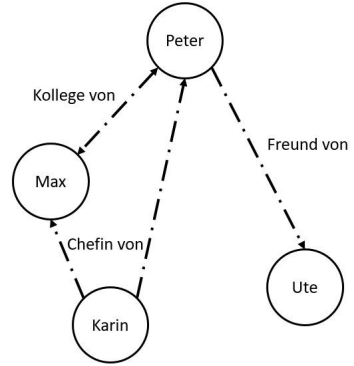
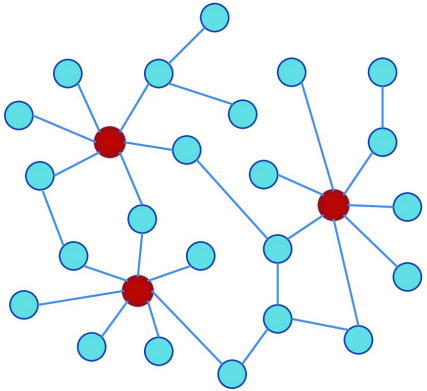
Ulla Moroff,
Raphael Kraus,
Michael Schwarz,
Philipp Seeberger,
Steffen Schulz

Quelle: https://memgraph.com/blog/graph-database-vs-relational-database

Graphdatenbanken

- Entities und deren Beziehungen werden mittels Knoten und Kanten dargestellt
- Zunehmend von Interesse aufgrund stark vernetzter Datenmengen
- Viele Anwendungsbereiche lassen sich in natürlicher Weise auf Graphen abbilden
- Graphen ermöglichen eine für Menschen einfach nachvollziehbare Darstellung

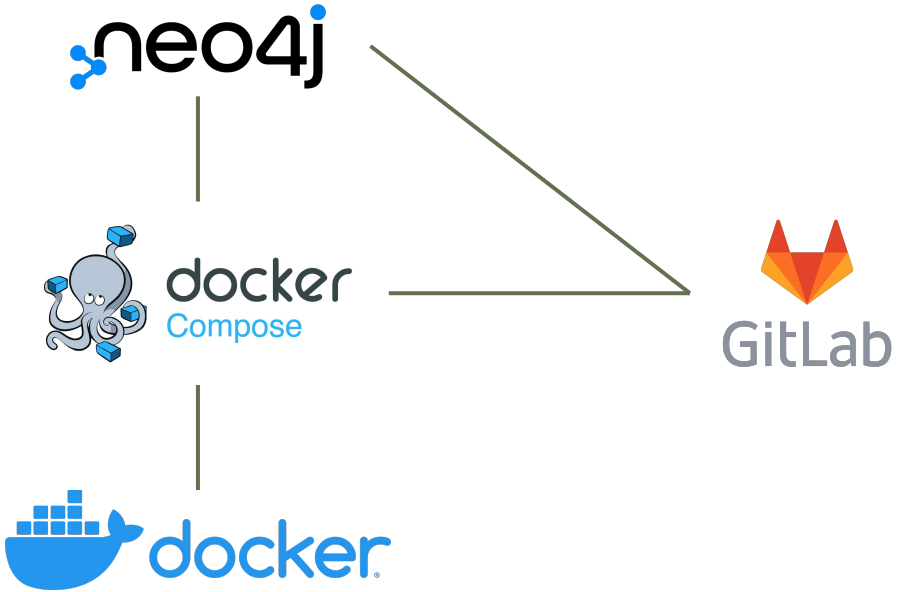




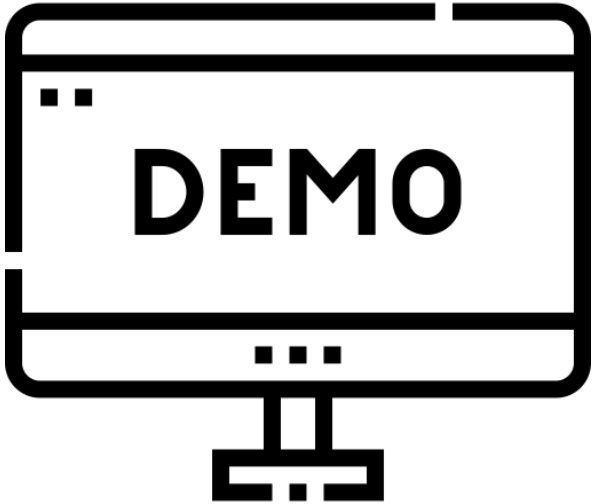
Agenda

1. Einrichten der Neo4j Datenbank
   - 1.1. Technologie-Überblick
   - 1.2. Neo4j Browser
2. Architektur und Herausforderungen
   - 2.1. Schemamigration
   - 2.2. Cypher Query Language
   - 2.3. CRUD Operationen
3. Fazit

