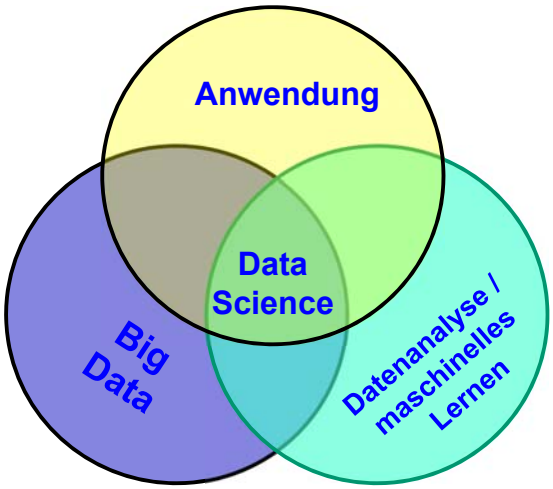
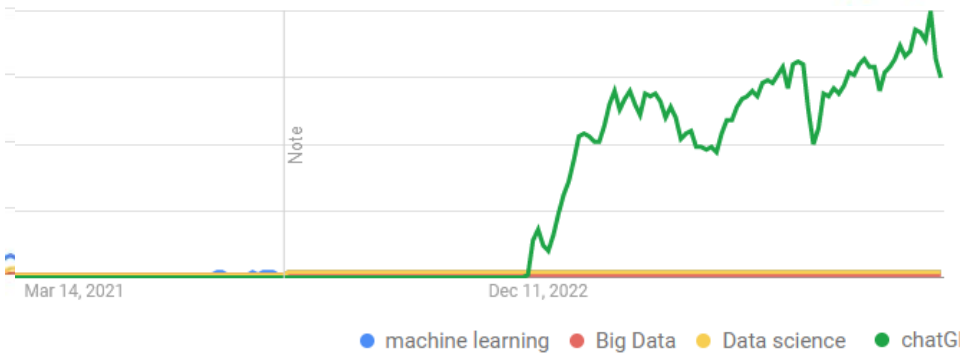
8. Big Data und NoSQL-Datenbanken

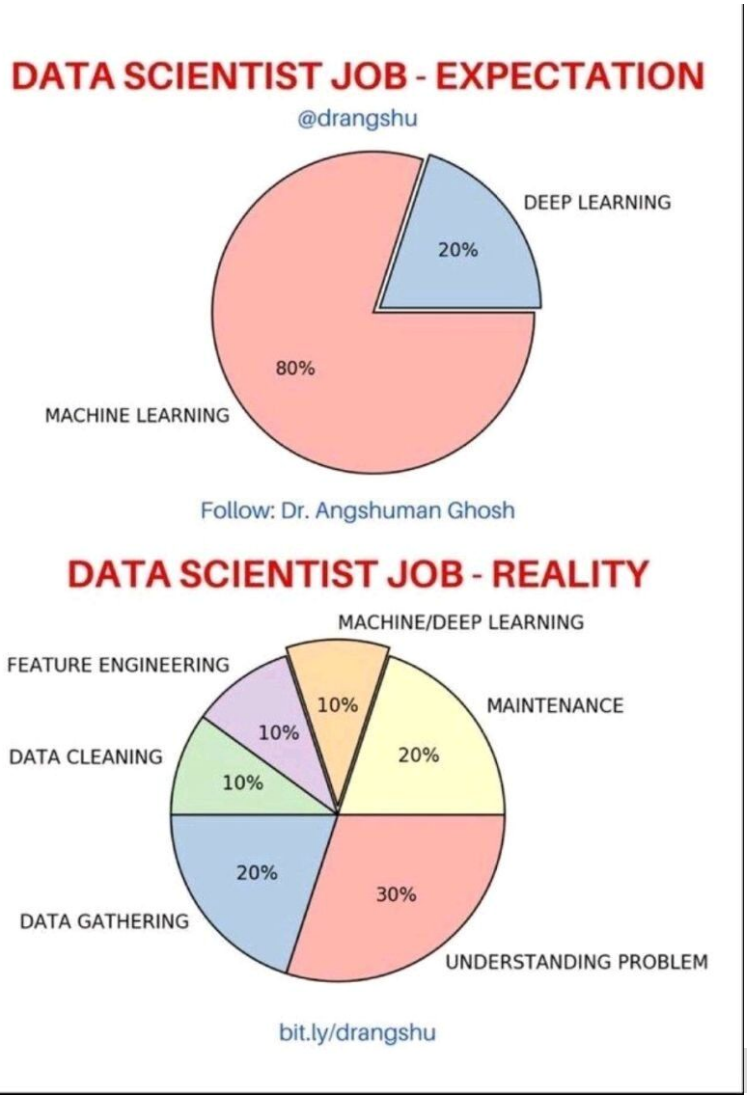
- Big Data
  - Herausforderungen (V-Eigenschaften)
  - Plattformen für Big Data Analytics (Hadoop, MapReduce, Spark/Flink)
- NoSQL-Datenbanken
  - Eigenschaften
  - Document Stores (MongoDB)
  - Graph-Datenbanken (neo4J)

