

8. Big Data und NoSQL-Datenbanken

■ Big Data

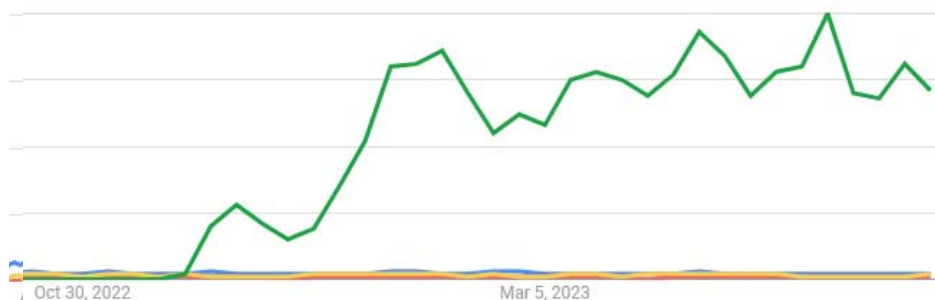
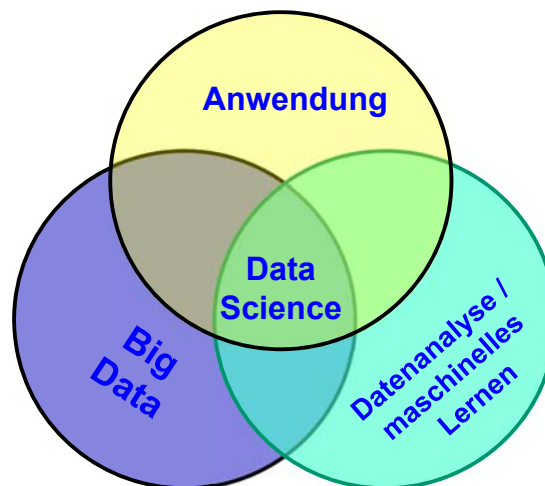
- Herausforderungen (V-Eigenschaften)
- Plattformen für Big Data Analytics (Hadoop, MapReduce, Spark/Flink)

■ NoSQL-Datenbanken

- Eigenschaften
- Document Stores (MongoDB)
- Graph-Datenbanken (neo4J)



Data Science



Google Trends

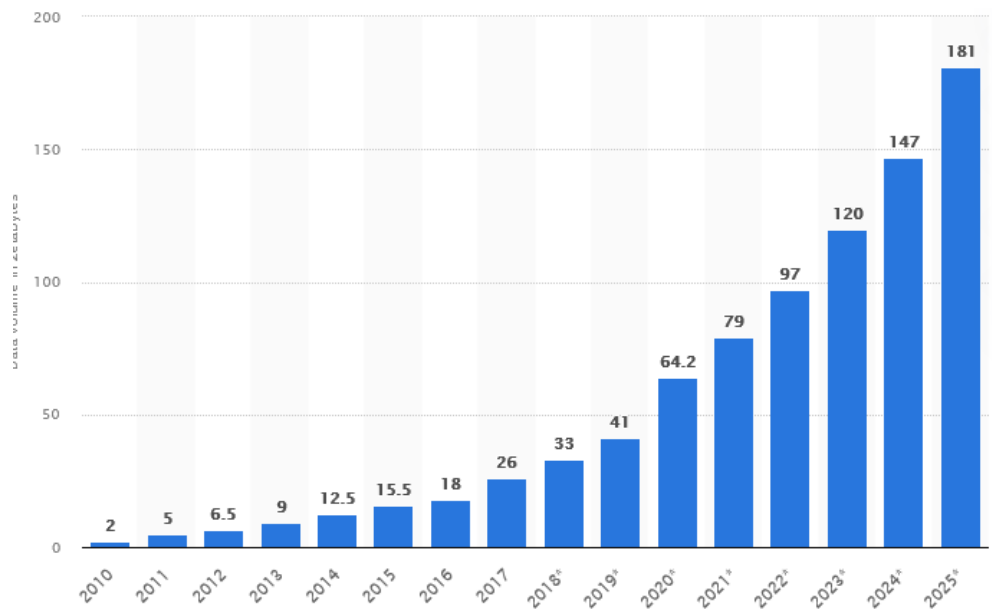
● machine learning ● Big Data ● Data science ● chatGPT



Wie groß ist Big Data?

■ Big wandelt sich schnell

- Gigabytes,
- Terabytes (10^{12}),
- Petabytes (10^{15}),
- Exabytes (10^{18}),
- Zettabytes (10^{21}),
- Yottabytes (10^{24}),
- Brontobytes (10^{27}),
- ...



geschätztes Datenaufkommen in Zettabytes

Quelle: Statista



Welche Art von Daten?

■ Unternehmensdaten

- strukturierte Daten (Datenbanken, Data Warehouses) mit Angaben zu Personal, Kunden, Bestellungen, Rechnungen ...
- Dokumente, E-Mails

■ Web-Daten

- Web-Seiten + Multimedia-Inhalte (Videos, Fotos, ...)
- Click Logs

■ soziale Netzwerke / Kommunikationsplattformen (Facebook, Twitter, LinkedIn, ...)

