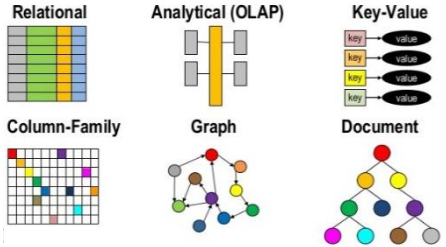
Hochschule für Technik und
Wirtschaft Dresden
University of Applied Sciences

1. NoSQL-Systeme: Überblick

Big Data Platforms

Prof. Dr.-Ing. Maik Thiele



NoSQL-Datenbanken

"I believe that **'one size does not fit all',** i.e. in every vertical market I can think of, there is a way to beat legacy relational DBMSs by 1-2 orders of magnitude."
Michael Stonebraker

Treiber Web-Entwicklung und Big Data

- Verschiedene Anwendungen erfordern unterschiedliche Typen von Datenbanken

Kriterien

- Kein relationales Datenmodell (Vermeidung impedance mismatch)
- Schemafrei oder schwache Schemarestriktionen (**V**ariety und **V**elocity)
  → gut für schreibende Transaktionen, schlecht für (komplexe) Anfragen
- Keine Joins → bewusste Denormalisierung
- **Horizontale Skalierbarkeit** dank Key-Value-Format (**V**olume), Scale-out statt Scale-up
- Keine Transaktionen → Konsistenzmodell **BASE** statt ACID
- Kein SQL → Zugriff per API / eigene Anfragesprache

RDBMS

NoSQL

NoSQL-Datenbanken (2)

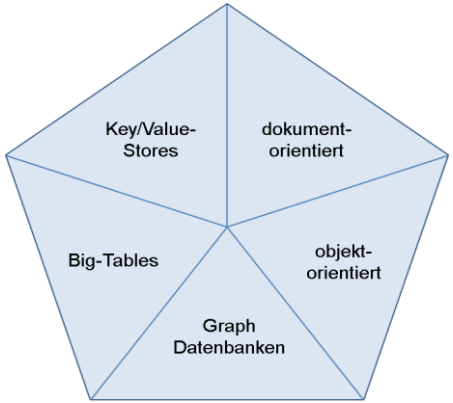
Schlanke, effiziente DBS, die auf eine performante und kostengünstige Verwaltung einfach strukturierter, sehr großer Datenmengen (Big Data) spezialisiert sind

Merkmale

- Im Wesentlichen Select- und Insert-Operationen
- Geringe Anforderungen an Konsistenz der DB

Vertreter

- Key Value / Tuple Store
- Wide Column Store / Column Families
- Document Store
- Graph Databases
- Ebenfalls: Object-Databases, XML Databases, Multi-Model Databases …



DB-Engines Rangliste

397 systems in ranking, October 2022

Rank Oct 2022	Rank Sep 2022	Rank Oct 2021	DBMS	Database Model	Score Oct 2022	Score Sep 2022	Score Oct 2021
1.	1.	1.	Oracle	Relational, Multi-model	1236.37	-1.88	-33.98
2.	2.	2.	MySQL	Relational, Multi-model	1205.38	-7.09	-14.39
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	924.68	-1.62	-45.93
4.	4.	4.	PostgreSQL	Relational, Multi-model	622.72	+2.26	+35.75
5.	5.	5.	MongoDB	Document, Multi-model	486.23	-3.40	-7.32
6.	6.	6.	Redis	Key-value, Multi-model	183.38	+1.91	+12.03
7.	7.	↑ 8.	Elasticsearch	Search engine, Multi-model	151.07	-0.37	-7.19
8.	8.	↓ 7.	IBM Db2	Relational, Multi-model	149.66	-1.73	-16.30
9.	9.	↑ 11.	Microsoft Access	Relational	138.17	-1.87	+21.79
10.	10.	↓ 9.	SQLite	Relational	137.80	-1.02	+8.43

https://db-engines.com/en/ranking

NoSQL Systems

NoSQL

Your Ultimate Guide to the Non-Relational Universe!

[including a historic Archive 2009-2020]

In April 2020, NoSQL-Database.org joined forces with hostingdata.co.uk. These two websites and the teams responsible for their creation will now work together.

The decision was made after the owners recognized that they have a common objective - helping people in the UK (and beyond) understand web hosting and all its intricacies, including NoSQL databases. This list is updated monthly. Learn more here.