Data Science





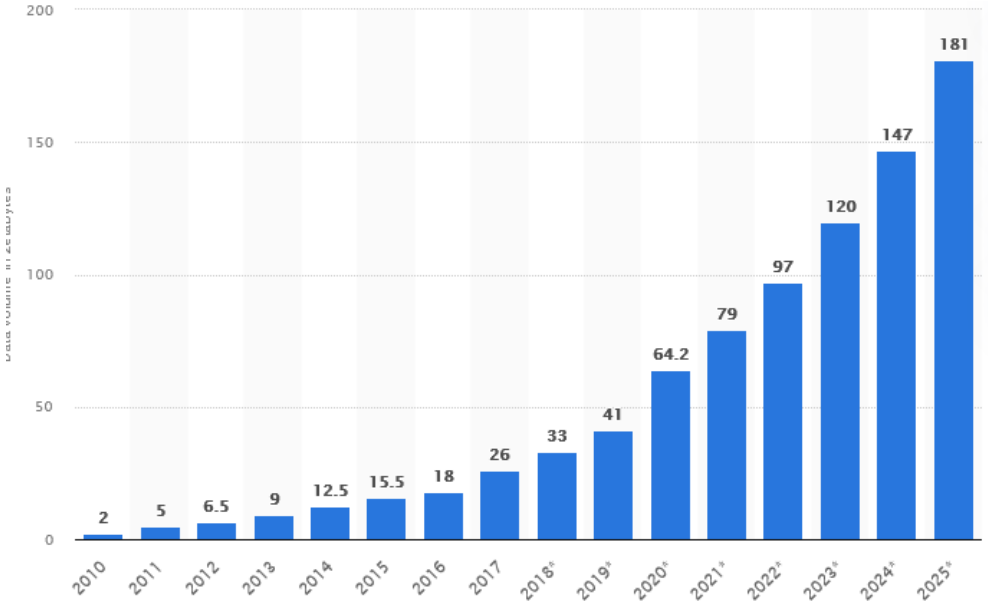
Google Trends



DBS 2

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**

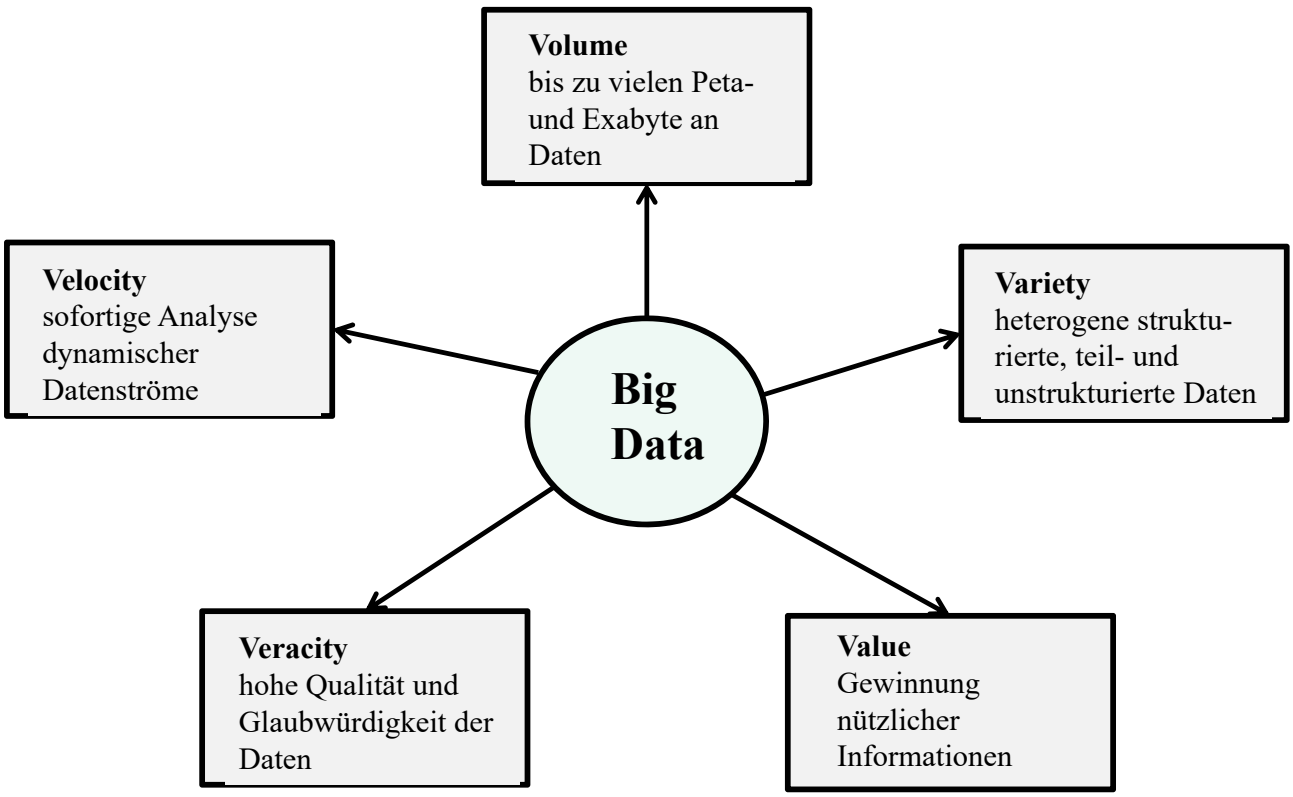
Quelle: Statista

DBS 2

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)

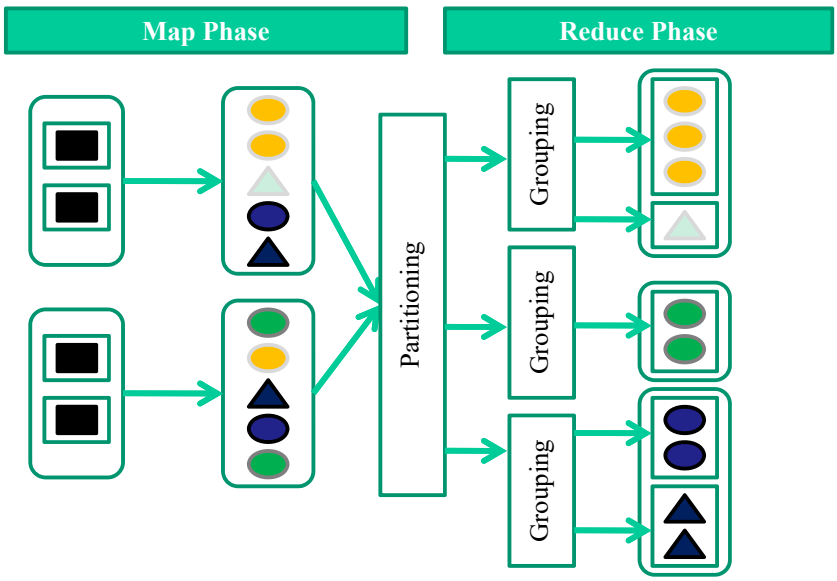
- Parallelverarbeitung auf unterschiedlichen Stufen
  - Data Center (Cluster)
  - Multi-core server
  - Spezialprozessoren: GPUs /FPGAs
- In-Memory Datenbanken / Data Warehouses
