****

Einsatz und Möglichkeiten von NoSQL-Datenbanken im Hinblick auf große Datenmengen (Big-Data)

am Beispiel von Webanwendungen mit MongoDB

**Bachelorarbeit**

im Studiengang Medieninformatik

vorgelegt von

**Okotto Pepin Armand**Matrikel-Nr. : 7001520

am 29. Juli 2015  
an der Hochschule Emden-Leer

Erstprüfer: Dipl.-Inform. Andreas Wilkens  
Zweitprüfer: Dipl.-Ing. Jürgen Meyer

# Kurzfassung

**Schlagwörter**: Datenbanken, NoSQL, Map-Reduce, Datenmodelle, Abfragesprache,

Webanwendungen, Big-Data , Big-User, Cloud, Performance, ACID, BASE, CAP.

**Abstracts:**

**Keywords:** Database, NoSQL, Map-Reduce, MongoDB, CouchDB, Web Applkation, Big-Data

# Inhaltsverzeichnis

Kurzfassung 2

Inhaltsverzeichnis 3

Abbildungsverzeichnis 6

Tabellenverzeichnis 6

Abkürzungsverzeichnis 7

1 Einleitung 8

2 Grundlagen 10

2.1 Probleme traditioneller relationaler Datenbanken 10

2.2 Treibende Faktoren 12

2.2.1 Big-Data 12

2.2.2 Big-User 13

2.2.3 Cloud Computing 14

2.3 Definition von NoSQL-Datenbanken 15

2.4 Grundlegende technische Konzepte von NoSQL 16

2.4.1 Verteiltes System (CAP Theorem /BASE) 16

2.4.2 Skalierbarkeit und Leistungssteigerung 17

2.4.3 Sharding und Replikation 18

2.4.4 Integriertes Caching 18

2.4.5 Map/Reduce 18

2.5 Zusammenfassung 19

3 NoSQL-Datenbanken 20

3.1 Key-Value-Datenbanken (Schlüssel/Wert-Speicher) 20

3.1.1 Key-Value Datenmodell 20

3.1.2 Anwendungsfall Speichern von Webseiten 21

3.1.3 Vor- und Nachteile 22

3.1.4 Key Value Vertreter 22

3.1.5 Fazit 23

3.2 Graph-Datenbanken 23

3.2.1 Datenmodelle 25

3.2.2 Abfragesprache 26

3.2.3 Anwendungsfälle 26

3.3 Spaltenorientierte Datenbanken 27

3.3.1 Zeilenorientierter Ansatz (relational Datenbanken) 27

3.3.2 Spaltenorientierter Ansatz 28

3.3.3 Vor- und Nachteile 30

3.3.4 Einsatzbereiche 30

3.3.5 Fazit 30

3.4 Dokumentorientierte Datenbanken 31

3.4.1 Anwendungsfälle 31

3.4.2 Fazit 31

3.5 Zusammenfassung 32

4 Einsatzmöglichkeiten und Grenzen von NoSQL im Hinblick auf Big-Data 35

4.1 Auswahlkriterien für eine NoSQL - Datenbank 35

4.1.1 Kriterienkatalog 35

4.1.2 Performance 35

4.2 Transaktionsanwendungen 36

4.2.1 Anforderungen an Datenmodelle 36

4.2.2 Datenzugriffsanforderung 36

4.2.3 Fazit 36

4.3 Rechnungsanwendungen 36

4.3.1 Anforderungen an Datenmodelle 36

4.3.2 Datenzugriffsanforderung 36

4.3.3 Fazit 36

4.4 Webanwendungen 36

4.4.1 Anforderungen an Datenhaltung 37

4.4.2 Datenzugriffanforderung 37

4.4.3 Zusammenfassung 37

5 Webanwendung mit MongoDB 38

5.1 Wer nutz MongoDB 38

5.2 Überblick über MongoDB 39

5.2.1 Datenbanken 41

5.2.2 Installation und Treiber 41

5.2.3 Kollektion und Dokument 42

5.3 Schema-Design – Blog Modellierung 44

5.3.1 Analyse 44

5.3.2 Modelle 44

5.4 CRUD (create, reads, update, delete)- Operationen Blog-User Interface 44

5.5 Session Manager –Benutzer Profile 44

5.6 Fortgeschrittene Abfragen und Map/Reduce (Blog-Cloud-Tag, Bewertung pro Autor) 44

5.7 Web Analytik mit MongoDB 45

5.8 MongoDB Admin 45

5.8.1 Replikation 45

5.8.2 Ausfallsicherheit 45

5.9 Bewertung 45

6 Zusammenfassung und Ausblick 46

Anhang A1: Übersicht von NoSQL-Datenbanken 47

A.2 47

Anhang B: Quellcode 48

Glossar 49

Literaturverzeichnis 50

Eidesstattliche Versicherung 52

# Abbildungsverzeichnis

Abbildung 2‑1 Erfassung Polymorphe Daten im RDBS [] 10

Abbildung 2‑2 Globale Bevölkerung online Nutzers und Smartphone Besitzer Weltweit[Couc15] 13