- Nutzer, Beziehungen, Nachrichten, Multimedia-Inhalte

■ Sensor-Daten / eingebettete Systeme

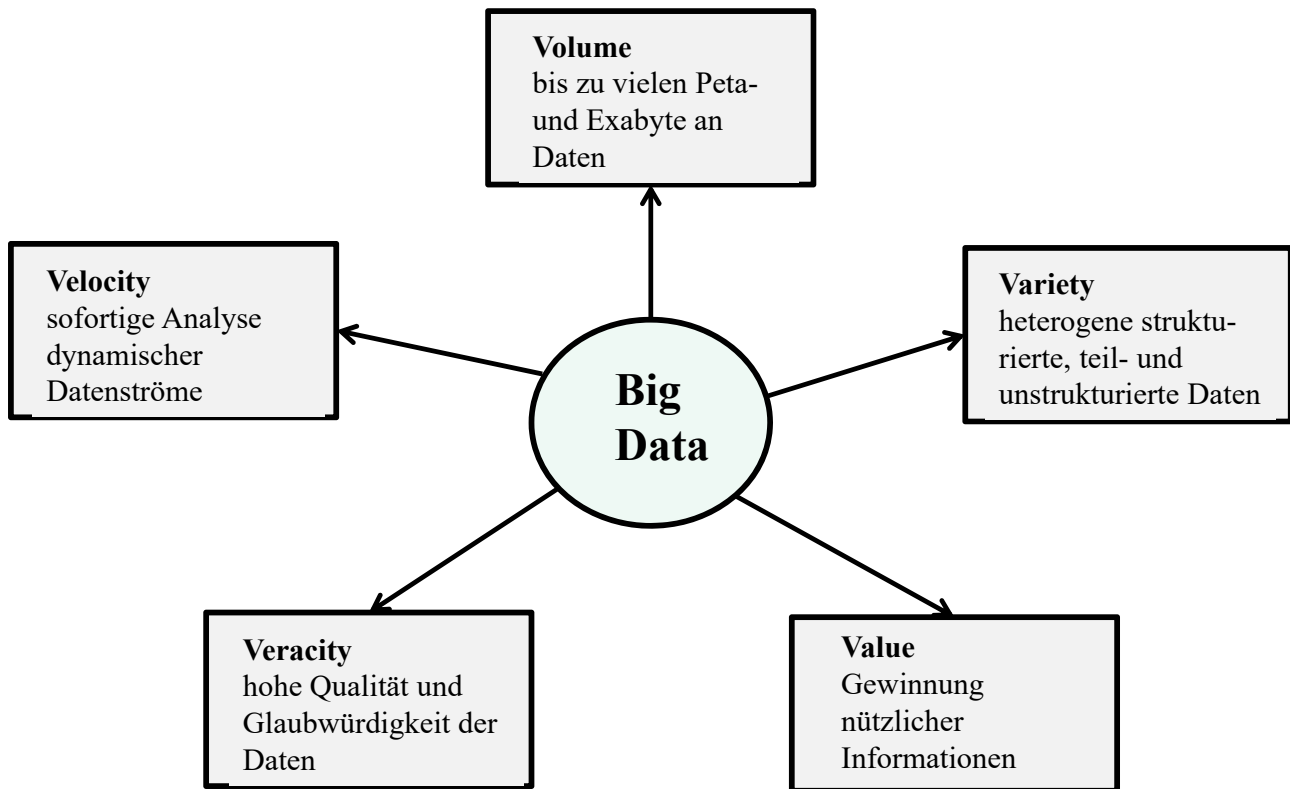
- vernetzte Produktion/Fertigung (Industrie 4.0)
- „intelligentes“ Haus, Verkehrsüberwachung, ...

■ umfangreiche und unterschiedliche wissenschaftliche Daten

- z.B. in Klimaforschung, Experimentalphysik, Lebenswissenschaften/Medizin (genetische Daten, Patientendaten mit Röntgen/MRT-Aufnahmen etc.), Sozialwissenschaften (Digital Humanities)



Big Data Challenges



Potenziale für Big-Data/ML

- Daten sind Produktionsfaktor: „data is the new oil“
 - ähnlich Betriebsmitteln und Beschäftigten
 - essenziell für viele Branchen und Wissenschaftsbereiche
- valide Grundlage für zahlreiche Entscheidungsprozesse
 - Vorhersage/Bewertung/Kausalität von Ereignissen
- verbesserte Analysen / Vorhersagen in zahllosen Anwendungen und Domänen
 - intelligente Werbung und Empfehlungen, Analyse/Optimierung von Produktionsprozessen, Lieferketten, etc.
 - Entdecken krimineller Aktivitäten (Kreditkartenmissbrauch ...)
 - verbesserte/personalisierte Therapien ...



Plattformen für Big Data Analytics

- massiv skalierbare Cloud-Architekturen (Shared Nothing)
- Frameworks zur automatischen Parallelisierung datenintensiver Aufgaben (Hadoop, MapReduce, Apache Spark/Flink)



- Parallelverarbeitung auf unterschiedlichen Stufen
 - Data Center (Cluster)
 - Multi-core server
 - Spezialprozessoren: GPUs /FPGAs
- In-Memory Datenbanken / Data Warehouses
- Spezialsysteme für Streaming-Daten, Graph-Daten, ...