- Further reading

NOSQL DEFINITION: Next Generation Database Management Systems mostly addressing some of the points: being **non-relational, distributed, open-source** and **horizontally scalable.**

The original intention has been **modern web-scale database management systems**. The movement began early 2009 and is growing rapidly. Often more characteristics apply such as: **schema-free, easy replication support, simple API, eventually consistent / BASE** (not ACID), a **huge amount of data** and more. So the misleading term "nosql" (the community now translates it mostly with "**not only sql**") should be seen as an alias to something like the definition above. [based on 7 sources, 15 constructive feedback emails (thanks!) and 1 disliking comment. Agree / Disagree? Tell us so! By the way: this is a strong definition and it is out there here since 2009!]

LIST OF NOSQL DATABASE MANAGEMENT SYSTEMS [currently >225]

http://www.nosql-database.org/

NoSQL ARCHIVE

NoSQL FORUMS

- Global NOSQL Forum »
- Forum Berlin »
- Forum France »
- Forum Japan »

NoSQL NEWS FEEDS

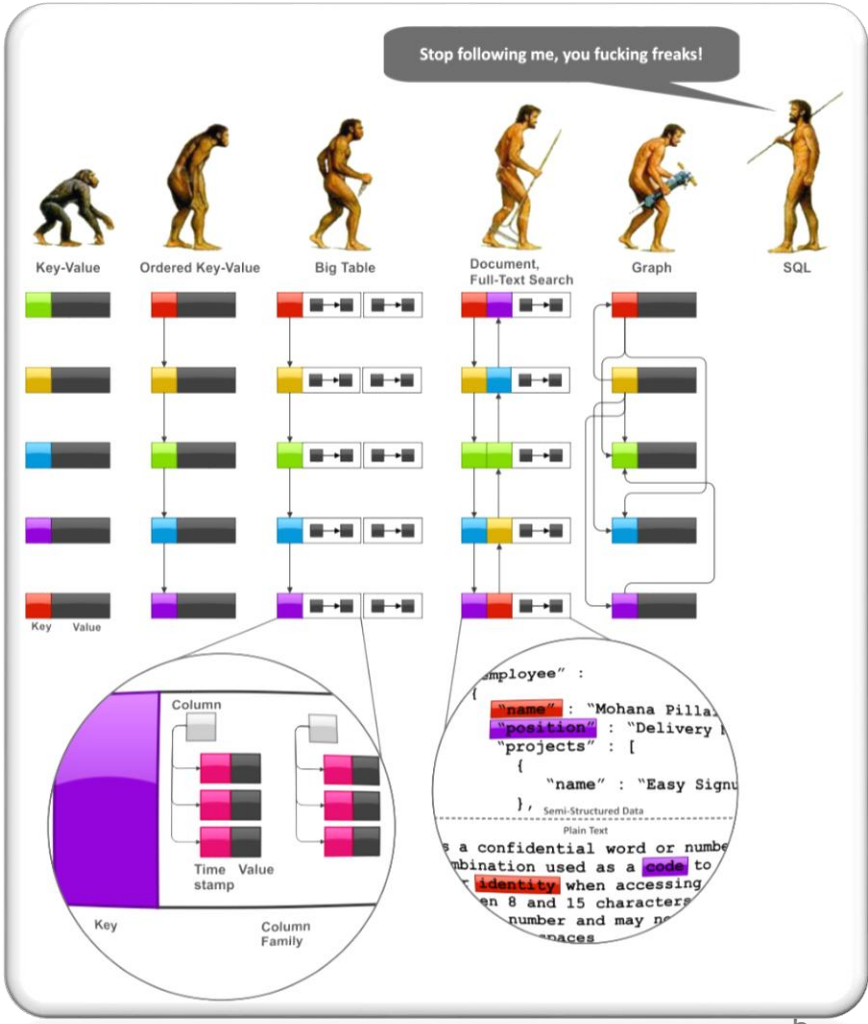
- MyNoSQL by Alex P »
- On Twitter: nosqlupdate »
- NoSQL Weekly » * new *
- HighScalability Blog »

Select the right Database Management System »

FUN

- A NoSQL parody »

"A DBA walks into a NOSQL bar, but turns



Wiederholung: Relationales Modell

Datenmodell

- Relationales Modell

Anfragesprache

- Relationale Algebra

Relation/Tabelle

- Menge von Tupeln mit der selben Semantik
- Tabelle besteht aus Zeilen und benannten Spalten (Attributen)
- Keine Duplikate vollständiger Zeilen erlaubt

Schema: spezifiziert Tabellenstruktur

- Datentypen (Domänen der Attribute)
- Konsistenz über Constraints
- Organisation und Optimierungen