- Spezialsysteme für Streaming-Daten, Graph-Daten, …

MapReduce

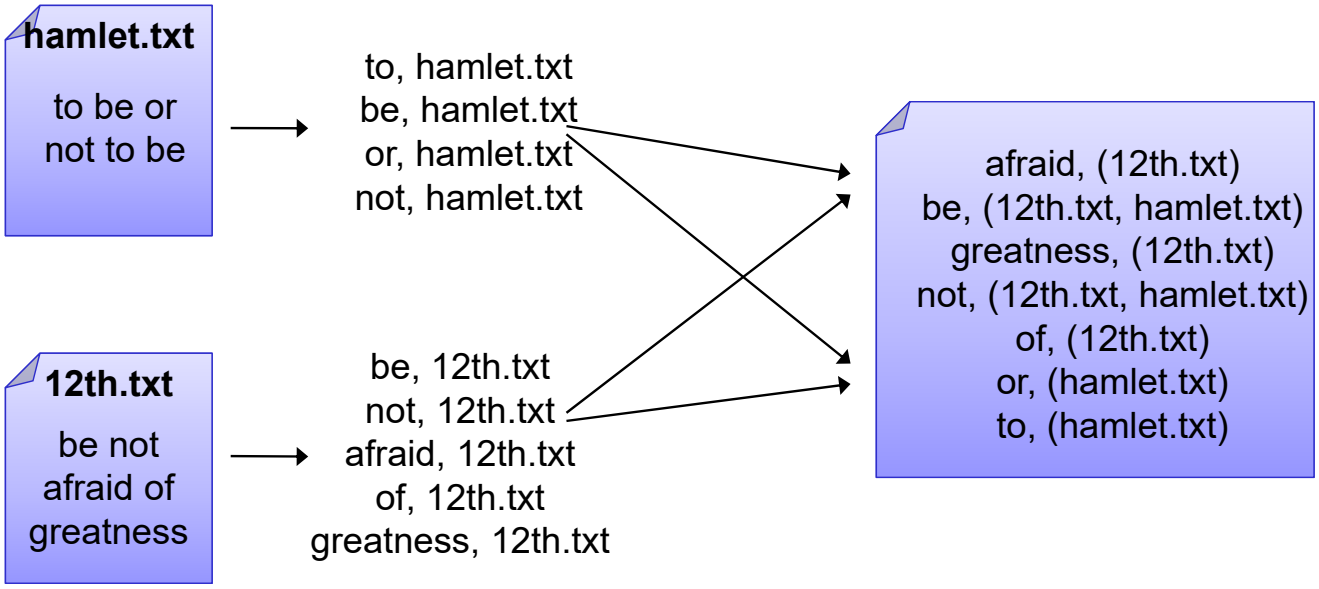
- Framework zur automatischen Parallelisierung von Auswertungen auf großen Datenmengen
  - Entwicklung bei Google
  - populäre Open-Source-Implementierung: Hadoop (seit 2006)
- Nutzung v.a. zur Verarbeitung riesiger Mengen teilstrukturierter Daten in einem verteilten Dateisystem
  - *Konstruktion Suchmaschinenindex*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