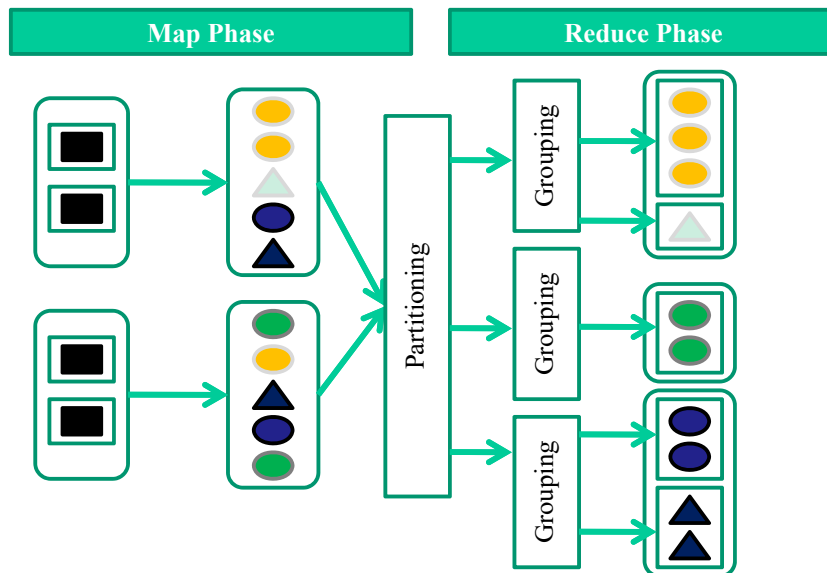
MapReduce

- Framework zur automatischen Parallelisierung von Auswertungen auf großen Datenmengen
 - Entwicklung bei Google
 - populäre Open-Source-Implementierung: Hadoop (seit 2006)
- Nutzung v.a. zur Verarbeitung riesiger Mengen teilstrukturierter Daten in einem verteilten Dateisystem
 - *Konstruktion Suchmaschinenindex*
 - *Clusterung von News-Artikeln*
 - *Spam-Erkennung ...*
- Fokus auf Batch-Verarbeitung

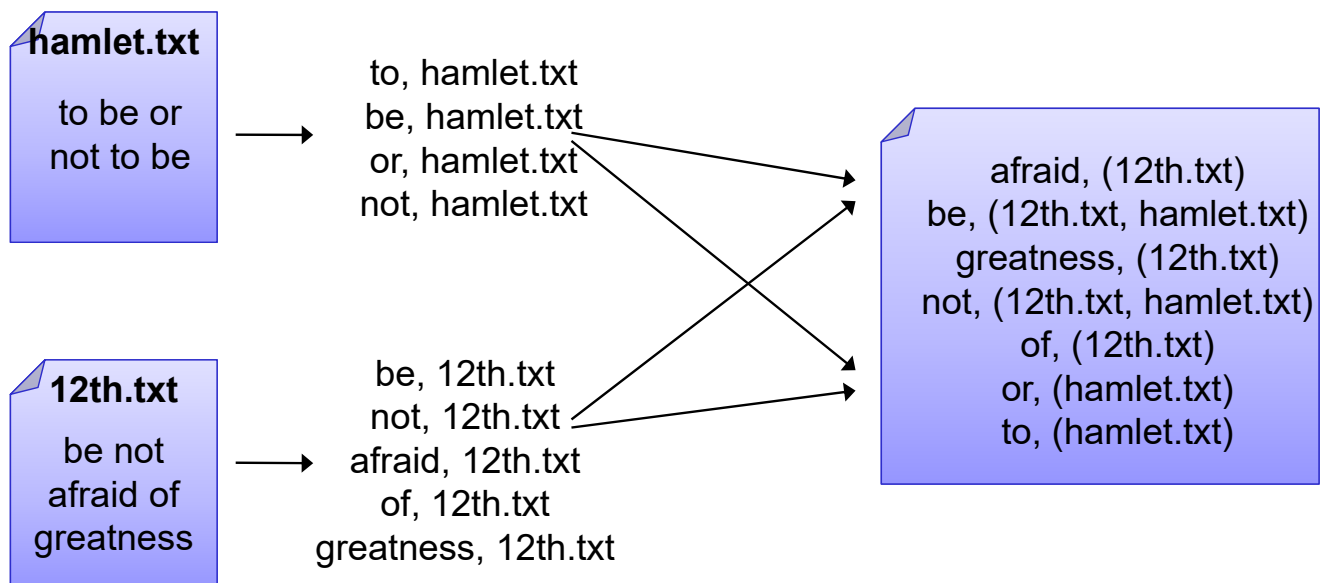


MapReduce

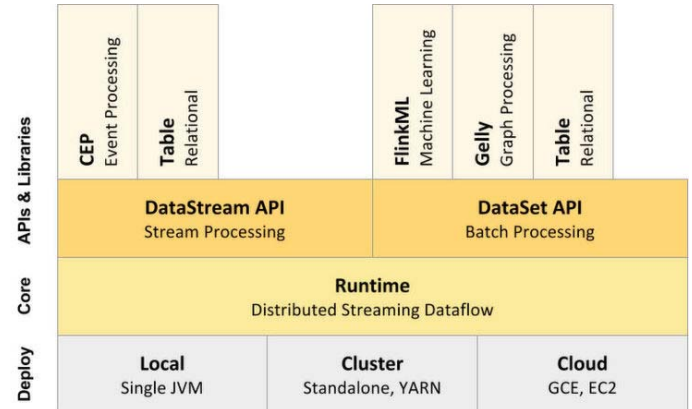
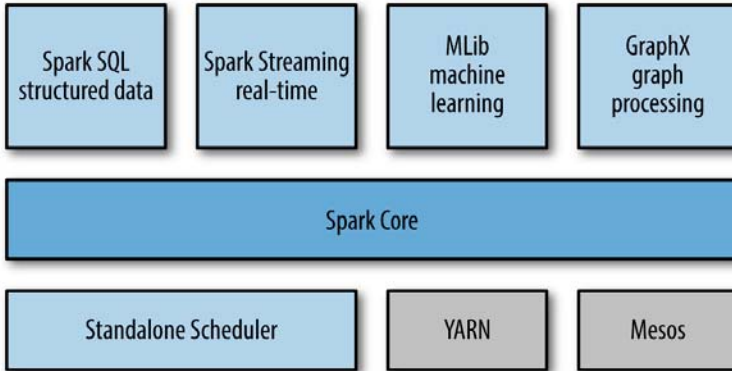
- Verwendung zweier Funktionen: **Map** und **Reduce**
- **Map**-Anwendung pro Eingabeobjekt zur Erzeugung von Key-value Paaren
 - jedes Key-Value-Paar wird einem Reduce-Task zugeordnet
- **Reduce**-Anwendung für jede Objektgruppe mit gleichem Key



MR-Beispiel: Generierung Text-Index



Spark vs Flink



- viele speziellere Frameworks z.B. für parallele Streamverarbeitung (Kafka, Storm, Samza ...)



