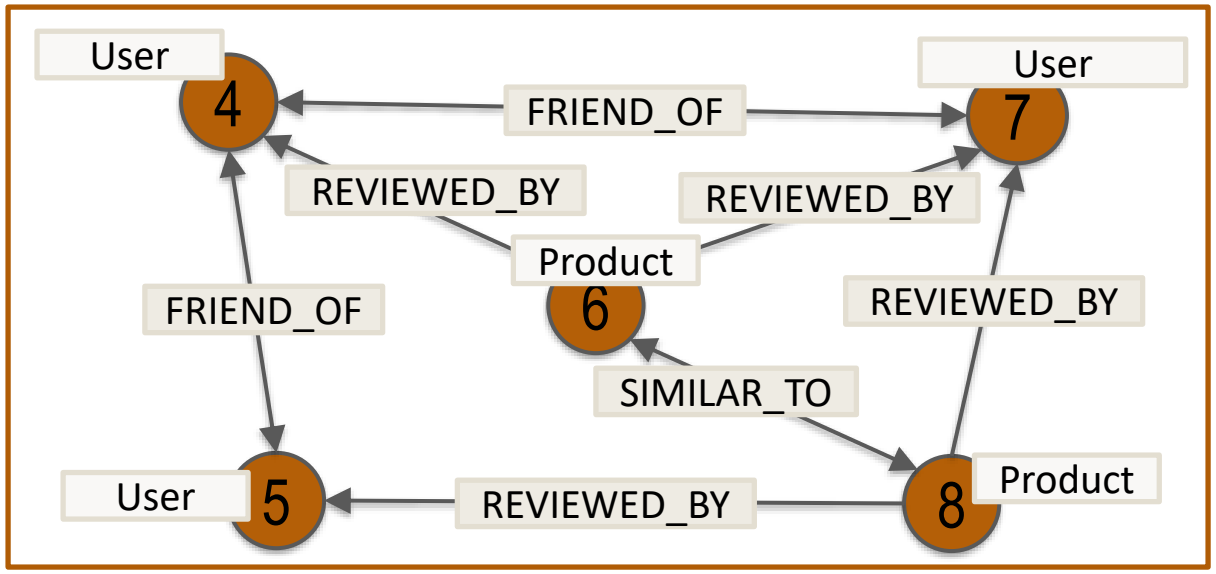


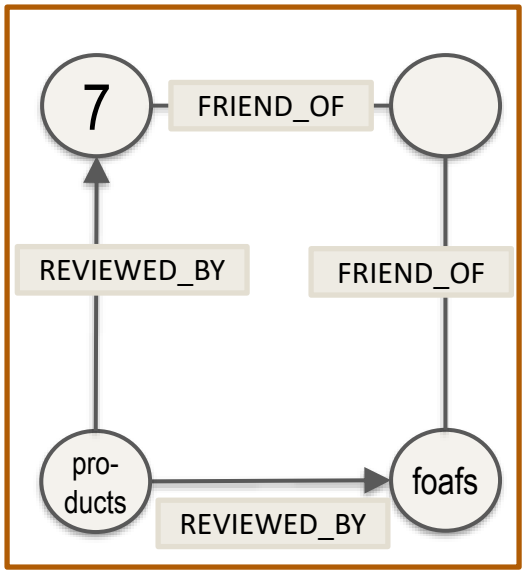


Cypher: Beispiel (7)

```
MATCH  (u:User)-[:FRIEND_OF*2..2]-(foaf),
       (foaf)<-[:REVIEWED_BY]-(product),
       (product)-[:REVIEWED_BY]->(u)
WHERE  u.name = 7 AND NOT (u)-[:FRIEND_OF]-(foaf)
RETURN foaf.name, product.name
```

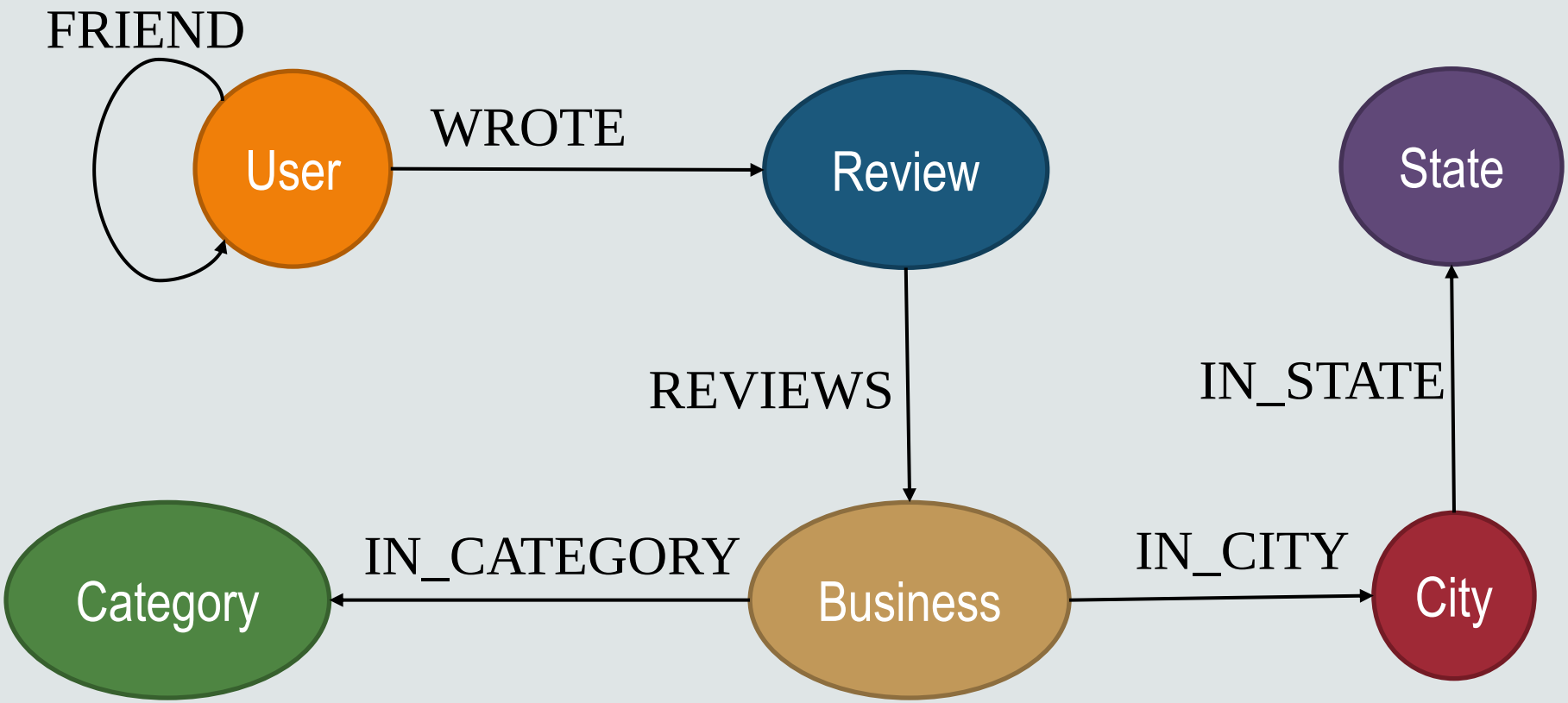
foafs.name	products.name
5	8





Übung

Schema



Übung

Formulieren Sie folgende Anfragen in Cypher:

1. Geben Sie die Namen (Attribut „name“) und Ids (Attribut „id“) aller Nutzer an, die über einen dritten Nutzer mit Jenny (id: vqd25k6Zx4mqYCCEFCT_Gw) befreundet sind!
2. Geben Sie die Anzahl der Unternehmen unterteilt nach den jeweiligen Staaten (Attribut „name“) an! Geben Sie zusätzlich die verschiedenen Städte (Attribut „name“) pro Staat an (Funktion *collect*)!
3. Geben Sie die Namen aller Unternehmen (Attribut „name“) an, die von einem Nutzer genauso bewertet wurden wie "Bill's Coffee Shop" (id: W80o5A-ajAznRWO3E5J7EQ)!
4. Geben Sie die durchschnittliche Bewertung (Attribut „stars“ von Review) aller Unternehmen an, welche der Kategorie 'Art Museums' (Attribut „name“) zugeordnet wurden!