4th Dimension
Adabas D
Alpha Five
Apache Derby
Aster Data
Amazon Aurora
Altibase
CA Datacom
CA IDMS
Clarion
ClickHouse
Clustrix
CSQL
CUBRID
DataEase
Database Management Library
Dataphor
dBase
Derby aka Java DB
Empress Embedded Database
EXASolution
EnterpriseDB
eXtremeDB
FileMaker Pro
Firebird
FrontBase
Google Fusion Tables
Greenplum
GroveSite
H2
Helix database
HSQLDB
IBM DB2
IBM Lotus Approach
IBM DB2 Express-C
Infobright
Informix
Ingres
InterBase
InterSystems Caché
LibreOffice Base
Linter
MariaDB
MaxDB
MemSQL
Microsoft Access
Microsoft Jet Database Engine
Microsoft SQL Server
Microsoft SQL Server Express
SQL Azure (Cloud SQL Server)
Microsoft Visual FoxPro
Mimer SQL
MonetDB
mSQL
MySQL
Netezza
NexusDB
NonStop SQL
NuoDB
Openbase
OpenLink Virtuoso
OpenLink Virtuoso Universal Server
OpenOffice.org Base
Oracle
Oracle Rdb for OpenVMS
Panorama
Pervasive PSQL
Polyhedra
PostgreSQL
Postgres Plus Advanced Server
Progress Software
RDM Embedded
RDM Server
R:Base
The SAS system
SAND CDBMS
SAP HANA
SAP Sybase Adaptive Server Enterprise
SAP Sybase IQ
SQL Anywhere
ScimoreDB
SmallSQL
solidDB
SQLBase
SQLite
SQream DB
Sybase Advantage Database Server
Teradata
TiDB
Tibero
TimesTen
Trafodion
txtSQL
Unisys RDMS 2200
UniData
UniVerse
Vectorwise
Vertica

Key-Value-Datenbanken

Key-Value-Datenbanken

Datenmodell

- Datenbank ist eine Sammlung von Key-Value-Paaren
- Unter einem eindeutigen Key / Schlüssel wird ein Wert gespeichert
- Key und Value enthalten Byte-Arrays = beliebige, serialisierte Datentypen (für value auch beliebig komplex)

Typische Basisoperationen

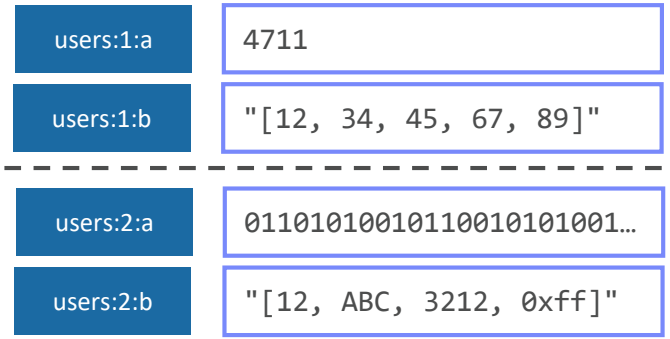
- APIs für CRUD-Operationen (create, read, update, delete)
  - `set (key, value)`
  - `value = get (key)`
  - `delete (key)`

Indexstrukturen

- Hash-Maps, B⁺-Bäume auf key

Skalierbarkeit

- Sharding: horizontale Partitionierung



DynamoDB
Azure Table Storage
Riak
Rdis
Aerospike
FoundationDB
LevelDB
Berkeley DB
Oracle NoSQL Database
GenieDb
BangDB
Chordless
Scalaris
Tokyo Cabinet/Tyrant
Scalien
Voldemort
Dynomite
KAI
MemcacheDB
Faircom C-Tree
LSM
KitaroDB
HamsterDB
STSdb
TarantoolBox
Maxtable
Quasardb
Pincaster
RaptorDB
TIBCO Active Spaces
Allegro-C
nessDB
HyperDex
SharedHashFile
Symas LMDB
Sophia
PickleDB
Mnesia
LightCloud
Hibari
OpenLDAP
Genomu
BinaryRage
Elliptics
Dbreeze
RocksDB
TreodeDB

Key-Value-Datenbanken: Beispielsysteme

Redis Data Types

- Redis ist kein einfacher KV-Store, sondern ein „data structure server“ mit persistentem Log (**appendfsync no/everysec/always**)
- **Key:** ASCII-Zeichenkette (max 512MB, z.B. comment:1234:reply.to)
- **Values:** strings, lists, sets, sorted sets, hashes (map of string-string), etc.