8. Big Data und NoSQL-Datenbanken

- Einleitung Big Data
 - Herausforderungen
 - Analyse-Pipeline
 - Einsatzbereiche
- Systemarchitekturen für Big Data Analytics
 - Hadoop, MapReduce, Spark/Flink
- NoSQL-Datenbanken
 - Eigenschaften
 - Document Stores (MongoDB)
 - Graph-Datenbanken (neo4J)
- parallele/verteilte Graphanalysen / Gradoop



Relationale Datenbanken: Pros and Cons

- universelle Verbreitung und auf absehbare Zeit ungefährdet für die meisten DB-Anwendungen
 - SQL: standardisierte, mächtige (deklarative) Query-Sprache
 - ACID
 - reife Technologie
 - automatische Parallelisierung ...
- schema-getrieben (“schema first”)
 - weniger geeignet für semi-strukturierte Daten
 - zu starr für irreguläre Daten, häufige Änderungen
- relativ hohe Kosten, v.a. für Parallele DBS (kein Open-Source System)
- Skalierbarkeitsprobleme für Big Data (Web Scale)
 - Milliarden von Webseiten
 - Milliarden von Nutzern von Websites und sozialen Netzen
- ACID / strenge Konsistenz nicht immer erforderlich



NoSQL-Datenbanken

- Entwicklung seit ca. 2009
 - ursprünglicher Fokus: moderne “web-scale” Datenbanken
- Merkmale
 - nicht-relational
 - verteilt,
 - open-source und
 - horizontal (auf große Datenmengen) skalierbar
- weitere Charakteristika:
 - schema-frei
 - Datenreplikation
 - einfache API
 - statt ACID nur BASE (**B**asic **A**vailability, **S**oft state, **E**ventually consistent)
- Koexistenz mit SQL
 - “NoSql” wird als “**N**ot **o**nly **S**ql“ interpretiert



Arten von NoSQL-Systemen

■ Key Value Stores

- Amazon Dynamo, Voldemort, Membase, Redis ...
- Speicherung eines Werts (z.B. BLOB) pro nutzer-definiertem Schlüssel bzw. Speicherung von Attribut/Wert-Paaren
- nur einfache key-basierte Lookup/Änderungs-Zugriffe (get, put)

■ erweiterte Record-Stores / Wide Column Store

- Google BigTable / Hbase, Cassandra, Accumulo ...
- tabellenbasierte Speicherung mit flexibler Erweiterung um neue Attribute

■ Dokument-Datenbanken

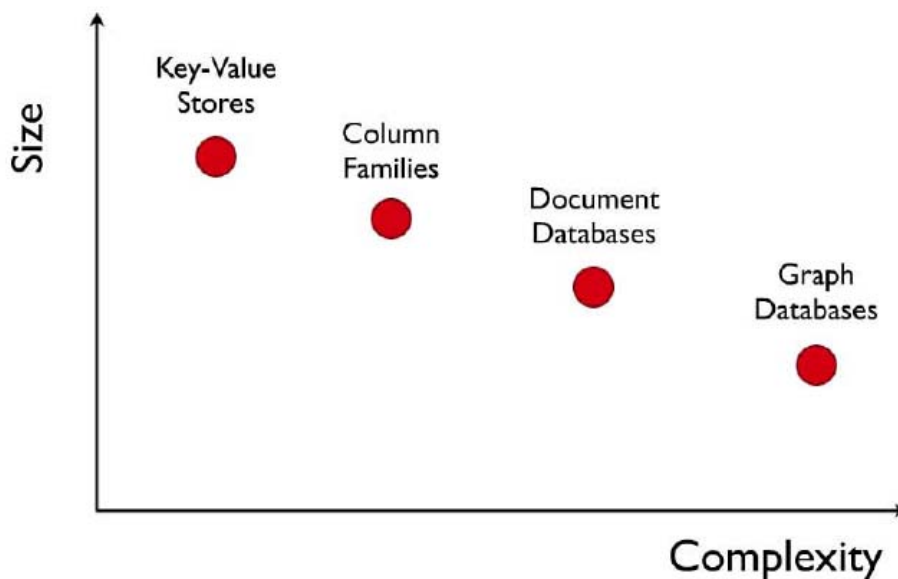
- CouchDB, MongoDB ...
- Speicherung semistrukturierter Daten als Dokument (z.B. JSON)