Hinweis: https://neo4j.com/docs/cypher-refcard/current/

Übung

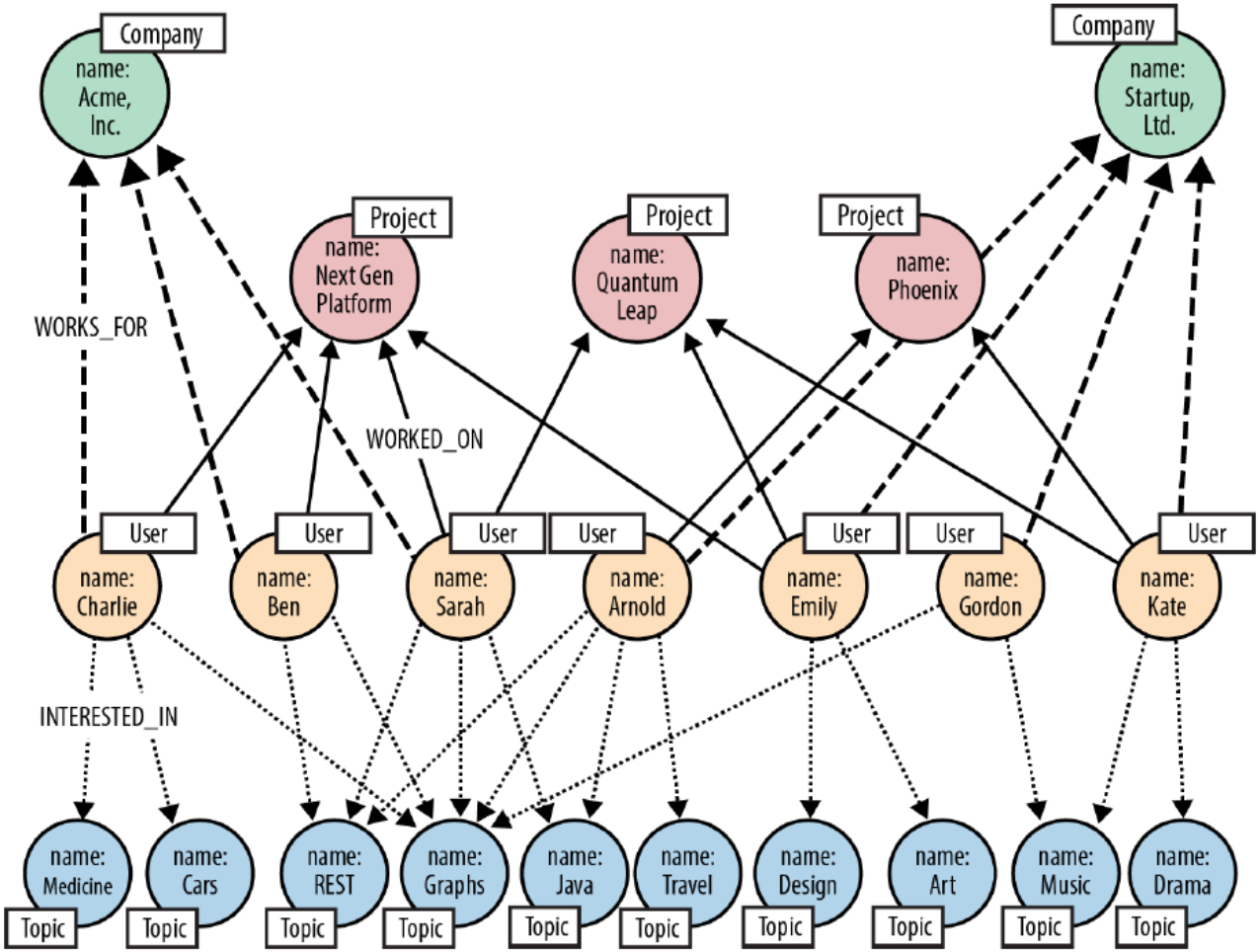
Formulieren Sie folgende Anfragen in Cypher:

5. Geben Sie alle Unternehmen an, welche der Kategorie 'Mexican' aber nicht der Kategorie 'Fast Food' zugeordnet wurden!
6. Geben Sie die 10 am besten bewerteten Kasinos (Kategorie: 'Casinos') in Nevada (State: 'NV') an (absteigend sortiert nach Bewertung)!
7. Geben Sie alle Paare von Nutzern an, die mehr als 50 gleiche Kategorien bewertet haben!
8. Gesucht sind Nutzer, welche viel reisen. Geben Sie hierzu die Namen der Nutzer und die Anzahl der unterschiedlichen Städte, die sie besucht haben (bzw. eine Bewertung hinterlassen haben), an! Sortieren Sie die Ausgabe nach Anzahl der Städte in absteigender Ordnung!
9. Wir wollen die drei beliebtesten Kategorien bezüglich jeder Region (State) haben. Geben Sie hierfür den Namen der Region, den Namen der Kategorie und die durchschnittliche Bewertung an!

Inhaltsverzeichnis: Graphdatenbanken

Anwendung: Empfehlungen

Empfehlung von Ressourcen (Produkte, Service, Menschen), welche für eine Person oder ein Gruppe interessant sein können



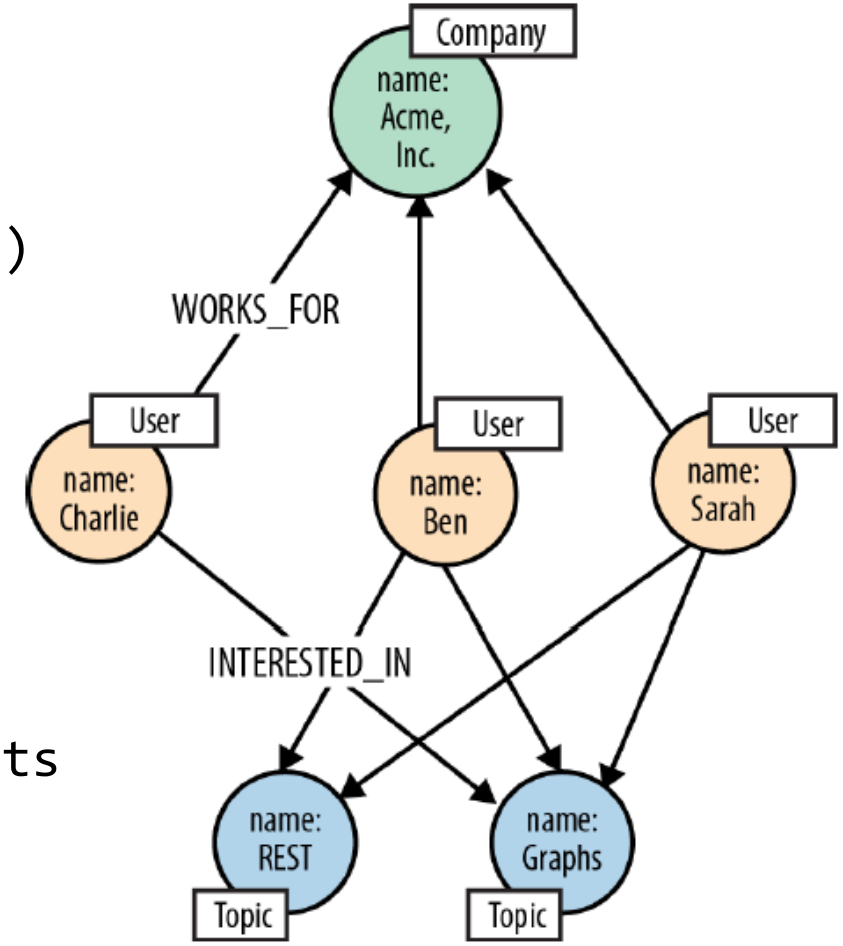
Quelle: Robinson et al (2015), S. 113

Anwendung: Empfehlungen

Empfehlung von Ressourcen (Produkte, Service, Menschen), welche für eine Person oder ein Gruppe interessant sein können

1. Empfehlungen von Freunden über gemeinsame Interessen

```
MATCH (subject:User {name:{name}})
MATCH (subject)-[:INTERESTED_IN]-
>(interest:Topic) <-
[:INTERESTED_IN]-(person:User),
(person)-[:WORKS_FOR]-
>(company:Company)
RETURN person.name, company.name,
count(interest) AS score,
collect(interest.name) AS interests
ORDER BY score DESC
```