Redis APIs

- **SET/GET/DEL:** Einfügen eines Key-Value-Paars, Lookup eines Value über Key oder Löschen über Key
- **MSET/MGET:** Einfügen / Lookup mehrerer Schlüssel gleichzeitig
- **INCRBY/DECBY:** Inkrementieren/Dekrementieren von Zählern
- andere: `EXISTS`, `LPUSH`, `LPOP`, `LRANGE`, `LTRIM`, `LLEN`, etc.

Andere Systeme

- Riak, Aerospike, Voldemort, LevelDB, RocksDB, FoundationDB, Memcached

Document-Datenbanken

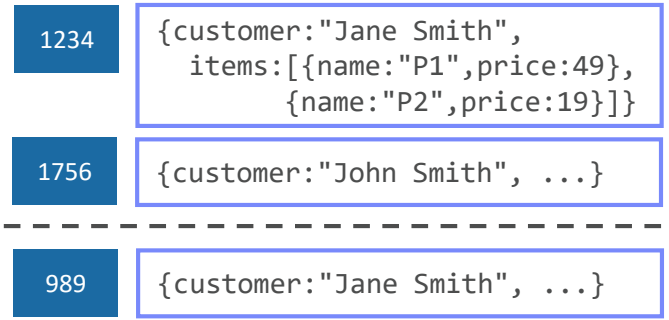
Document-Datenbanken

Motivation

- Anwendungsorientierte Verwaltung **strukturierter, semistrukturierter und unstrukturierte Daten**
- Skalierbarkeit mittels Sharding (horizontale Paritionierung)

Datenmodell

- Sammlung (collection) von (key, document)
- Format meist JSON, BSON
- Dokumente können beliebig strukturiert sein



Andere Systeme

- MongoDB, RethinkDB, Espresso, Amazon DocumentDB, CouchDB

AmisaDB
ArangoDB
BaseX
Cassandra
Cloudant
Clusterpoint
Couchbase
CouchDB
Densodb
Djondb
EJDB
Elasticsearch
eXist
FleetDB
iBoxDB
Inquire
JasDB
MarkLogic
MongoDB
MUMPS
NeDB
OrientDB
RaptorDB
RavenDB
RethinkDB
SDB
SisoDB
Terrastore
ThruDB

JSON (JavaScript Object Notation)

JSON Data Model

- Datenaustauschformat für **Semistrukturierte Daten**
- **Weniger komplex als XML** (besonders für Arrays)
- Populäres Format (z.B. Twitter)

```
{
  "students": [
    {
      "id": 1,
      "courses": [{"id": 12, "name": "DM"},
                  {"id": 9, "name": "AMLS"}]},
    {
      "id": 5,
      "courses": [{"id": 17, "name": "DIA"}]
    }
  ]
}
```

Anfragesprachen

- **Üblich: Bibliotheken** zum Traversieren von Bäumen und zur Datenextraktion
- **JSONiq:** XQuery-ähnliche Anfragesprache
- **JSONPath:** XPath-ähnliche Anfragesprache

Beispiel: MongoDB

Collection anlegen

```
import pymongo as m
conn = m.MongoClient("mongodb://localhost:27017")
db = conn["acme"]       # acme
cust = db["customers"]  # collection customers
```

In Collection einfügen

```
mdict = {
  "name": "Claire Grube",
  "address": "Hochschulstr. 12, Dresden"
}
id = cust.insert_one(mdict).inserted_id
```

Collection anfragen

```
print(cust.find_one({"_id": id}))

ret = cust.find({"name": "Claire Grube"})
for x in ret:
  print(x)
```

[Quelle: https://api.mongodb.com/python/current]

Beispiel: ElasticSearch

Neuen Index anlegen

```
PUT cars{
  "settings": {
    "index": {
      "number_of_shards": 1,
      "number_of_replicas": 0
}}}
```

Dokument einfügen

```
POST cars {
  "make" : "VW",
  "model" : "Sharan",
  "color" : "blue",
  "year" : 2018 }
```