■ Graph-Datenbanken

- Neo4J, OrientDB ...
- Speicherung / Auswertung großer Graph-Strukturen



Grobeinordnung NoSQL-Systeme

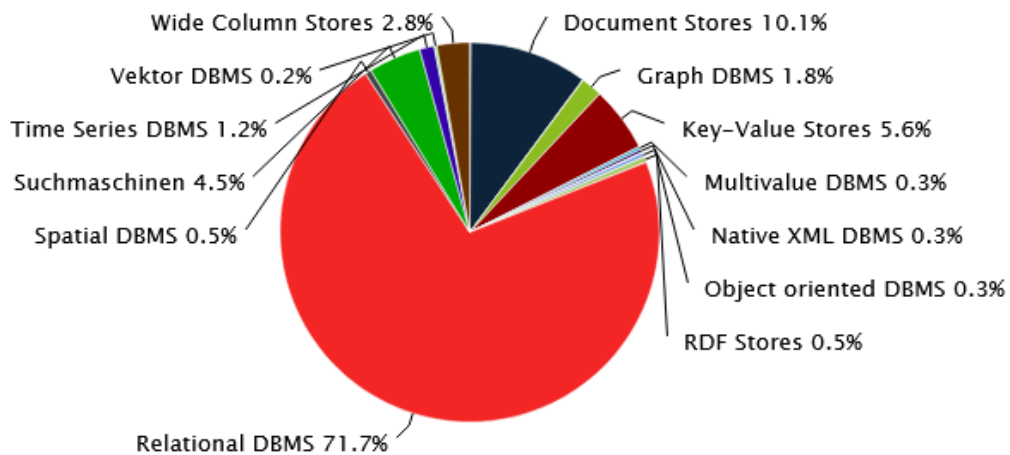


Rang			DBMS	Datenbankmodell	Punkte		
Jun 2023	Mai 2023	Jun 2022			Jun 2023	Mai 2023	Jun 2022
1.	1.	1.	Oracle +	Relational, Multi-Model	1231,48	-1,16	-56,27
2.	2.	2.	MySQL +	Relational, Multi-Model	1163,94	-8,52	-25,27
3.	3.	3.	Microsoft SQL Server +	Relational, Multi-Model	930,06	+9,97	-3,76
4.	4.	4.	PostgreSQL +	Relational, Multi-Model	612,82	-5,08	-8,02
5.	5.	5.	MongoDB +	Document, Multi-Model	425,36	-11,25	-55,36
6.	6.	6.	Redis +	Key-value, Multi-Model	167,35	-0,78	-7,96
7.	7.	7.	IBM Db2	Relational, Multi-Model	144,89	+1,87	-14,30
8.	8.	8.	Elasticsearch	Suchmaschine, Multi-Model	143,75	+2,11	-12,25
9.	↑ 10.	9.	Microsoft Access	Relational	134,45	+3,28	-7,36
10.	↓ 9.	10.	SQLite +	Relational	131,21	-2,65	-4,22
11.	11.	↑ 13.	Snowflake +	Relational	114,13	+2,41	+17,71
12.	12.	↓ 11.	Cassandra +	Wide column	108,55	-2,58	-6,90
13.	13.	↓ 12.	MariaDB +	Relational, Multi-Model	97,31	+0,44	-14,27
14.	14.	14.	Splunk	Suchmaschine	89,45	+2,81	-6,11
15.	15.	↑ 16.	Amazon DynamoDB +	Multi-Model	79,90	-1,20	-3,98
16.	16.	↓ 15.	Microsoft Azure SQL Database	Relational, Multi-Model	78,96	-0,23	-7,05
17.	17.	17.	Hive	Relational	75,52	+1,91	-6,06
18.	18.	↑ 24.	Databricks	Multi-Model	65,82	+1,87	+17,69
19.	19.	↓ 18.	Teradata	Relational, Multi-Model	62,64	-0,07	-7,76
20.	20.	↑ 23.	Google BigQuery +	Relational	54,64	-0,24	+5,57
21.	21.	↑ 22.	FileMaker	Relational	54,38	+2,39	+2,80
22.	22.	↓ 19.	Neo4j +	Graph	52,77	+1,66	-6,76
23.	23.	↓ 21.	SAP HANA +	Relational, Multi-Model	51,42	+1,06	-3,11
24.	24.	↓ 20.	Solr	Suchmaschine, Multi-Model	49,56	-0,20	-7,05
25.	25.	25.	SAP Adaptive Server	Relational, Multi-Model	43,76	+0,67	-2,91

<https://db-engines.com/de/ranking>



Rankingpunkte pro Kategorie in Prozent, Juni 2023



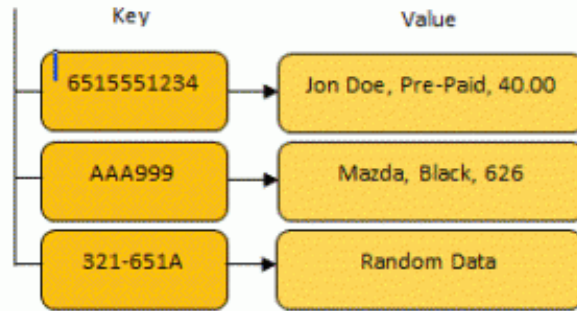
© 2023, DB-Engines.com



Key-Value Stores

■ Speicherung von Schlüssel-Werte-Paaren

- im einfachsten Fall bleiben Werte systemseitig uninterpretiert (BLOBs)
- flexibel, kein Schema
- günstig für wenig strukturierte Inhalte bzw. stark variable Inhalte, z.B. Twitter-Nachrichten, Webseiten etc.
- keine Verwaltung von Beziehungen



■ schnelle Lese/Schreibzugriffe über Schlüssel

- put (key, value)
- get (key)

■ keine komplexen Queries (z.B. Bereichsabfragen)



Erweiterbare Record Stores

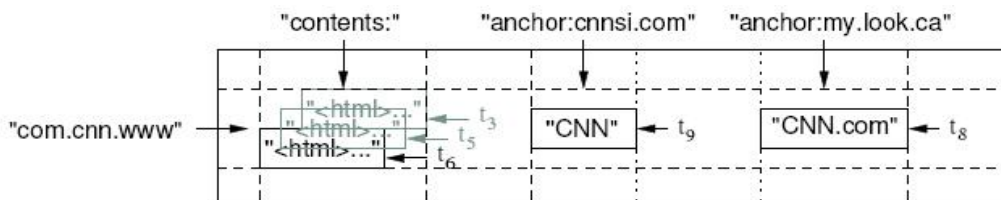
■ spaltenorientierte Key-Value Stores mit Erweiterbarkeit über Spaltenfamilien

■ Vorbild: Google BigTable

- Clones: HBase, Cassandra, ...