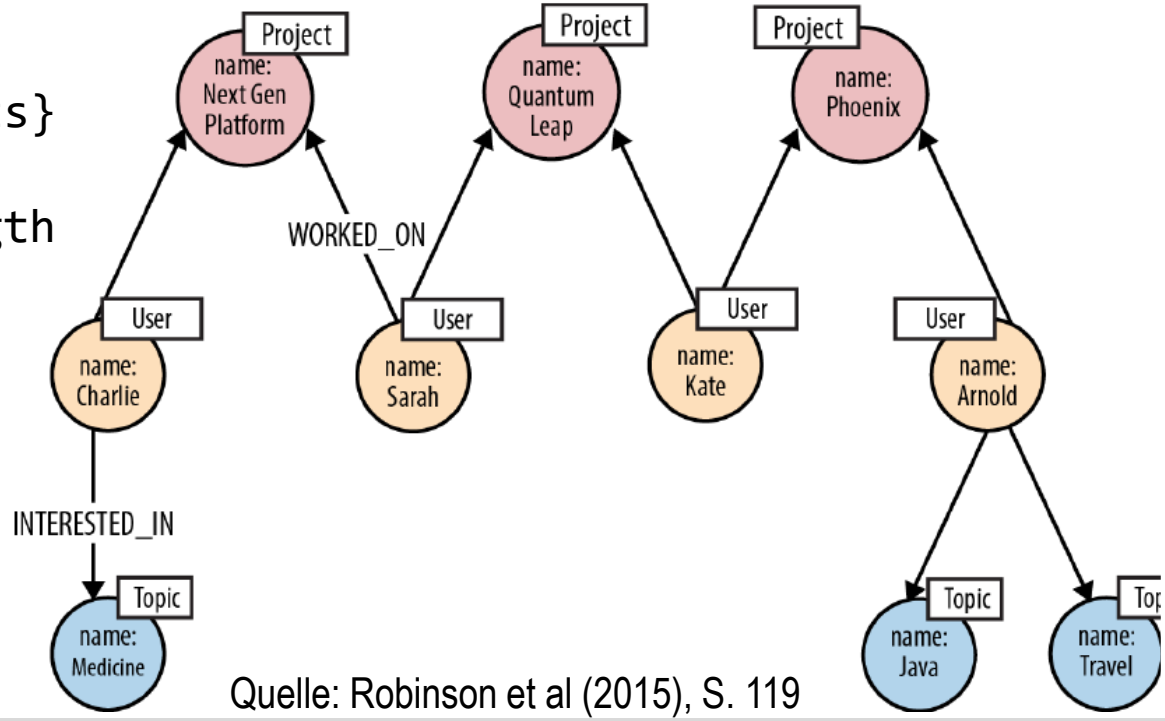
Quelle: Robinson et al (2015), S. 115

Anwendung: Empfehlungen

Empfehlung von Ressourcen (Produkte, Service, Menschen), welche für eine Person oder ein Gruppe interessant sein können

2. Empfehlungen von Personen mit bestimmten Interessen

```
MATCH (subject:User {name:{name}})
MATCH p=(subject)-[:WORKED_ON]->(:Project)-[:WORKED_ON*0..2]-
(:Project)<-[:WORKED_ON]-(person:User)-[:INTERESTED_IN]-
>(interest:Topic)
WHERE person<>subject AND
interest.name IN {interests}
WITH person, interest,
min(length(p)) as pathLength
ORDER BY interest.name
RETURN person.name,
count(interest) AS score,
collect(interest.name),
((pathLength - 1)/2)
ORDER BY score DESC
LIMIT 25
```



Quelle: Robinson et al (2015), S. 119

Anwendung: Geodaten

Verarbeitung von Geodaten: Routen und Distanzen zwischen Orten

city	city_ascii	lat	lng	pop	country
Greenock	Greenock	55.93329002	-4.750030763	59065	United Kingdom
Sunderland	Sunderland	54.92001853	-1.380029746	315449.5	United Kingdom
Southampton	Southampton	50.90003135	-1.399976849	384417	United Kingdom
Bristol	Bristol	51.44999778	-2.583315472	492120.5	United Kingdom
Bournemouth	Bournemouth	50.72999005	-1.900049684	295272.5	United Kingdom
Omagh	Omagh	54.60001223	-7.300004315	18691	United Kingdom
Chester	Chester	53.20002016	-2.919987428	83285.5	United Kingdom
Swansea	Swansea	51.6299868	-3.950002077	232611	United Kingdom
Carlisle	Carlisle	54.87999514	-2.929986818	69270	United Kingdom

```
LOAD CSV WITH HEADERS FROM 'worldcities-basic.csv' as line
CREATE (c:City {name: line.city,
          location: point({ latitude: toFloat(line.lat),
                            longitude: toFloat(line.lng)}),
          population: line.pop})
MERGE (country:Country {name: line.country})
CREATE (c)-[:IN]->(country)
```

Quelle: https://medium.com/neo4j/whats-new-in-neo4j-spatial-features-586d69cda8d0

Anwendung: Geodaten

Verarbeitung von Geodaten: Routen und Distanzen zwischen Orten

1. Welche Städte sind London am nächsten?

```
MATCH (c1:City { name: "London" })-[:IN]->(:Country { name:
"United Kingdom" })<-[:IN]-(c2:City)
WHERE c2.name <> "London"
WITH c1, distance(c1.location, c2.location) as dist, c2
RETURN c1.name, dist, c2.name ORDER BY dist ASC LIMIT 10;
```

2. Berechnung aller Distanzen zwischen den Städten

```
MATCH (c1:City)-[:IN]->(:Country { name: "United Kingdom"
})<-[:IN]-(c2:City)
WHERE id(c1) < id(c2)
CREATE (c1)-[r:PATH { distance: distance(c1.location,
c2.location) }]->(c2)
RETURN count(r);
```

Quelle: https://medium.com/neo4j/whats-new-in-neo4j-spatial-features-586d69cda8d0

Weitere Anwendungen

- **Analyse sozialer Netzwerke**: Vorhersage von Verhalten/Attributen über soziale Beziehungen
- **Datenverwaltung**: Identifizierung, Säuberung und Verwaltung von zerstreut gespeicherten und teilweise redundanten Daten in verschiedenen Formaten und mit unterschiedlichen Zugangsrechten
- **Verwaltungen von Netzwerken und Datenzentren**: Identifizierung von Abhängigkeiten zwischen Komponenten und Komponenten/Kunden
- **Betrugserkennung**: Erkennung von Mustern und schnelle Reaktionen