1.1 Technologie-Überblick

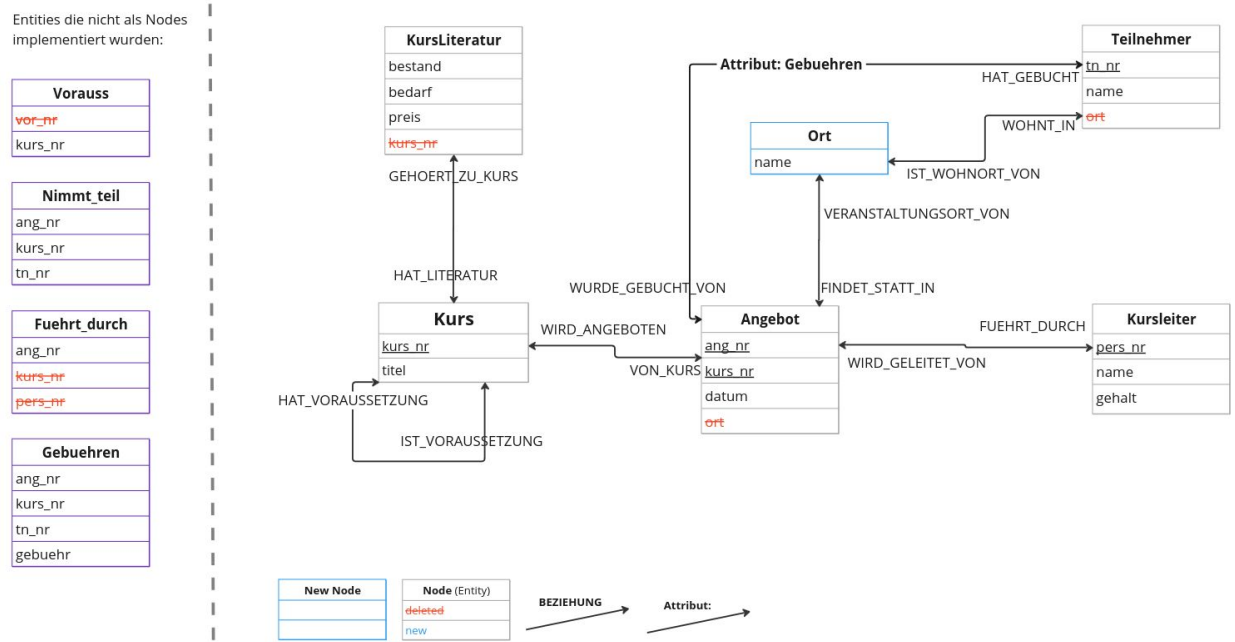


1.2 Neo4j Browser



2.1 Schemamigration

- Die Zeilen einer Tabelle werden zu Knoten
- Das Label der Knoten wird aus den Tabellennamen abgeleitet
- Joins und Fremdschlüssel werden in Kanten überführt



2.2 Cypher Query Language

```
1
2   MATCH (a:Angebot)-[edge:FINDET_STATT_IN]->(o:Ort {name:'Wedel'})
3   MATCH (o)-[edge2:VERANSTALTUNGSORT_VON]->(a)
4   MATCH (change_o:Ort {name:'Augsburg'})
5   DELETE edge, edge2
6   CREATE (a)-[:FINDET_STATT_IN]->(change_o), (change_o)-[:VERANSTALTUNGSORT_VON]->(a);
7
```

Statement in Cypher

```
1
2   UPDATE Ort set name = 'Augsburg' where name = 'Wedel';
3
```