MapReduce

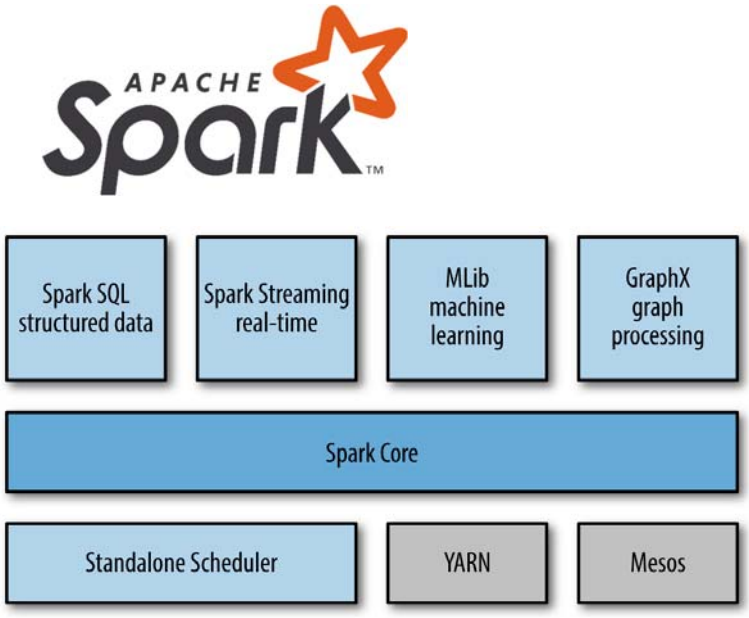
- Verwendung zweier Funktionen: **Map** und **Reduce**
- **Map**-Anwendung pro Eingabeobjekt zur Erzeugung von Key-value Paaren
  - jedes Key-Value-Paar wird einem Reduce-Task zugeordnet
- **Reduce**-Anwendung für jede Objektgruppe mit gleichem Key