Inhaltsverzeichnis: Graphdatenbanken

DB Ranking

Quelle: http://db-engines.com/en/ranking/

☐ include secondary database models — 31 systems in ranking, March 2019

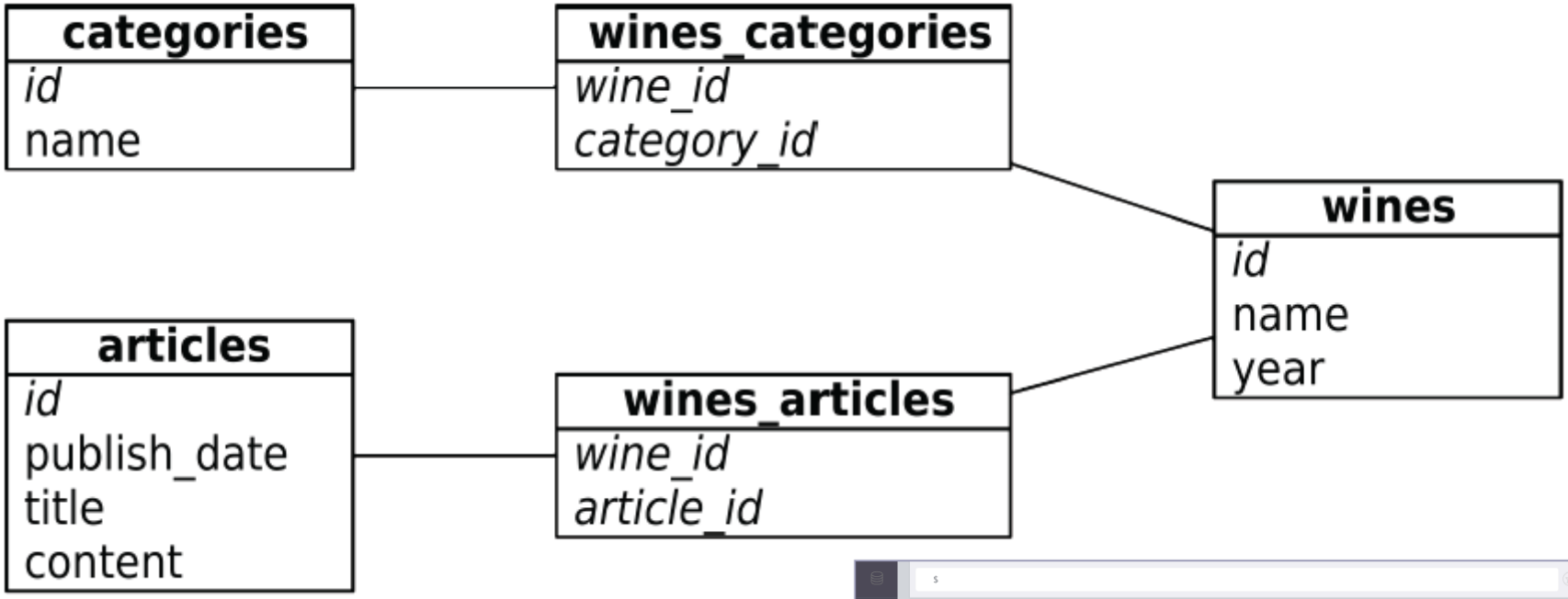
Rank Mar 2019	Rank Feb 2019	Rank Mar 2018	DBMS	Database Model	Score Mar 2019	Score Feb 2019	Score Mar 2018
1.	1.	1.	Neo4j	Graph	48.58	+0.72	+7.68
2.	2.	2.	Microsoft Azure Cosmos DB	Multi-model	24.83	-0.03	+8.07
3.	3.	3.	OrientDB	Multi-model	6.13	+0.08	-0.33
4.	4.	4.	ArangoDB	Multi-model	4.26	-0.10	+0.14
5.	5.	5.	Virtuoso	Multi-model	3.20	+0.26	+1.37
6.	6.	↑ 13.	JanusGraph	Graph	1.33	+0.10	+1.02
7.	↑ 8.	↓ 6.	Giraph	Graph	1.05	+0.04	-0.03
8.	↓ 7.	↓ 7.	Amazon Neptune	Multi-model	1.03	-0.03	+0.33
9.	↑ 10.	↑ 11.	GraphDB	Multi-model	0.93	+0.07	+0.48
10.	↓ 9.	↓ 8.	AllegroGraph	Multi-model	0.88	-0.01	+0.24
11.	↑ 13.	↑ 22.	TigerGraph	Graph	0.81	+0.13	+0.70
12.	12.	↓ 9.	Stardog	Multi-model	0.77	+0.08	+0.24
13.	↓ 11.	↑ 18.	Dgraph	Graph	0.69	-0.01	+0.57
14.	14.	↓ 12.	Graph Engine	Multi-model	0.56	-0.01	+0.14
15.	15.	15.	Blazegraph	Multi-model	0.55	+0.02	+0.40
16.	↑ 18.	↑ 21.	FaunaDB	Multi-model	0.52	+0.16	+0.41
17.	↓ 16.	↓ 10.	Sqrrl	Multi-model	0.50	+0.06	+0.03
18.	↓ 17.	↑ 19.	InfiniteGraph	Graph	0.39	-0.01	+0.27
19.	↑ 20.	↑ 20.	InfoGrid	Graph	0.28	+0.05	+0.16
20.	↓ 19.	↓ 16.	FlockDB	Graph	0.26	+0.01	+0.12
21.	21.	↓ 17.	HyperGraphDB	Graph	0.18	+0.01	+0.05
22.	↑ 23.		AnzoGraph	Graph, Multi-model	0.18	+0.04	
23.	↓ 22.	↑ 25.	AgensGraph	Multi-model	0.17	+0.02	+0.13
24.	24.	↓ 14.	Sparksee	Graph	0.12	-0.01	-0.11
25.	↑ 26.	↓ 24.	TinkerGraph	Graph	0.10	+0.00	+0.05
26.	↓ 25.	↓ 23.	VelocityDB	Multi-model	0.10	-0.01	+0.01
27.	27.	↑ 28.	GRAKN.AI	Multi-model	0.09	0.00	+0.07
28.	↑ 29.	↓ 27.	GraphBase	Graph	0.07	+0.03	+0.04
29.	↓ 28.		Memgraph	Graph	0.05	-0.03	

Neo4j

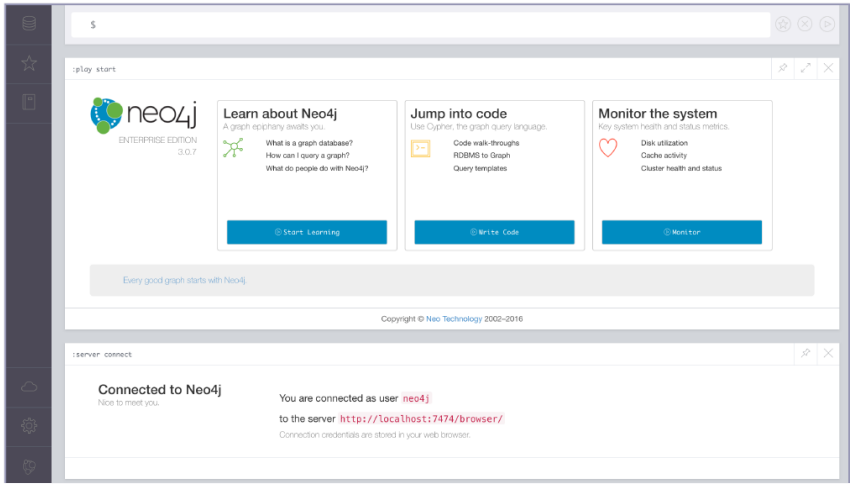
- Entwickelt seit 2004 (2010: Version 1.0; aktuell: Version 3.5)
- Open Source: github.com/neo4j/neo4j
- Implementierung des Property Graph Model
- „Natives“ GDBMS: Indexfreie Adjazenz
- Fokus auf OLTP mit lokalem Bezug, aber auch Graphanalyse
- ACID Garantien für Transaktionen
- Schema ist optional
- Zugriff (empfohlen): Cypher
  - Neo4j Desktop/Browser und Shell
  - HTTP REST API, C#, Java, JavaScript, Python
- Zugriff auch „embedded“ in Java möglich: Core API
- Erweiterbar: APOC (Nutzer-definierte Funktionen)
- Horizontal Skalierbar (Replikation): Master-Slave

Neo4j: Demo

- Empfehlungssystem für Wein



- Quelle: Perkins et al (2018), Kapitel 6



Neo4j: Indizes und Restriktionen

- Index auf Attribut

```
CREATE INDEX ON :Wine(name);
```

- Restriktion

```
CREATE CONSTRAINT ON (w:Wine)
ASSERT w.name IS UNIQUE;
```

Funktionen (https://neo4j.com/docs/cypher-manual/current/functions/)

- Predicate: all(), any(), none(), single(), exists()
- Scalar: endNode(), head(), id(), last(), length(), ...
- Aggregating: avg(), collect(), count(), max(), min(), ...
- Lists: keys(), labels(), nodes(), reduce(), reverse(), ...
- Mathematical: abs(), round(), exp(), log(), sqrt(), ...
- String: left(), replace(), reverse(), split(), substring(), ...

Neo4j: Features (Auswahl)

Rechtemanagement

- Begrenzter Zugriff auf Teilgraphen
- Begrenzter Zugriff auf Attribute

```
dbms.security.property_level.enabled=true
dbms.security.property_level.blacklist=\
       Confidential=top_secret;
       Unclassified=top_secret,confidential

CALL dbms.security.addRoleToUser("Confidential", "james");
CALL dbms.security.addRoleToUser("Unclassified", "tim");

CREATE (u:Document {name:'Aliens?',
                    top_secret:'They hate us!',
                    confidential:'They exist!',
                    public:'They do not exist.'})
```

Quelle: https://maxdemarzi.com/2018/03/12/keeping-properties-secret-in-neo4j/

Neo4j: Features (Auswahl)

Volltextsuche

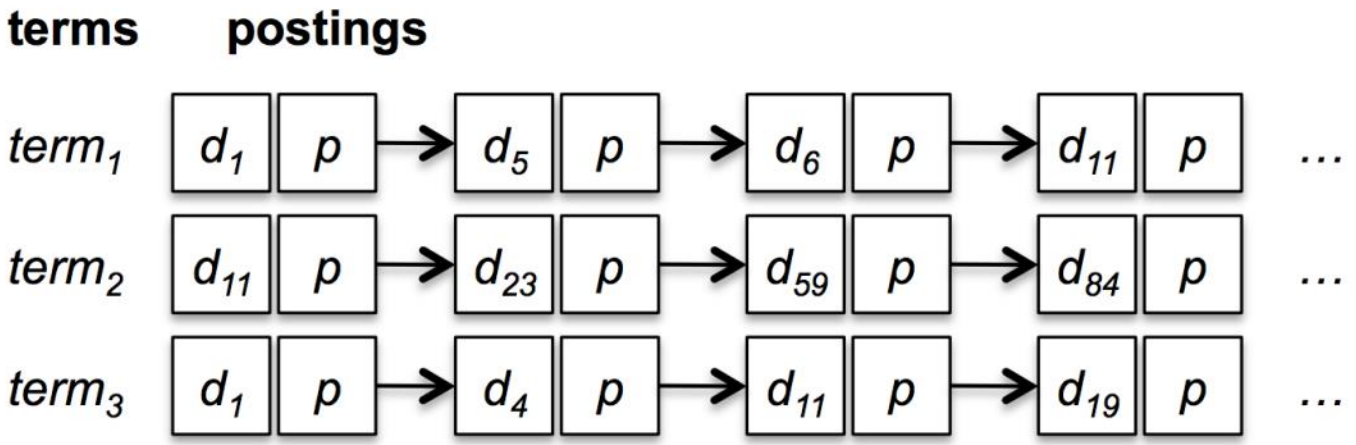
- Über Apache Lucene (Inverted Indexes)
- Unterstützung von Phrasen, Wildcards, Einschränkung auf Attribute, FuzzySearch (Damarau-Levenshtein Distanz)

```
CALL db.index.fulltext.queryNodes('books', 'secret')
CALL db.index.fulltext.queryNodes('books', '"secret life"')
CALL db.index.fulltext.queryNodes('books', 'secr*')

CALL db.index.fulltext.queryNodes('books', 'authors: rowling')
CALL db.index.fulltext.queryNodes('books', 'authors: rowling
                                            AND title: goblet')

CALL db.index.fulltext.queryNodes('books', 'garde~')
```