Statement in SQL

2.2 Cypher Query Language

```
1
2   MATCH (a:Angebot)-[:FINDET_STATT_IN]->(o:Ort)
3   MATCH (a)-[:ANGEBOT_KURS]->(k:Kurs)
4   MATCH (a)-[:WIRD_GELEITET_VON]->(kl:Kursleiter)
5   RETURN k.titel, kl.name, a.datum, o.name;
6
```

Statement in Cypher

```
1
2   SELECT k.Titel, k.name, a.Datum a.Ort
3   FROM Angebot a
4   INNER JOIN Kurs k ON k.KursNr = a.KursNr
5   INNER JOIN Fuehrt_durch fd ON fd.KursNr = a.KursNr
6   INNER JOIN Kursleiter kl on kl.PersNr = fd.PersNr
7
```

Statement in SQL

2.3 CREATE Operation

- **CREATE** Befehl erzeugt Knoten und Kanten
- **CONTSTRAINT+UNIQUE** Befehl erzeugt Regel bzgl. Einzigartigkeit

```
1
2   CREATE (:Kurs {kurs_nr: 'G08', titel: 'Grundlagen I'});
3   CREATE (:Kurs {kurs_nr: 'G10', titel: 'Grundlagen II'});
4   CREATE (:Kurs {kurs_nr: 'P13', titel: 'C-Programmierung'});
5   CREATE (:Kurs {kurs_nr: 'I09', titel: 'Datenbanken'});
6
7   CREATE CONSTRAINT Kurs_kurs_nr IF NOT EXISTS
8   FOR (kurs:Kurs) REQUIRE kurs.kurs_nr IS UNIQUE;
9
```

Knoten und Constraints erstellen

```
1
2   MATCH (angebot:Angebot {ang_nr: 2, kurs_nr: 'G08'})
3   MATCH (kursleiter:Kursleiter {pers_nr: 38197})
4   CREATE (kursleiter)-[:FUEHRT_DURCH]->(angebot), (angebot)-[:WIRD_GELEITET_VON]->(kursleiter);
5
```

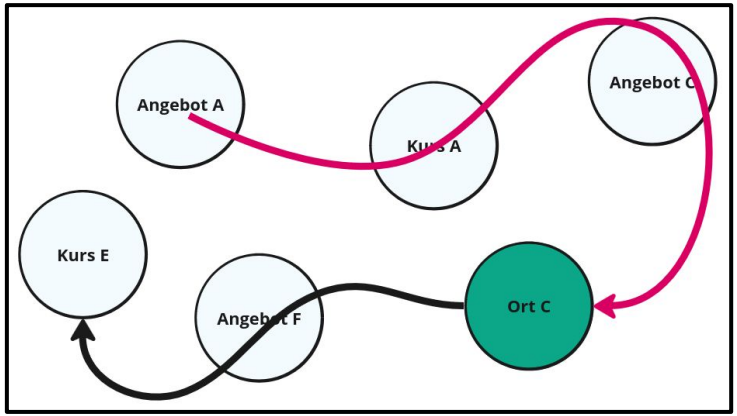
Kanten erstellen

2.3 CREATE Operation

- `INDEX` Befehl erzeugt Index
- `O(log n)` statt `O(n)`

```
1
2   CREATE INDEX Ort_name_index IF NOT EXISTS
3   FOR (ort:Ort) ON (ort.name);
4
5   CREATE INDEX Angebot_kurs_nr_index IF NOT EXISTS
6   FOR (angebot:Angebot) ON (angebot.ang_nr);
7
8   CREATE INDEX Angebot_ort_index IF NOT EXISTS
9   FOR (angebot:Angebot) ON (angebot.ort);
10
```

Erstellen von Indizes



Durchlauf mit und ohne Index

2.3 READ Operation

- Verknüpfung zweier Knoten über die Kante mit **`MATCH`**
- Ausgabewerte müssen im Return definiert sein
- Optional: **`WHERE`**-Bedingung, **`UNION`**, **`ORDER BY`**, Aggregationsfunktionen wie **`COUNT`**
- **`OPTIONAL MATCH`** vs. **`MATCH`**