Dokumente suchen

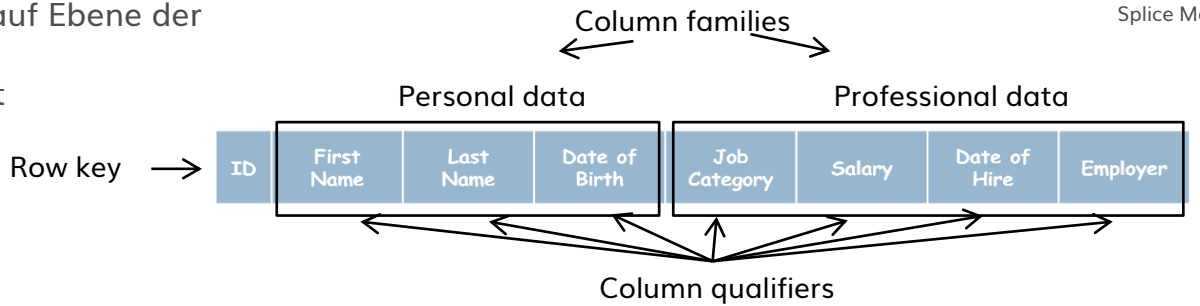
```
POST cars/_search {
"query": {
   "match_phrase": {
       "make": "VW" }
}}
```

Wide Column-Datenbank (Column Families)

Accumulo
Amazon SimpleDB
BigTable
Cassandra
Cloudata
Cloudera
Druid
Flink
Hbase
Hortonworks
HPCC
Hyupertable
KAI
KDI
MapR
MonetDB
OpenNeptune
Qbase
Splice Machine
Sqrrl

Datenmodell

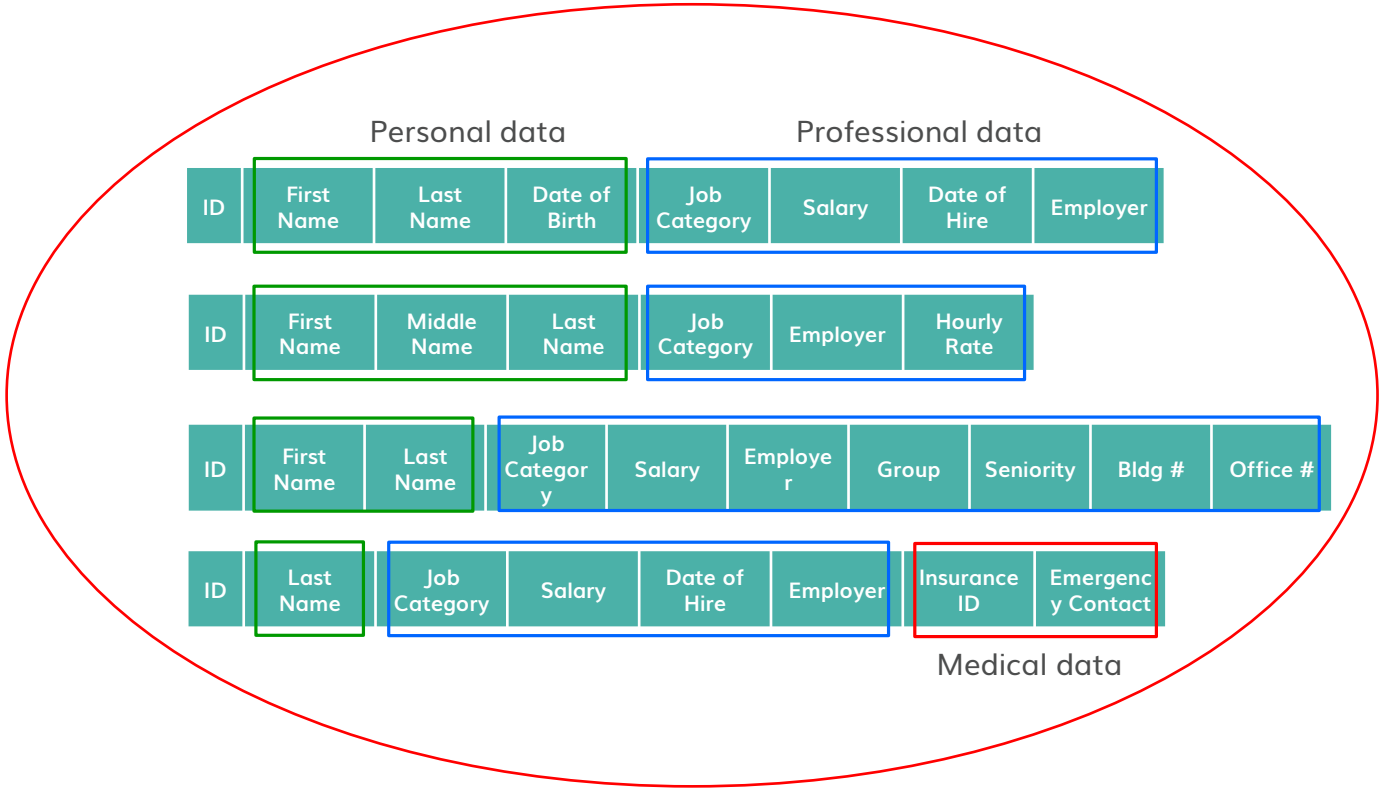
- Datenbank ist Sammlung von Key-Value-Paaren
- Key besteht aus drei Teilen: Row-key, Column-key und Zeitstempel (d.h. Version)
- Flexibles Schema: Menge der Spalten ist variabel
- Column Family
  - Zeilen gruppieren sich nach Spaltenfamilien
  - Werden zusammen abgefragt
  - Tuning (z.B. Kompression) auf Ebene der Column-Families möglich
  - Zur Entwurfszeit spezifiziert (ähnlich DDL)