Quelle: https://graphaware.com/neo4j/2019/01/11/neo4j-full-text-search-deep-dive.html

Einschub: Inverted Index

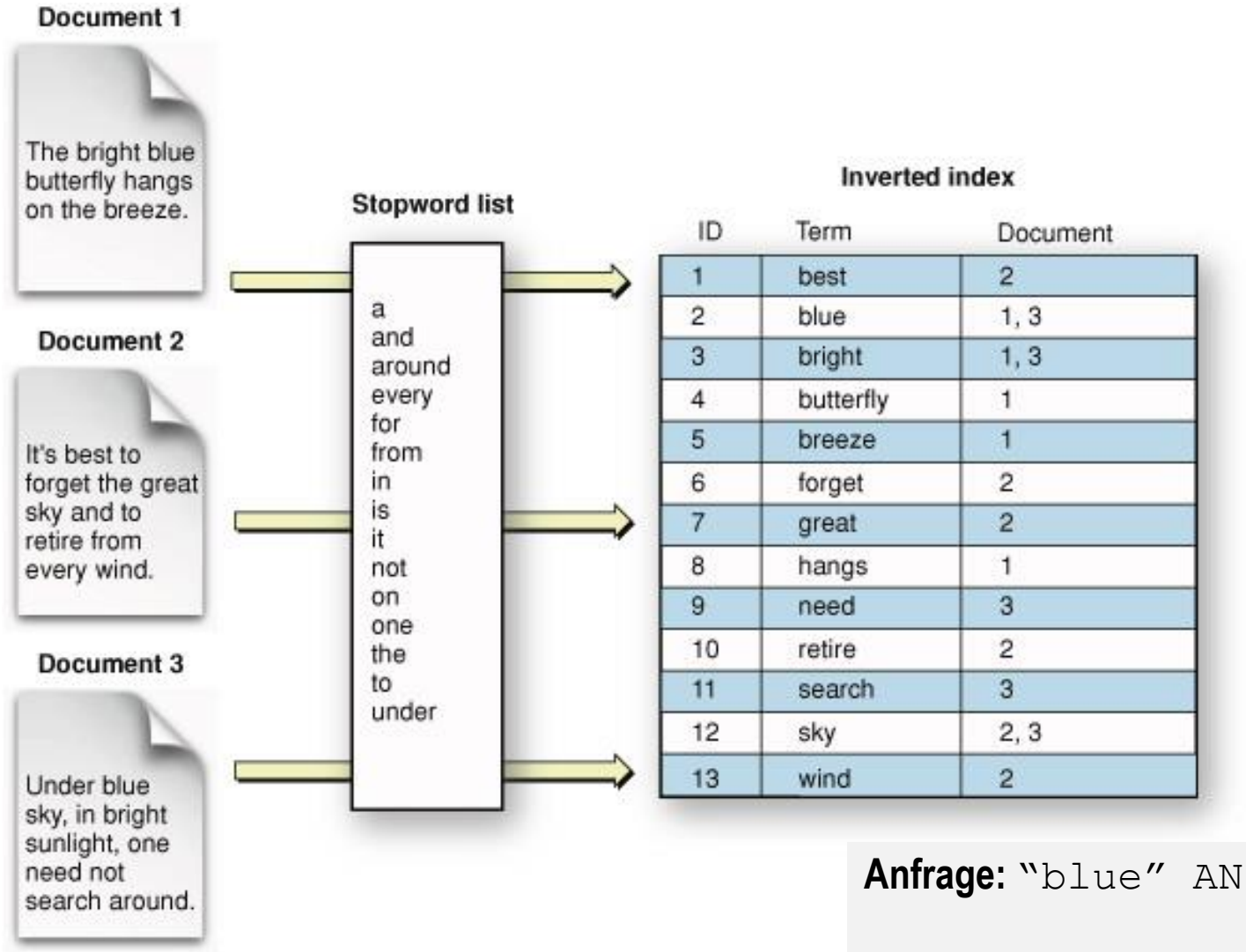
- Index-Struktur für Terme einer Dokumentenkollektion
  - Zuordnung: Term → Liste der Dokumente, die Term enthalten
  - Erweiterungen: Häufigkeit der Terme im Dokument, Position im Dokument, ...



https://medium.com/@shashankbaravani/database-storage-engines-under-the-hood-705418dc0e35

- Suchanfrage nach Termen kann effizient durch Index realisiert werden
- Unterstützung von AND, OR und NOT durch Mengenoperationen
- Kompression der Einträge zur effizienten Speicherung und Suche
- Aufwendige Aktualisierung des Index

Einschub: Inverted Index - Beispiel



http://karthikkumar.me/images/intro-to-lucene/inverted_index.jpg

Anfrage: "blue" AND "sky"

{d1,d3} ∩ {d2,d3} = {d3}

Neo4j: Features (Auswahl)

Graphanalyse

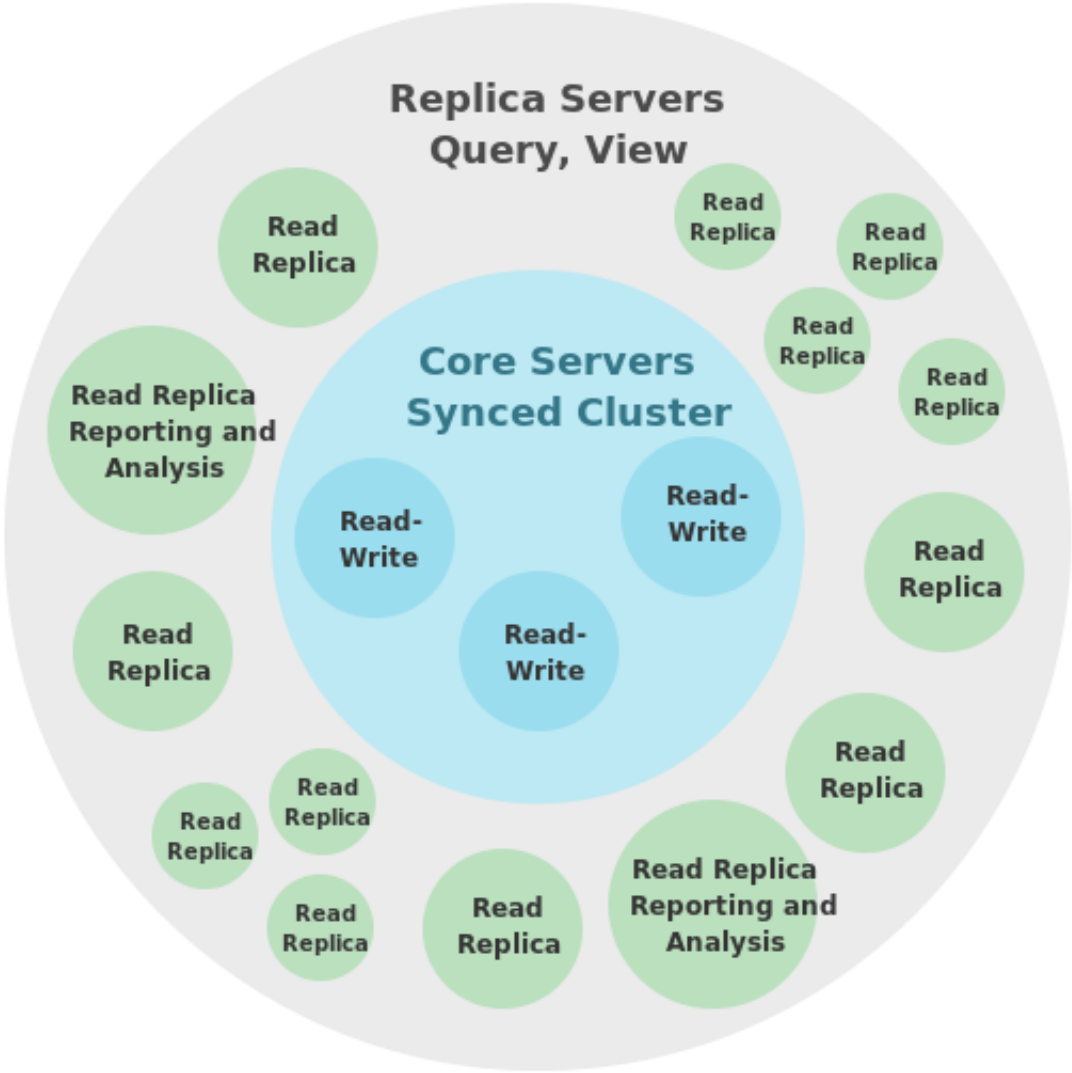
- Zentralität (PageRank, Betweenness, Closeness, …)
- Community Detection (Louvain, Label Propagation, …)
- Pfade (Kürzester Pfad, Dijkstra, A*, …)
- Ähnlichkeit (Jaccard, Cosine, Pearson, Euclidean, …)
- Link Prediction (Common Neighbors, Preferential Attachement, …)
- Natural Language Processing (Topics Extraction, Word2Vec, …)
- Machine Learning (In Bearbeitung)