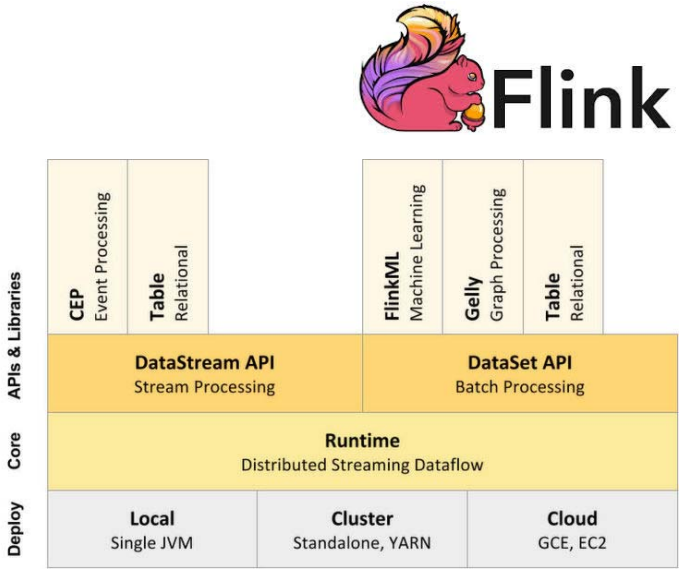


MR-Beispiel: Generierung Text-Index



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

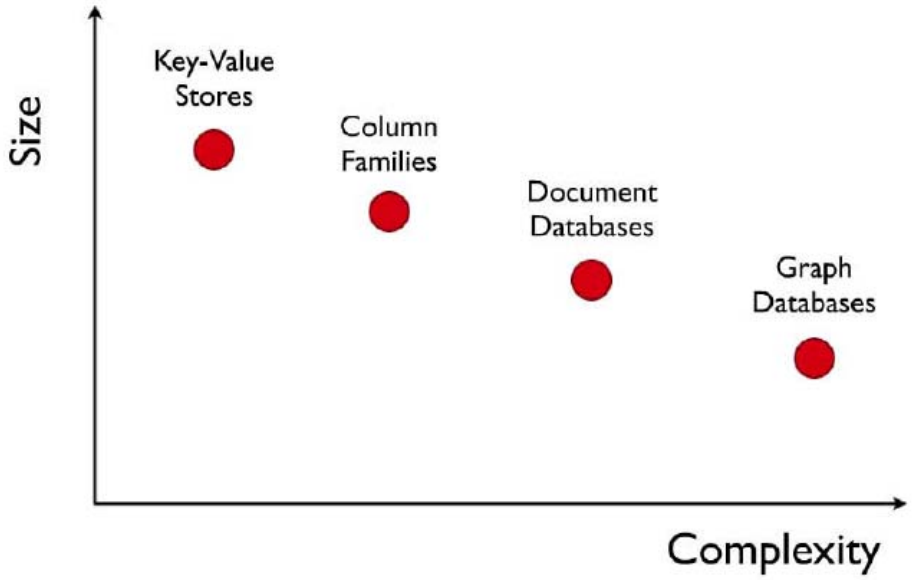
NoSQL-Datenbanken

- Entwicklung seit ca. 2009
  - ursprünglicher Fokus: moderne “web-scale” Datenbanken
- Merkmale
  - nicht-relational
  - verteilt,
  - open-source und
  - horizontal (auf große Datenmengen) skalierbar sind
- 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 “Not only Sql“ 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