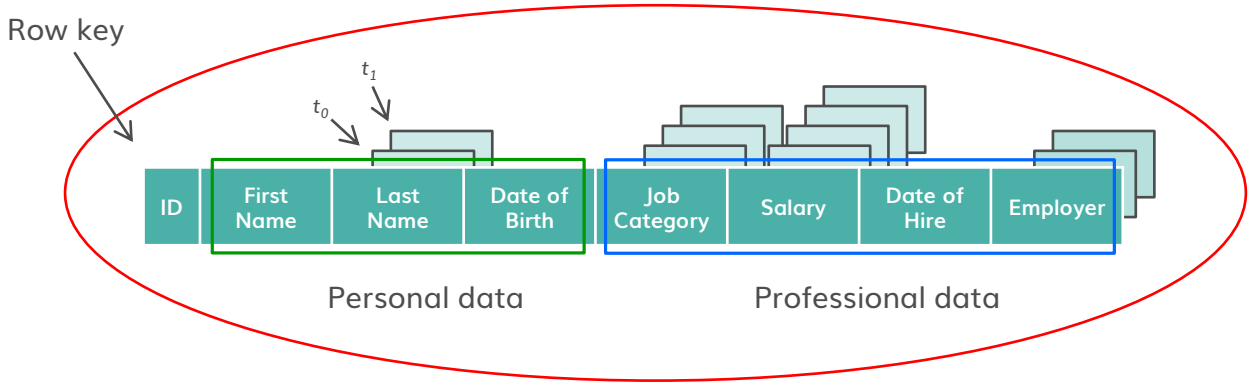
Andere Systeme

- Wide-column stores: Google BigTable, Apache HBase, Apache Cassandra

Wide Column-Datenbank



Wide Column-Datenmodell (2)

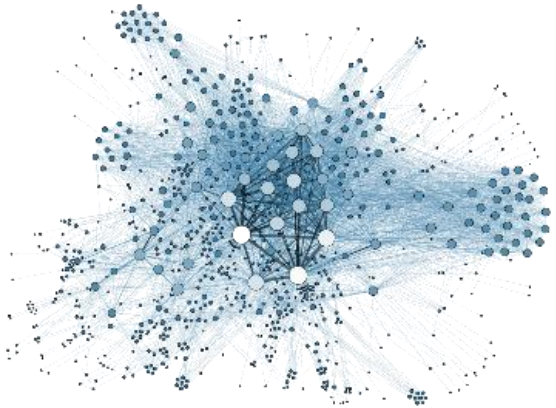


Eine „Zeile“

Eine „Zeile“ in einer Wide-column-Datenbank
=
Viele Zeilen in mehreren Tabellen in einer relationalen Datenbank

Graph-Datenbanken

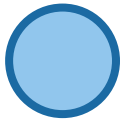
Graph-Datenbanken



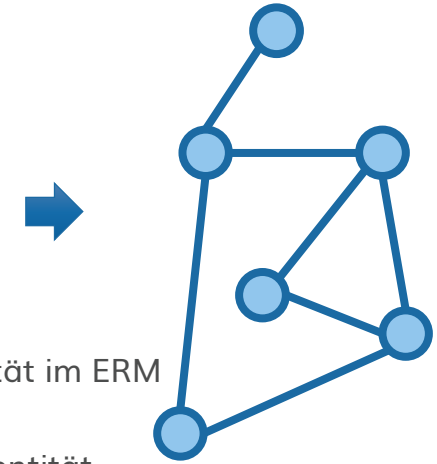
Anwendungsbeispiele

- Soziale Netzwerke, Open/Linked Data, Knowledge Bases, Bioinformatik
- Fragestellungen: Influencer-Analyse, Ranking, topologische Analysen