Neo4j: Eigenschaften

- ACID Garantien für Transaktionen
  - Schreibsperren
  - Speicherung auf Festplatte bei Erfolg: über „Write Ahead Log“
  - Rollback bei Fehler/Abbruch
- Transaction Replay nach Fehler (z.B. Serverabsturz)
  - „Active Transaction Log“ und „Write Ahead Log“
  - Replay ist „idempotent“
- In-memory Caching
  - Store Files werden in Regionen aufgespalten
  - Cache beinhaltet eine feste Anzahl an Regionen pro Store File
  - Regionen mit wenig (kurz- und langfristigen) Zugriffen werden aus Cache entfernt
- Lokale Anfragen: Traversierungen ausgehend von bestimmten Knoten
  - Unterstützung durch Speicherstruktur (indexfreie Adjazenz) und Caching
  - Schneller als globale Anfragen über eine Menge unbestimmter Knoten

Neo4j: Verfügbarkeit

- Causal Cluster
- Replikation: Master-Slave-Architektur
- Keine Datenpartitionierung: Vollständiger Graph auf jedem Replikat
- Ein paar wenige Core Server für Datensicherheit
- Zahlreiche Read Replika für Skalierbarkeit von Leseanfragen



Neo4j: Causal Cluster

- Core Server:
  - Annahme aller Schreibbefehle
  - Replikation zwischen Core Servern über **Raft Protokoll**: Transaktion ist erfolgreich, falls diese von Mehrheit aller Core Server verarbeitet wurde
  - Wahl eines Leaders: Verantwortlich für alle Transaktionen
  - Serialisierbarkeit (innerhalb Core Server Cluster)
  - Verfügbar solange eine Mehrheit der Server funktioniert/erreichbar (andernfalls werden verbleibende Core Server Read-Only)
- Read Replikate
  - Read-Only Kopien der Core Server
  - Asynchrone Synchronisation mit Core Server (im Abstand von Millisekunden)
  - Kausale Konsistenz
    - Read-your-Writes über Bookmarks
    - Nutzer erhält Bookmark beim Schreiben auf Core Server
    - Sicherstellen, dass nur die Server eine weitere Transaktion des Nutzers ausführen, wenn sie die mit einem Bookmark versehenen Transaktionen verarbeitet haben

Neo4j: Causal Cluster

- Aufsetzen eines lokalen Clusters
- Benötigt Neo4j Enterprise Edition (http://dist.neo4j.org/neo4j-enterprise-3.5.3-unix.tar.gz)

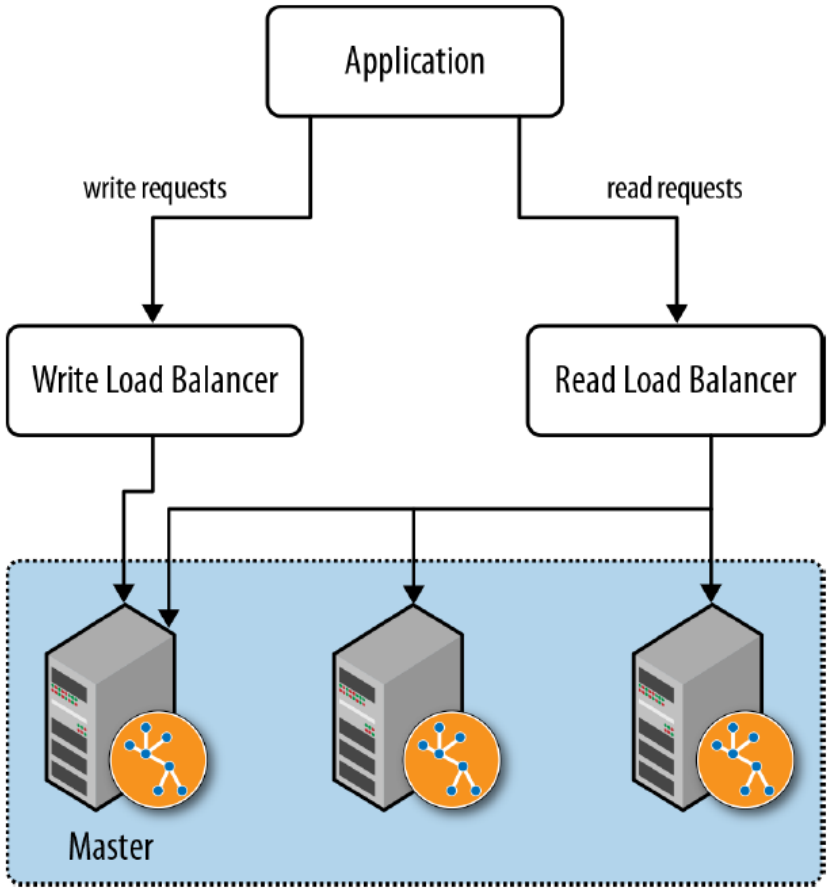
```
cd neo4jcluster

tar -xf neo4j-enterprise-3.5.3-unix.tar.gz
cp -R neo4j-enterprise-3.5.3 neo4j-1.local
cp -R neo4j-1.local neo4j-2.local
cp -R neo4j-1.local neo4j-3.local

nano neo4j-1.local/conf/neo4j.conf
nano neo4j-2.local/conf/neo4j.conf
nano neo4j-3.local/conf/neo4j.conf

neo4j-1.local/bin/neo4j start
neo4j-2.local/bin/neo4j start
neo4j-3.local/bin/neo4j start
```

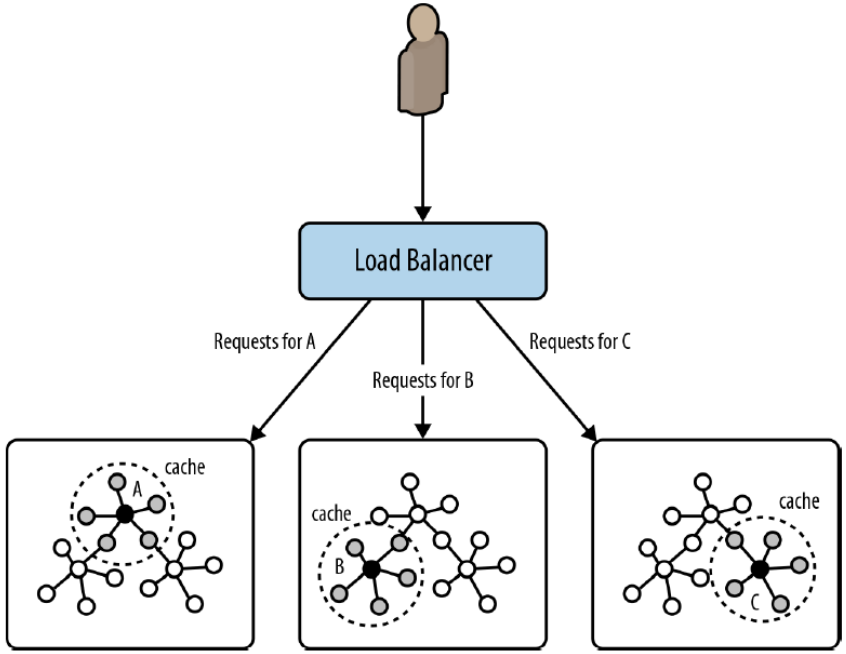
Für Parameter, siehe https://neo4j.com/docs/operations-manual/current/tutorial/local-causal-cluster/