■ Ziele (BigTable):

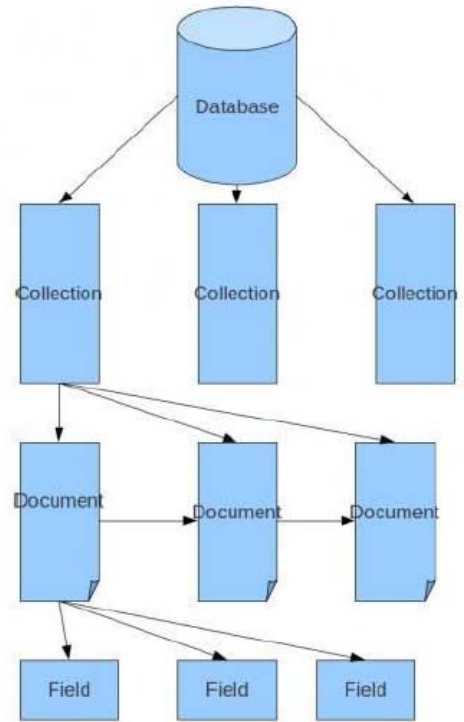
- Milliarden von Zeilen, Millionen von Spalten, Versionierung (z.B. für Web-Seiten)
- einfache Erweiterbarkeit um Spalten (innerhalb vordefinierter *Spaltenfamilien*)
- mehrdimensionale Keys: Zeilen- und Spalten-Schlüssel, Versionsnr



Row Key	Time Stamp	Column <i>contents</i>	Column Family <i>anchor</i>	
"com.cnn.www"	T9		anchor:cnnsi.com	CNN
	T8		anchor:my.look.ca	CNN.COM
	T6	"<html>.. "		
	T5	"<html>.. "		



- **Dokumenten-Store** mit bereits hoher Verbreitung
 - open source-Version verfügbar
 - JSON-Dokumente, gespeichert als BSON (Binary JSON)
- DB besteht aus Kollektionen von Dokumenten
- einfache Anfragesprache MQL
 - Indexierung von Attributen möglich
 - Map/Reduce-Unterstützung
- Skalierbarkeit und Fehlertoleranz
 - Skalierbarkeit durch horizontale Verteilung der Dokumentkollektionen unter vielen Knoten (“Sharding”)
 - automatische Replikation mit Konsistenzwahrung
- zunächst kein ACID
 - Änderungen nur bzgl einzelner Dokumente atomar



Beispiel: relational vs. dokumentenorientiert

Relational

STUDENTS	
sno	name
1	Peter
2	Tom

POSTS		
postID	topic	pdate
p1	NoSQL	01-09-2022
p2	RelationalDB	02-09-2022

COMMENTS				
commentID	postID	sno	content	cdate
c1	p1	1	Excellent	02-09-2022
c2	p1	2	I do not understand.	03-09-2022
c3	p1	1	This is a good idea!	04-09-2022

MongoDB (JSON)

```
{topic: 'NoSQL',
pdate: 01-09-2022,
comments : [{name: 'Peter',
content: 'Excellent',
cdate: 02-09-2022},
{name: 'Tom',
content: 'I do not understand.',
cdate: 03-09-2022},
{name: 'Peter',
content: 'This is a good idea!',
cdate: 04-09-2022}
]}
```

- geschachtelte Komponenten (ähnlich XML)
 - keine Beziehungen zwischen Dokumenten (-> keine Joins)
 - Redundanz bei n:m-Beziehungen
 - schemaflexibel



MongoDB: Operationen

RDB		MongoDB
SELECT * FROM students;	↔	db.students.find();
INSERT INTO students ...;	↔	db.students.insert(...);
UPDATE students ...;	↔	db.students.update(...);
DELETE FROM students ...;	↔	db.students.remove(...);
...	↔	...

■ Beispiele:

```
> db.students.insert({sno: 1, name: 'Peter'});
```

```
> db.posts.insert({topic: 'NoSQL', pdate: ...});
```

```
student_a={sno:731, name:'Irving Bonce', address:
```

```
{street:'38 Glenpark Ave',  
  city:'Dunedin',  
  phone:4535467}};
```

```
> db.students.find();
```

```
{_id:1, sno:731, name:'Irving Bonce', address:  
  {street:'38 Glenpark Ave',  
    city:'Dunedin',  
    phone:4535467}}
```

```
> db.students.save(student_a);
```



MongoDB: Operationen (2)

```
# find the student whose name is "Stephen Hong"
```

```
> db.students.find({name: 'Stephen Hong'});
```

```
# find all students whose last name is 'Hong'
```

```
> db.students.find({name: /Hong$/});
```

```
# find all students who live in Canberra
```

```
> db.students.find({address.city: 'Canberra'});
```

```
# count the number of students
```

```
> db.students.count();
```

```
# find all students who enrolled courses either "SP03" or "SP04"
```

```
> db.students.find({courses:{$in:['SP03','SP04']},  
  {sno: 1, name: 1}});
```

```
# find all students whose ages are below 20
```

```
> db.students.find({age:{$lt: 20}});
```