KNOTEN



- Ähnlich einer Entität im ERM
- Existieren für sich
- Besitzen Objektidentität



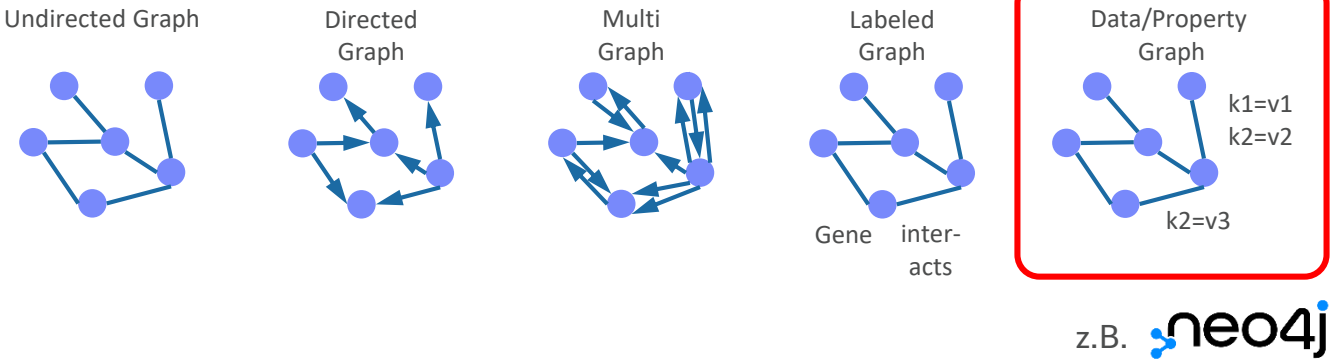
KANTEN

- Ähnlich einer Beziehung im ERM
- Existiert nur zwischen Knoten
- Identität durch die verbundenen Knoten bestimmt

Graph-Datenmodell

Vorherrschendes Datenmodell: Property Graph Model

- Gerichteter Graph
- Knoten und Kanten besitzen Eigenschaften (“labels”)



AllegroGraph
ArangoDB
Bigdata
Bitsy
BrightstarDB
DEX/Sparksee
Execom IOG
Fallen *
Filament
FlockDB
GraphBase
Graphd
Horton
HyperGraphDB
IBM System G Native Store
InfiniteGraph
InfoGrid
jCoreDB Graph
MapGraph
Meronymy
Neo4j
Orly
OpenLink virtuoso
Oracle Spatial and Graph
Oracle NoSQL Datbase
OrientDB
OQGraph
Ontotext OWLIM
R2DF
ROIS
Sones GraphDB
SPARQLCity
Sqrrl Enterprise
Stardog
Teradata Aster
Titan
Trinity
TripleBit
VelocityGraph
VertexDB
WhiteDB

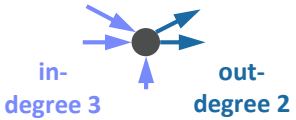
Terminologie und Metriken

Terminologie

- Graph G = (V, E) der Knotenmenge V und Kantenmenge E
- **Pfad:** Sequenz von Kanten und Knoten (**walk:** Wiederholung von Kanten/Knoten mgl.)
- **Zyklus:** Geschlossener walk, d.h. Start und Ende an dem selben Knoten
- **Clique:** Subgraph bestehende aus Knoten die jeweils paarweise adjazent sind

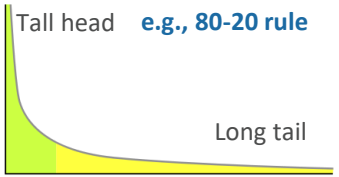
Metriken

- **Degree** (in/out-degree): Anzahl der eingehenden bzw. ausgehenden Kanten eines Knoten
- **Diameter:** Maximale Distanz aller Knotenpaare (längster kürzester Weg)

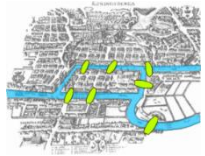


Power Law Distribution

- Grad der Knoten in den meisten realen Graphen sind nach dem Potenzgesetz verteilt

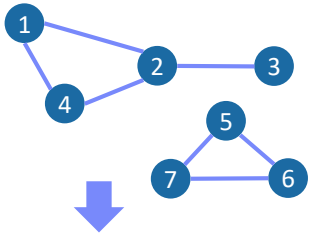


Knotenzentrierte Verarbeitung



Google Pregel

- Name: Königsberger Brückenproblem (Euler 1736)
- **"Think-like-a-vertex"** Verarbeitungsmodell
- Iterative Verarbeitung in "super steps", Kommunikation: message passing