Neo4j: Load Balancing

Nur über Applikation möglich (keine Unterstützung durch Neo4j)

- z.B. HTTP POST/PUT/DELETE zu Core und HTTP GET zu Replikat
- Cache Sharding

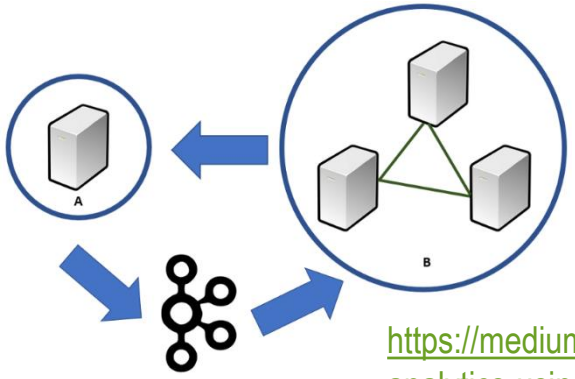


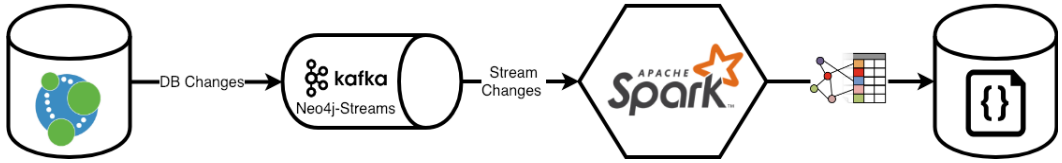


Quelle: Robinson et al (2015), S. 83-84

Neo4j: Grenzen

- Keine Datenpartitionierung
  - Über Applikation möglich, falls Graph klare Grenzen (unverbundene Teilgraphen)
  - Ohne klare Grenzen:
    - NP-schweres Problem und daher bisher nur ineffiziente Lösungen
    - Traversierung muss evtl. mehrmals zwischen Servern wechseln
- Graph Algorithmen (z.B. Page Rank) benötigt viele Ressourcen
  - Showcase: https://go.neo4j.com/WBCODWBRGraphAlgorithms_LP-Video.html
  - Größter Graph: 3 Milliarde Knoten und 18 Milliarden Kanten
  - Server mit 900 GB RAM und 144 CPUs
- Lösung: Auslagern der analytischen Berechnungen/verteilte Berechnung





https://medium.freecodecamp.org/how-to-leverage-neo4j-streams-and-build-a-just-in-time-data-warehouse-64adf290f093

https://medium.freecodecamp.org/how-to-embrace-event-driven-graph-analytics-using-neo4j-and-apache-kafka-474c9f405e06

Übung

- Installation Neo4j: z.B. https://neo4j.com/docs/operations-manual/current/installation/
- Yelp Dataset: https://www.yelp.com/dataset/documentation/main
- Reduzierter Datensatz auf AlmaWeb: Reviews aus dem Jahr 2006
  - review2006.json
  - business2006.json
  - user2006.json
- Import Yelp-Daten
  - Über APOC: https://neo4j.com/docs/graph-algorithms/current/yelp-example/
  - Anpassung der Befehle um zusätzlich die Beziehungen Business-City und City-State hinzuzufügen
  - Entfernen der Nutzer ohne Namen:

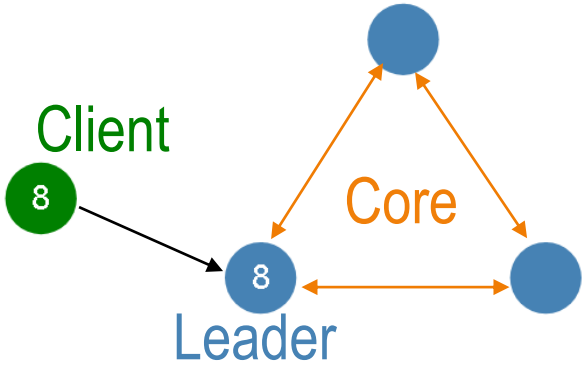
```
MATCH (n:User)
WHERE n.name is null
OPTIONAL MATCH (n)-[r]-()
DELETE n, r;
```

- Ausführen der Cypher-Anfragen aus obiger Übung

Inhaltsverzeichnis: Graphdatenbanken

Raft Protokoll

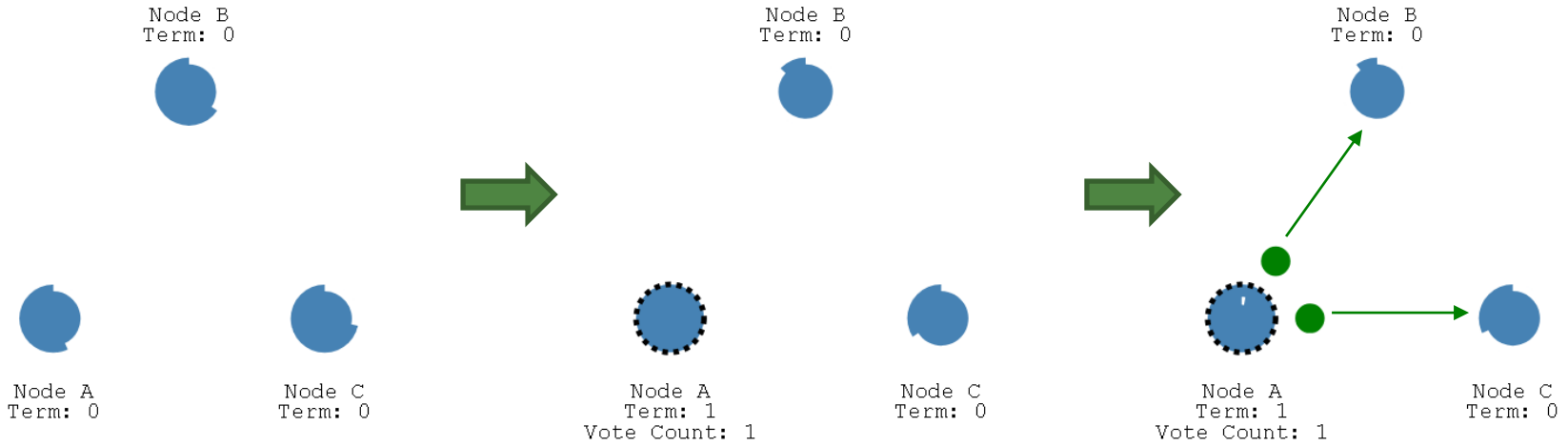
- Konsens zwischen eine Gruppe von Replikaten (z.B. Neo4j Core Server)
  - System ist funktionsfähig solange Mehrheit der Server funktionieren
  - Datensicherheit auch bei Ausfall der Mehrheit
  - Erlaubte Fehler: Ausfall eines Servers oder des Netzwerks (nicht-byzantinische Fehler)



- Alle Server haben ***Transaction Log*** mit allen ausgeführten Transaktionen: dieser soll serialisierbar repliziert werden
- 3 Rollen in Raft: **Follower**, **Candidate**, **Leader**
  - *Es kann immer nur einen Leader geben*
  - Der Leader bekommt alle Transaktionen und ist für deren Replikation zuständig
- Eine **Runde** kann mehrere Transaktionen von Clients beinhalten, hat aber nur einen *Leader* (neuer Leader = neue Runde)
- Quellen: https://raft.github.io/ und http://thesecretlivesofdata.com/raft/

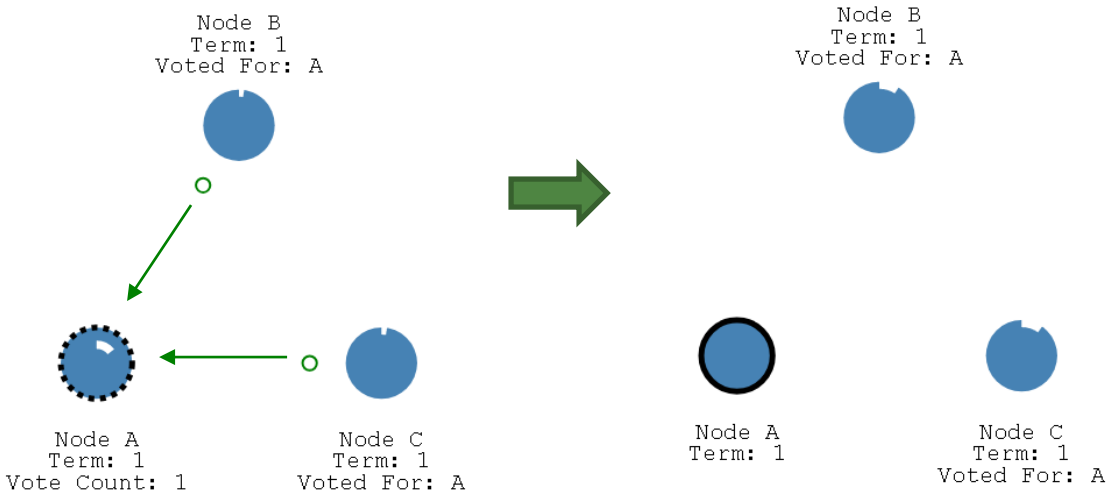
Raft Protokoll: Leader Election 1

- Zu Beginn sind alle Server in der Rolle des *Follower*
  - Jeder Follower bekommt einen zufälligen *Election Timeout* (150-300 ms)
  - Nach Ablauf des Election Timeout: Follower (Node A) wird zu *Candidate* und eine *neue Runde (Term)* beginnt
  - Candidate sendet *Request Votes* (●) zu allen anderen Servern und setzt Election Timeout zurück