Graph-Datenbanken

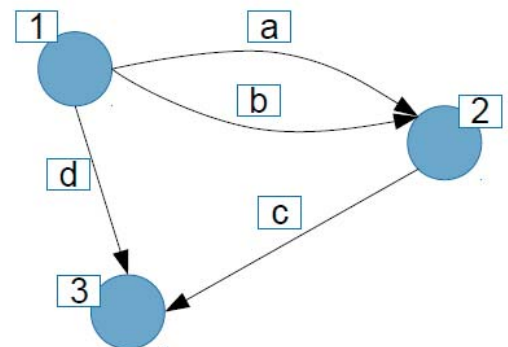
- bessere Unterstützung stark vernetzter Daten als mit Relationenmodell
 - soziale Netzwerke
 - Protein-Netzwerke
 - stark vernetzte Webdaten / Unternehmensdaten / ...
- Graph-Verwaltung im Relationenmodell oft nicht ausreichend schnell
 - viele Tabellen, viele Joins
 - langsame Traversierung von Kanten
 - langsame Umsetzung von Graph-Algorithmen
- Graph-Datenbanken
 - Graph-Datenmodell mit Gleichbehandlung von Entities und Relationships
 - Graph-Anfragesprachen
 - optimierte Graph-Operationen (z.B. finde „friends of friends“)



Neo4J



- native Graph-Datenbank von Neo Technology
 - Community Edition (open-source) + kommerzielle Varianten
 - in Java realisiert , Unterstützung weiterer Sprachen (Ruby, Python)
 - ACID-Unterstützung
 - Skalierbarkeit durch Datenreplikation
- zentrale Datenstruktur: **Property-Graphen**
 - Knoten/Kanten haben Label (Id bzw. Typ))
 - Knoten und Kanten können Properties haben
 - Property: Key-Value-Paar (Key ist vom Typ String)
 - gerichtete Kanten
- konstanter Aufwand zur Traversierung zu Nachbarknoten (statt Join im RM)
 -



■ Merkmale

- Pattern Matching
- OLTP-artige Lese/Änderungsoperationen
- schnelle Traversierungen
- ACID-basierte Updates

■ Klauseln:

- **START**: Starting points in the graph.
- **MATCH**: The graph pattern to match.
- **WHERE**: Filtering criteria.
- **RETURN**: What to return.
- **CREATE**: Creates nodes and relationships.
- **DELETE**: Removes nodes, relationships and properties.
- **SET**: Set values to properties.
- **FOREACH**: Performs updating actions once per element in a list.
- **WITH**: Divides a query into multiple, distinct parts.

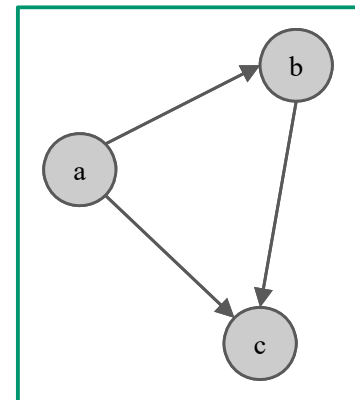
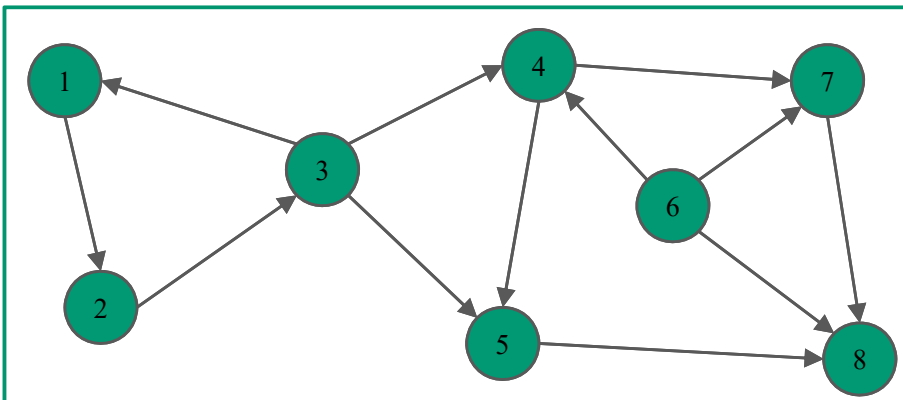