Programmiermodell

- Graph als Kollektion von Knoten mit adjazente Kanten repräsentiert
- Implementierung von Algorithmen via Vertex API
- Algorithmus terminiert wenn alle Knoten stoppen (halt)
  → keine Nachrichten mehr

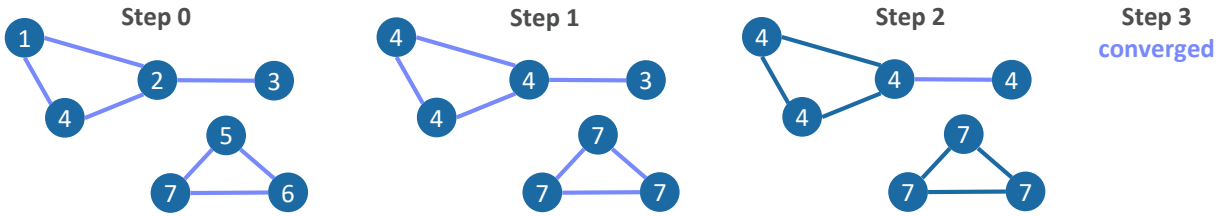
```
public abstract class Vertex {
  public String getID();
  public long superstep();
  public VertexValue getValue();

  public compute(Iterator<Message> msgs);
  public sendMsgTo(String v, Message msg);
  public void voteToHalt();
}
```

2	[1, 3, 4]	
7	[5, 6]	Worker 1
4	[1, 2]	
1	[1, 2, 4]	
5	[6, 7]	
3	[2]	Worker 2
6	[5, 7]	

Knotenzentrierte Verarbeitung (2)

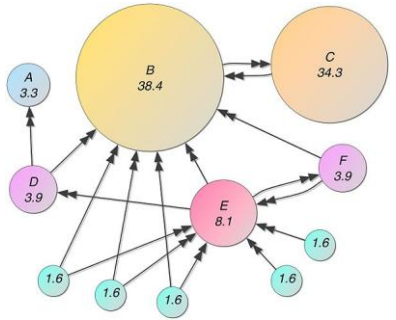
Beispiel 1: Zusammenhangskomponente (connected components)

- Zusammenhängende Subgraphen finden
- Propagiere `max(current, msgs) if != current to neighbors, terminate if no msgs`



Beispiel 2: Page Rank

- Ranking von Webseiten nach Wichtigkeit
- **#1:** Initialisiere Knoten mit 1/numVertices()
- **#2:** In jedem super step
  - Berechne aktuellen Knotenwert:
    `value = 0.15/numVertices()+0.85*sum(msg)`
  - Sende zu allen Nachbarn:
    `value/numOutgoingEdges()`



[**Quelle:** https://en.wikipedia.org/wiki/PageRank]

ACID versus BASE Transactions

Atomarität

- „Alles oder nichts“: wenn ein Teil der Transaktion scheitert, scheitert die gesamte Transaktion

Consistency

- Garantiert dass jedes Transaktion eine Datenbank von einem konsistenten Zustand in einen anderen konsistenten Zustand überführt

Isolation

- Garantiert dass die nebenläufige Ausführung von Transaktionen äquivalent zu ihrer seriellen Ausführung ist

Dauerhaftigkeit

- Sobald eine Transaktion ein Commit durchführt bleiben die Änderungen bestehen (auch im Fehlerfall)

BASE

- Basically Available (garantiert dank Replikation), Verfügbarkeit ist das wichtigste Kriterium
- Soft state (es ist Aufgabe der Anwendung, die Konsistenz zu gewährleisten)
- Eventually Consistent (die Datenbank wird auf längere Sicht konsistent sein, 'Veraltete' Daten sind OK)

Zusammenfassung

Datenmodellierung in „Big-Data“-Umgebungen

- “Data comes first!!!”
- Kein Möglichkeit der Festlegung eines logischen Datenmodells
  - „Daten sind einfach da“
  - Datenquelle ändern sich permanent

NoSQL

- Alternative Datenmodelle und Systemarchitekturen (mehrheitlich Scale-out)

Heterogene Datenmodelle

- Relationales Datenmodell, Key-Value-Modell, Document-Modell (JSON), Graph-Modell, Zeitreihen-Modell