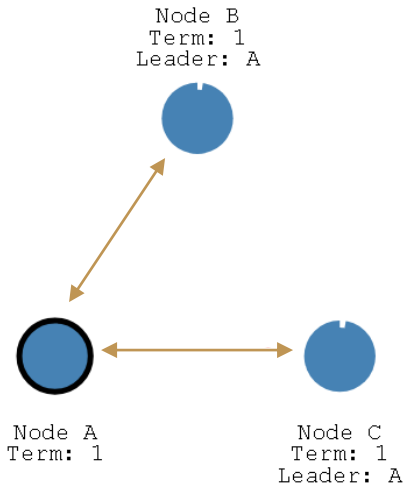
Raft Protokoll: Leader Election 2

- Pro Runde hat jeder Knoten eine Stimme
  - Ein Candidate (Node A) stimmt für sich selbst
  - Ein Follower (Nodes B und C) stimmt für Candidate von dem er den ersten Request Vote bekommen hat und setzt den Election Timeout zurück
  - Ein Follower stimmt für Candidate nur dann, wenn dessen Log (über Request Vote erfahren) mind. so aktuell wie der eigene Log ist
- Candidate mit Mehrheit an Stimmen wird zu „Leader“



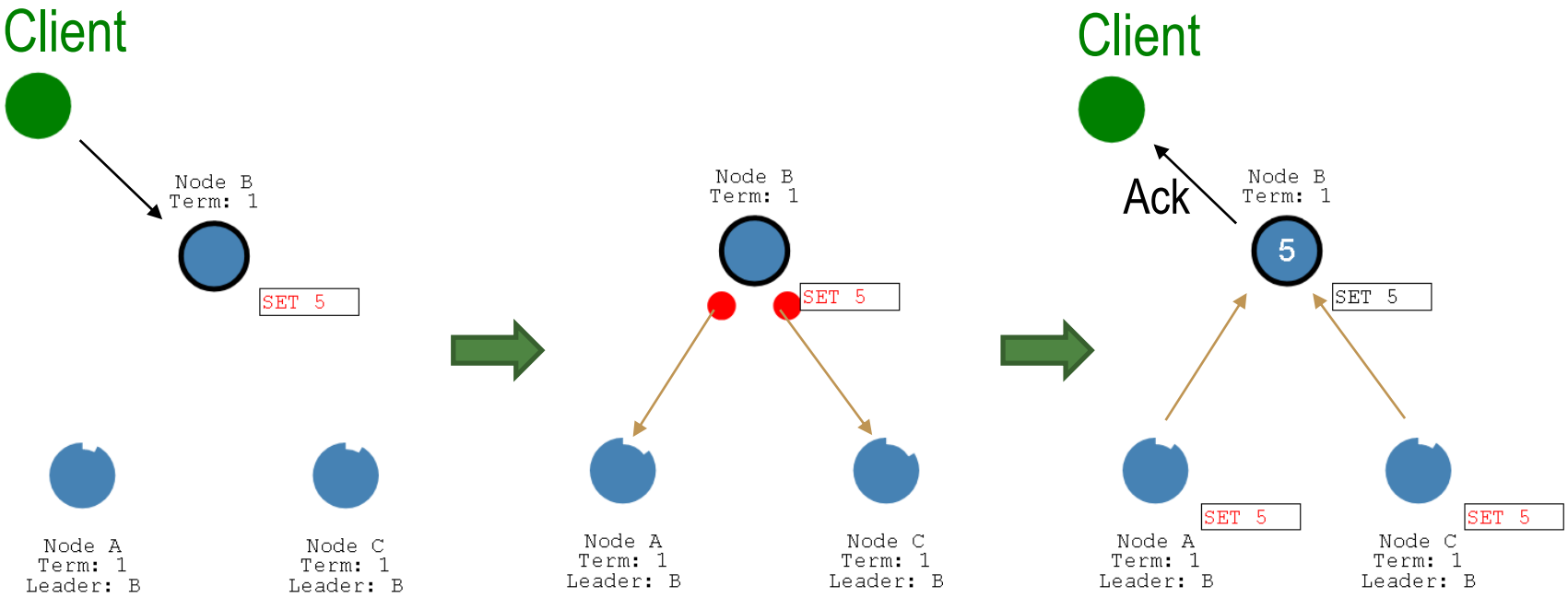
Raft Protokoll: Log Replication

- Leader sendet regelmäßig *Append Entries* zu allen Followern (*Heartbeat Timeout < 150ms*)
  - Follower antworten auf jeden Append Entry und setzten Election Timeout zurück
  - Konsensprüfung: mit jedem Append Entry wird aktueller Zustand der Follower überprüft und, wenn nötig, aktualisiert



Raft Protokoll: Log Replication

- Nur der Leader nimmt Transaktionen von Clients entgegen
  - Transaktionen werden über Append Entries an andere Server verteilt
  - Falls Mehrheit der Follower die Transaktion bestätigt erfolgt Commit und Antwort an Client (Acknowledge)



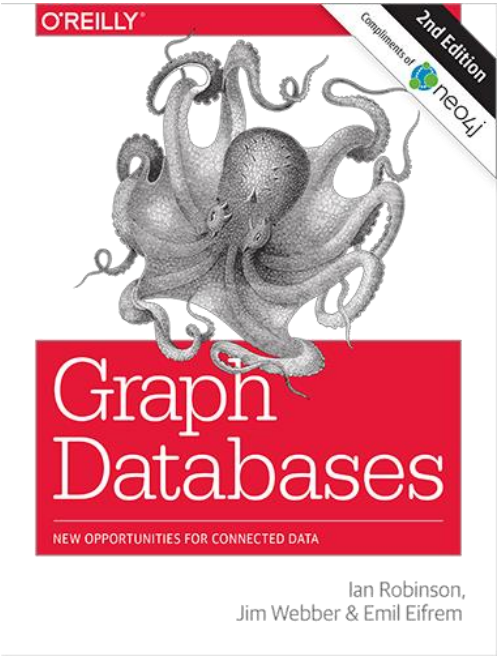
Inhaltsverzeichnis: Graphdatenbanken

Zusammenfassung: Graphdatenbanken

- Verstärktes Aufkommen stark vernetzter umfangreicher Daten
  - Soziale, technologische, biologische und Informationsnetzwerke
  - Nutzung für Empfehlungen, Datenmanagement, BI, Betrugserkennung, …
- Porperty Graph Modell als generisches Modelle vernetzter Daten
- GDBMS vs RDBMS: Einsatz abhängig von Art der Daten/Anfragen
- Natives GDBMS ermöglicht effizienten lokalen Zugriff auf Beziehungen: Indexfreie Adjazenz
- Cypher: Anfragesprache für GDBMS
- Neo4j: derzeit populärste Graphdatenbank mit umfangreichen Features

Quellen

- I. Robinson, Webber, J. and Eifrem, E.: Graph databases, 2nd edition, 2015, https://neo4j.com/lp/book-graph-databases/
- L. Perkins, Redmond, E. and Wilson, J. R.: Seven Databases in Seven Weeks, 2nd edition, 2018, Kapitel 6
- Neo4j: https://neo4j.com/docs/getting-started/current/



- Neo4j Internals: http://www.slideshare.net/thobe/an-overview-of-neo4j-internals
- Martin Junghanns, Max Kießling: “(Cypher)-[:ON]->(ApacheFlink)<-[:USING]-(Gradoop)”, FOSDEM 2017, Graph Devroom
- Marko A. Rodriguez, Peter Neubauer. 2010. Constructions from Dots and Lines.
- Renzo Angles and Claudio Gutierrez. 2008. Survey of graph database models. ACM Comput. Surv. 40, 1, Article 1 (February 2008), 39 pages.