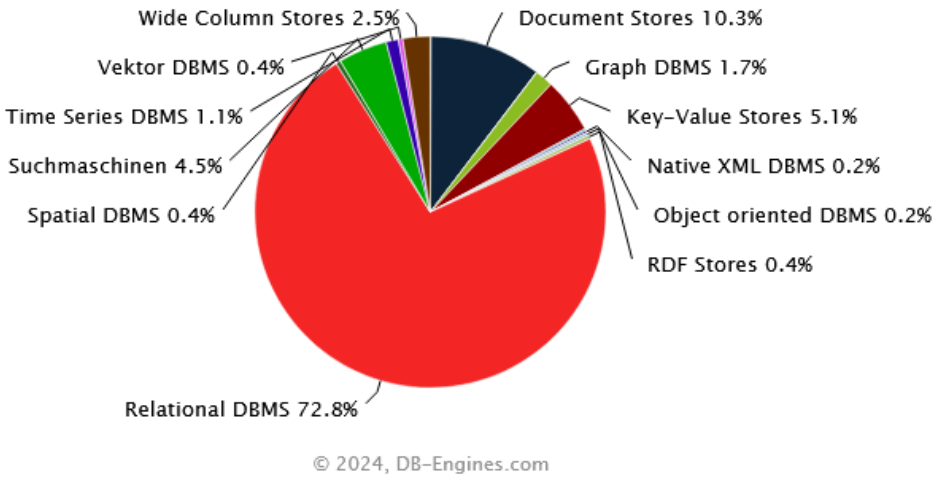


https://db-engines.com/de/ranking

421 Systeme im Ranking, Juni 2024

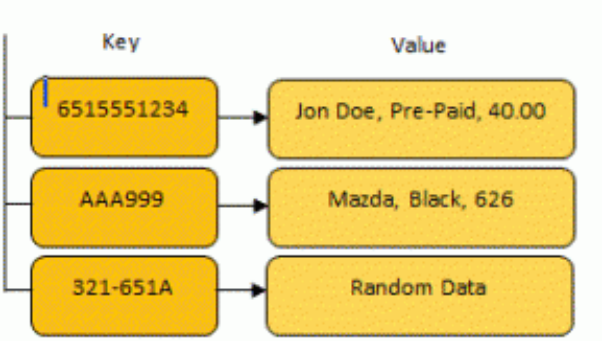
Rang Jun 2024	Rang Mai 2024	Rang Jun 2023	DBMS	Datenbankmodell	Punkte Jun 2024	Punkte Mai 2024	Punkte Jun 2023
1.	1.	1.	Oracle	Relational, Multi-Model	1244,08	+7,79	+12,61
2.	2.	2.	MySQL	Relational, Multi-Model	1061,34	-22,39	-102,59
3.	3.	3.	Microsoft SQL Server	Relational, Multi-Model	821,56	-2,73	-108,50
4.	4.	4.	PostgreSQL	Relational, Multi-Model	636,25	-9,30	+23,43
5.	5.	5.	MongoDB	Document, Multi-Model	421,08	-0,58	-4,29
6.	6.	6.	Redis	Key-value, Multi-Model	155,94	-1,86	-11,41
7.	7.	↑ 8.	Elasticsearch	Suchmaschine, Multi-Model	132,83	-2,52	-10,92
8.	↑ 9.	↑ 11.	Snowflake	Relational	130,36	+9,03	+16,23
9.	↓ 8.	↓ 7.	IBM Db2	Relational, Multi-Model	125,90	-2,56	-18,99
10.	10.	10.	SQLite	Relational	111,41	-2,91	-19,81
11.	11.	↓ 9.	Microsoft Access	Relational	101,16	-3,75	-33,29
12.	12.	12.	Cassandra	Wide column, Multi-Model	98,83	-3,06	-9,73
13.	13.	13.	MariaDB	Relational, Multi-Model	91,04	-2,17	-6,28
14.	14.	14.	Splunk	Suchmaschine	89,10	+2,65	-0,35
15.	15.	↑ 18.	Databricks	Multi-Model	81,08	+2,47	+15,27
16.	16.	16.	Microsoft Azure SQL Database	Relational, Multi-Model	76,78	-1,20	-2,18
17.	17.	↓ 15.	Amazon DynamoDB	Multi-Model	74,45	+0,38	-5,46
18.	18.	↓ 17.	Hive	Relational	59,76	-1,42	-15,76
19.	19.	↑ 20.	Google BigQuery	Relational	58,10	-2,28	+3,46
20.	20.	↑ 21.	FileMaker	Relational	47,91	-0,29	-6,47
21.	↑ 23.	↑ 22.	Neo4j	Graph	44,89	+0,44	-7,87
22.	↓ 21.	↓ 19.	Teradata	Relational, Multi-Model	44,87	-0,46	-17,77
23.	↓ 22.	23.	SAP HANA	Relational, Multi-Model	44,27	-0,42	-7,14

Rankingpunkte pro Kategorie in Prozent, Juni 2024



Key-Value Stores

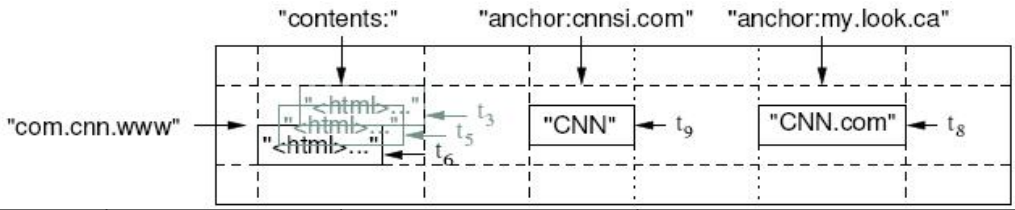
- Speicherung von Schlüssel-Werte-Paaren
  - im einfachsten Fall bleiben Werte systemseitig uninterpretiert (BLOBs)
  - flexibel, kein Schema
  - günstig für wenig strukturierte Inhalte bzw. stark variable Inhalte, z.B. Twitter-Nachrichten, Webseiten etc.
  - keine Verwaltung von Beziehungen



- schnelle Lese/Schreibzugriffe über Schlüssel
  - put (key, value)
  - get (key)
- keine komplexen Queries (z.B. Bereichsabfragen)

Erweiterbare Record Stores

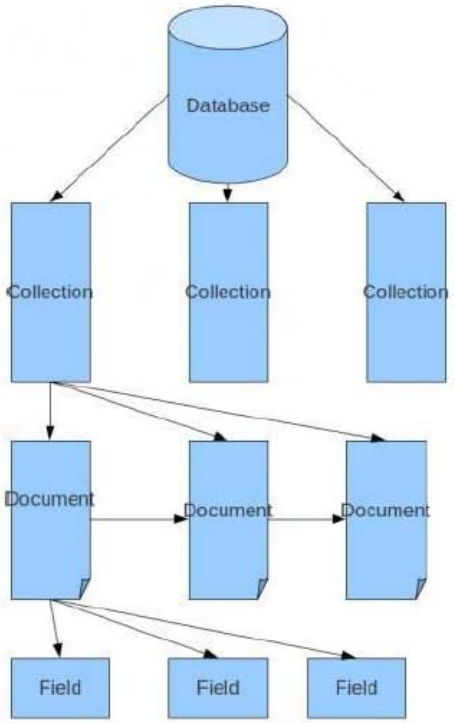
- spaltenorientierte Key-Value Stores mit Erweiterbarkeit über Spaltenfamilien
- Vorbild: Google BigTable
  - Clones: HBase, Cassandra, …
- Ziele (BigTable):
  - Milliarden von Zeilen, Millionen von Spalten, Versionierung (z.B. für Web-Seiten)
  - einfache Erweiterbarkeit um Spalten (innerhalb vordefinierter *Spaltenfamilien*)
  - mehrdimensionale Keys: Zeilen- und Spalten-Schlüssel, Versionsnr