→ intuitive und relativ schnelle Abfragen, bei Funktionen performancelastiger

```
1
2   MATCH (k:Kurs)-[:HAT_VORAUSSETZUNG]->(kv:Kurs)
3   RETURN k.titel AS kurs, collect(kv.titel) AS voraussetzung
4   UNION
5   MATCH (k:Kurs)
6   WHERE NOT(k)-[:HAT_VORAUSSETZUNG]->(:Kurs)
7   RETURN k.titel AS kurs, NULL AS voraussetzung;
8
```

Auslesen der Kurse mit und ohne Voraussetzung

2.3 UPDATE Operation

```
1
2   CREATE (:Kursleiter {pers_nr: 27183, name: 'Meier, I.', gehalt: 4300.50});
3
4   MATCH (k:Kursleiter {pers_nr: 27183})
5   SET k.gehalt: 4800;
6
```

Cypher: Update eines Knoten-Attributs

```
1
2   UPDATE Kursleiter
3   SET gehalt = 4800
4   WHERE pers_nr = 27183;
5
```

SQL: Updaten eines Tabellen-Eintrags

2.3 UPDATE Operation

```
1
2   MATCH (a:Angebot)-[edge:FINDET_STATT_IN]->(o:Ort {name:'Wedel'})
3   MATCH (o)-[edge2:VERANSTALTUNGSORT_VON]->(a)
4   MATCH (change_o:Ort {name:'Augsburg'})
5   DELETE edge, edge2
6   CREATE (a)-[:FINDET_STATT_IN]->(change_o), (change_o)-[:VERANSTALTUNGSORT_VON]->(a);
7
```

Updaten von Kanten

Aktueller Weg:

1. Selektieren und temp. speichern der benötigten Knoten und Kanten per **`MATCH`**-Operator
2. Löschen der ausgewählten Kanten von den zuvor selektierten Knoten
3. Erstellung neuer Kanten für die ausgewählten Knoten

→ Es ist aktuell mit Cypher **nicht** möglich bereits vorhandene Kanten zu updaten

2.3 DELETE Operation

```
1
2   MATCH (k:Kurs {titel: 'C-Programmierung'})-[:HAT_LITERATUR]->(kl:KursLiteratur)
3   DELETE kl;
4
5   MATCH (a:Angebot)-[g:WURDE_GEBUCHT_VON]->(t:Teilnehmer)
6   MATCH (a)-[:ANGEBOT_KURS]->(k:Kurs)
7   WITH k, count(k.kurs_nr) AS anzahl
8   WHERE anzahl < 2
9   DETACH DELETE k;
10
```

Delete Operationen

- **DELETE**-Operator: Löscht Knoten oder Kanten
- **DETACH DELETE**-Operator: Löscht Knoten/Kanten sowie und abhängigen Elemente

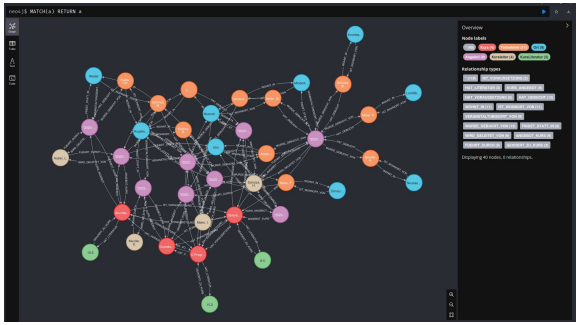
3. Fazit

- Cypher: Parallelen zu SQL

Leseoperationen vs. Schreiboperationen

Vor der Aufsetzung: Entwicklung der Datenbankstruktur

- Unterstützung durch graphische Darstellung in der Browseranwendung



- Aufsetzung viel (Schreib-)Aufwand

→ Wir können uns gut vorstellen, Neo4j in zukünftigen Projekten einzusetzen

Quellen

1. https://alban992346933.files.wordpress.com/2020/04/nosql.png
2. https://www.datenbanken-verstehen.de/lexikon/graphdatenbanken/
3. https://www.researchgate.net/figure/a-small-example-of-a-scale-free-graph-In-larger-variants-of-such-graphs-the-degree_fig4_262359440
4. https://www.flaticon.com/
5. https://memgraph.com/blog/graph-database-vs-relational-database
6. https://neo4j.com/docs/getting-started/cypher-intro/