Cypher-Beispiel (1)

```
MATCH (a)-->(b)-->(c),
      (a)-->(c)
RETURN a,b,c
```

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

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

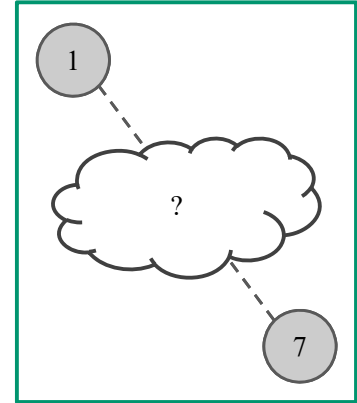
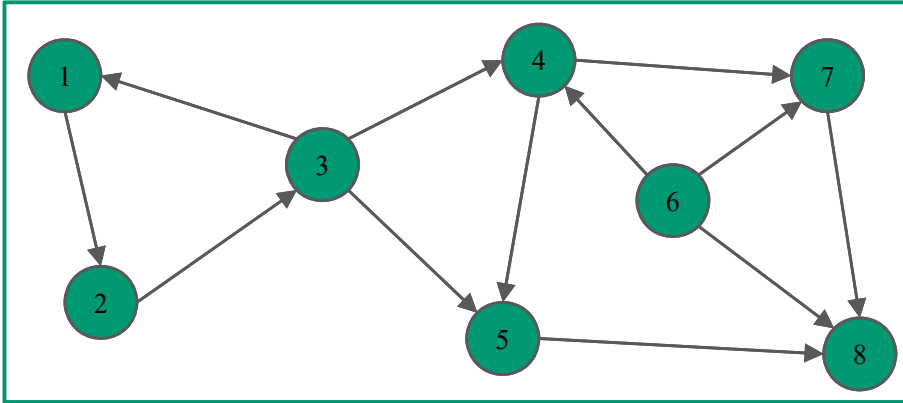


Cypher-Beispiel (2)

```

MATCH p=(a)-[*1..4]-(b)
WHERE a.id = 1 AND b.id = 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

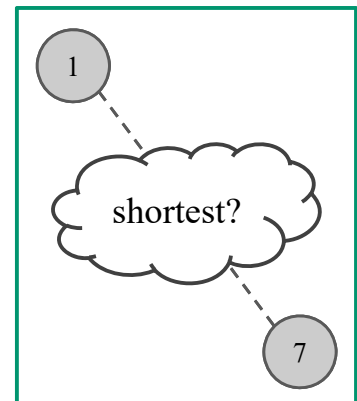
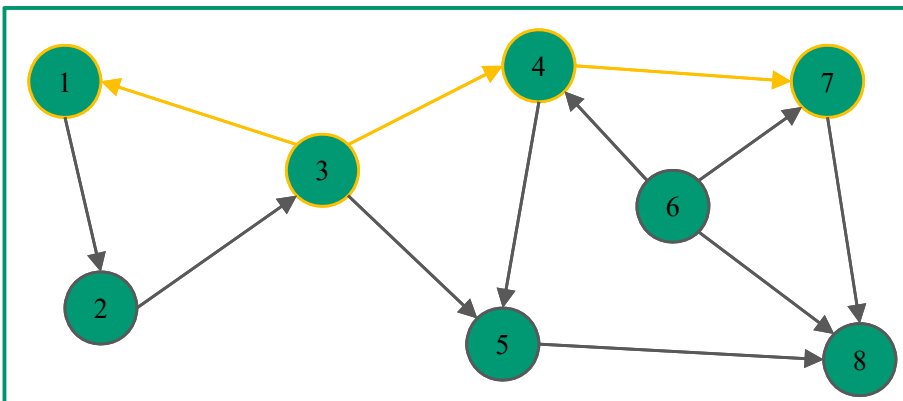


Cypher-Beispiel (3)

```

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

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

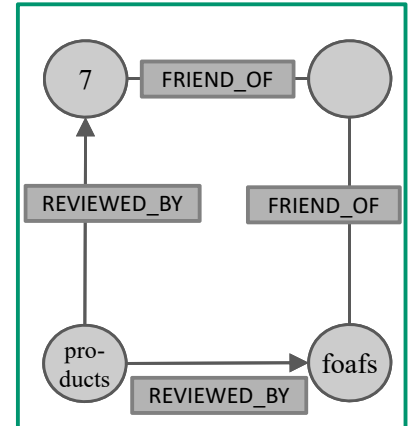
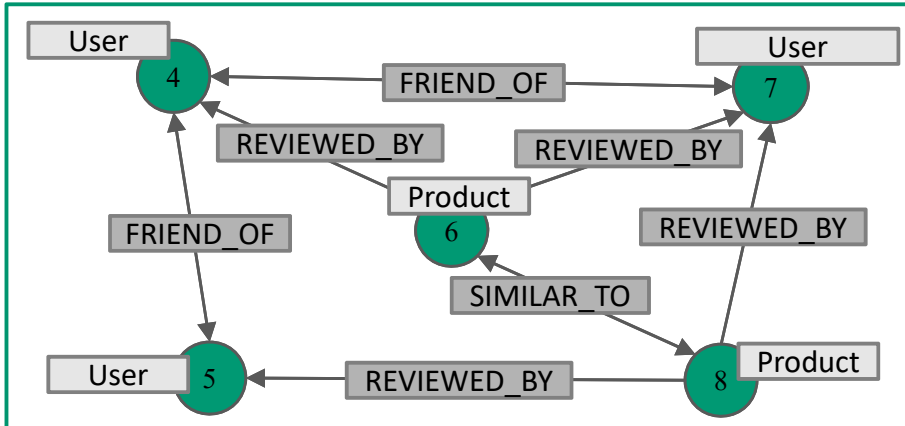


Cypher-Beispiel (4)

Freundesfreunde von Nutzern, die das selbe Produkt wie ein Nutzer bewertet haben

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

foafs	products
5	8



Zusammenfassung

- Big Data Herausforderungen
 - Volume, Variety, Velocity, Veracity, Privacy
 - skalierbare Analysen / Machine Learning
- skalierbare Cluster-Architekturen auf Basis von Hadoop mit Framework wie Apache Spark / Flink
- NoSQL
 - Auslöser: webskalierbares Datenmanagement
 - semistrukturierte schemafreie Daten
 - Verzicht auf SQL/ACID
- unterschiedliche Systemarten
 - Key/Value-Stores, erweiterte Record-Stores (Spaltenfamilien)
 - Dokumenten Stores
 - Graph-Datenbanken
- Hauptproblem für NoSQL: fehlende Standards