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

MongoDB

- 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.save(student_a);
```

```
> db.students.find();
{_id:1, sno:731, name:'Irving Bonce', address:
        {street:'38 Glenpark Ave',
         city:'Dunedin',
         phone:4535467}}
```

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}});
```

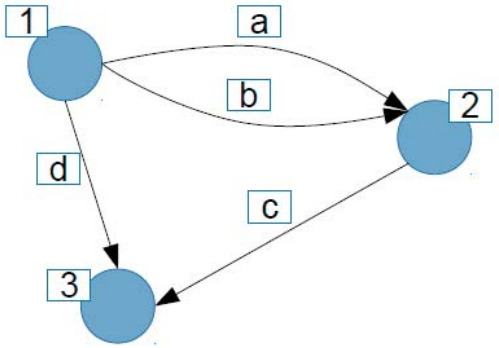
Graph-Datenbanken

- 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)
  -

Query-Sprache Cypher

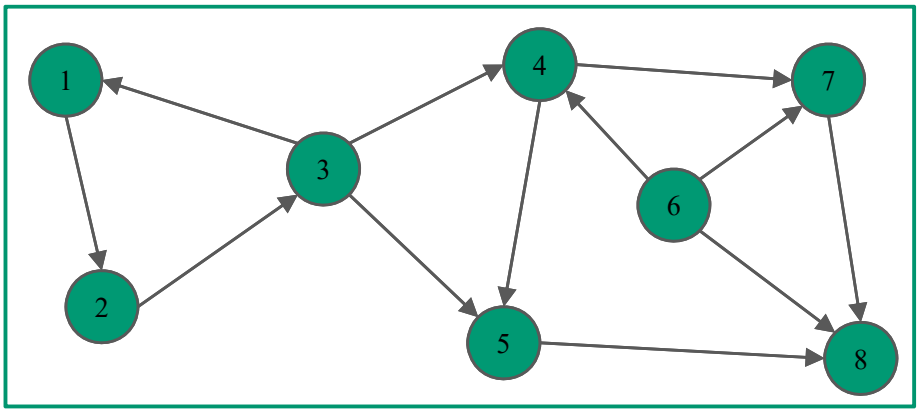
- Merkmale
  - Pattern Matching
  - OLTP-artige Lese/Änderungsoperationen
  - schnelle Traversierungen
  - ACID-basierte Updates
- Klauseln:
  - **START**: Starting points in the graph.
  - **MATCH**: The graph pattern to match.
  - **WHERE**: Filtering criteria.
  - **RETURN**: What to return.
  - **CREATE**: 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.

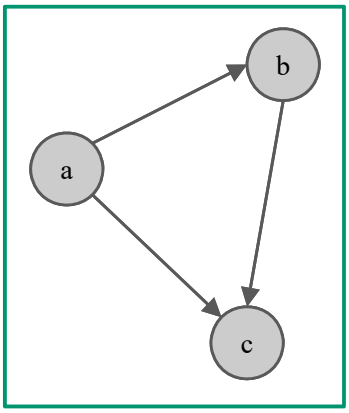
Cypher-Beispiel (1)

```
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

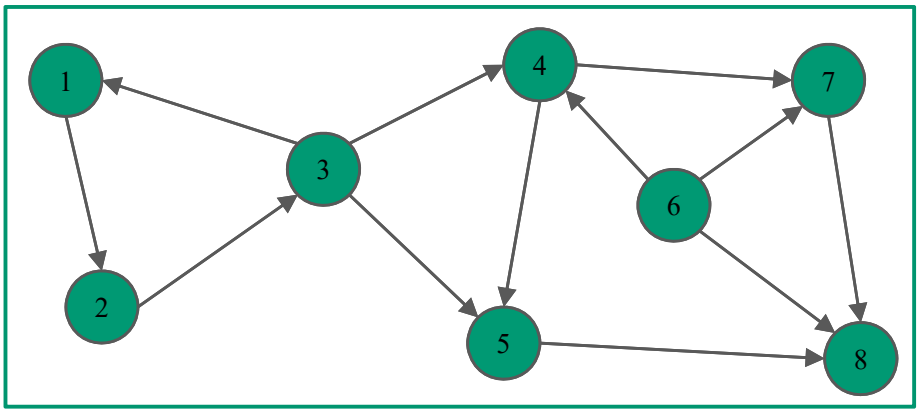


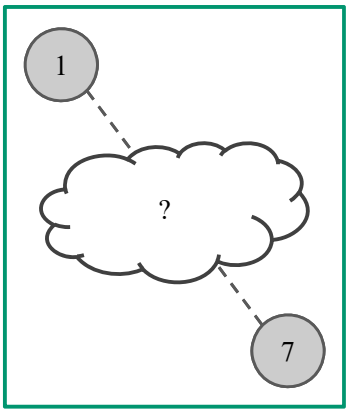


Cypher-Beispiel (2)

```
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

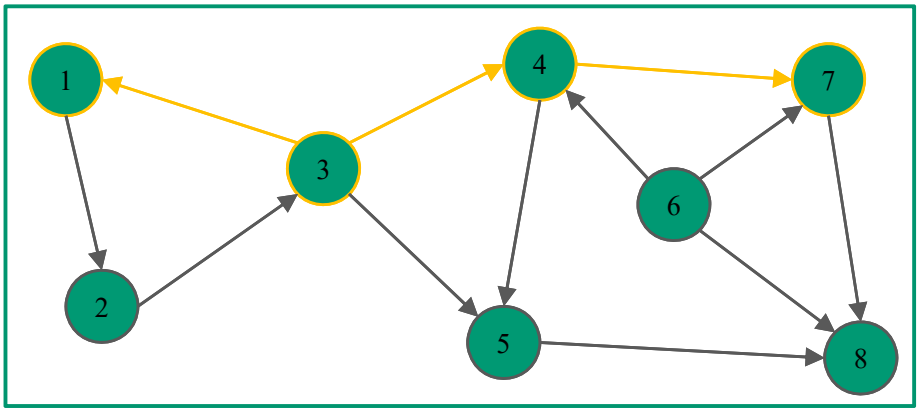


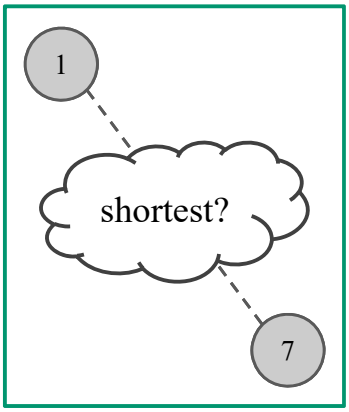


Cypher-Beispiel (3)

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

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



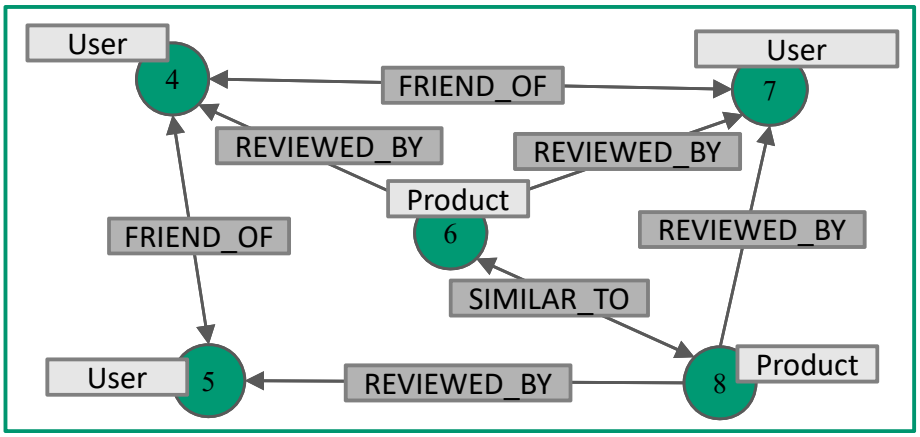


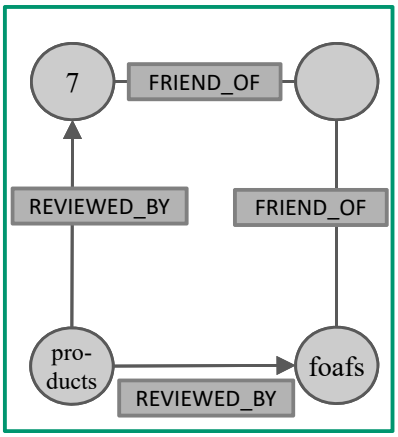
Cypher-Beispiel (4)

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: kaum Standards
  - Ausnahme: Graph Query Language (GQL) seit 2024