Abbildung 2‑3 CAP Eigenschaften [Schi14] 17

Abbildung 2‑4 MapReduce Prinzip [Holl12] 19

Abbildung 3‑1 Beispiel Wörterbuch. Schlüssel: Werte [McKe14] 20

Abbildung 3‑2 Key-Value Datenmodell [McKe14] 21

Abbildung 3‑4 Linked-Data-(RDF) – Repräsentation einer Kontaktinformation [NeHu15] 25

Abbildung 3‑5 Property-Graphen mit Knoten und Kanten [Geye15] 26

Abbildung 3‑6 Beispiel Datei in spaltenorientierten Datenbanken 29

Abbildung 3‑7 Entscheidungspyramide nach Hurst [Hurs10] 33

Abbildung 0‑1 Überblick über verschiedenen NoSQL-Datenbanken 47

# Tabellenverzeichnis

Tabelle 3‑1 URL als Key und Seite als Value [McKe14] 23

Tabelle 3‑2 Key-Value Überblick [Edli12] 24

Tabelle 3‑3 Relationale Modelle im Vergleich zu Graph-Modellen 25

Tabelle 3‑4 Überblick Graph-Datenbanken [Edli12] 28

Tabelle 3‑5 Zeilenorientierter Ansatz Objekt „Person“ 29

Tabelle 3‑6 Objekt Person im Speicher (relationale Datenbanken) 29

Tabelle 3‑7 Objekt Person im Speicher (spaltenorientierte Datenbanken) 30

Tabelle 3‑8 Überblick Spaltenorientierte Datenbanken [Edli12] 31

Tabelle 3‑9 Überblick Dokumentorientierte Datenbanken [Edli12] 33

Tabelle 3‑10 Überblick über populärsten NoSQL –Datenbanken und Einsatz[Gull11] 35

Tabelle 5‑1 MongoDB Struktur 42

Tabelle 5‑2 Aufbau einer ObjekID[Mong15] 45

# Abkürzungsverzeichnis

NoSQL Not only Structured Query Language

SQL Structured Query Language

URL Uniform Resource Locator

WWW World Wide Web

# Einleitung

*„Wer als Werkzeug nur einen Hammer hat, sieht in jedem Problem einen Nagel*.“*[Watz15]*

Daten sind heute sehr wichtige Produktionsfaktoren. Sie werden erzeugt, gesammelt, manipuliert, analysiert und archiviert. In der IT werden ständig Lösungen entwickelt, um diese Daten zu persistieren. Ohne Datenbankensysteme wären große Teile der Wirtschaft und des Handels sowie Forschung und Entwicklung nicht realisierbar. Vor 40 Jahren gab es keine großen Anforderungen an Datenbanken, die meisten Datenbanksysteme befanden sich auf Großrechnern oder Minicomputern. Die Systeme waren zentral und besaßen eine Ebene, was bedeutet, dass sich das Datenbankmanagementsystem selbst und die Daten am selben Ort befanden [MaCu03].

In Zeiten von Web 2.0 haben die wachsende Anzahl und Verwendung digitaler Endgeräte (Mobiltelefon, Tablets), die Nutzung von Sozialen Netzwerken und Diensten (Facebook, Twitter) sowie Sensordaten und e-Business-Anwendungen zu einer exponentiellen Datengenerierung geführt. In Sozialen Netzwerken kursieren Videos, Pinnwandeinträge etc. auf verschiedenen Geräten mit verschiedenen Betriebssystemen. Diese Aktivitäten im Netz erzeugen große Datenmengen (Big-Data). Eine IDC Studie zeigt, dass die weltweit gespeicherte, genutzte, erstellte oder kopierte Datenmenge im Jahr 2012 1,8 Zettabyte (1,8 Billionen Gigabyte) beträgt [GaRe11]. Die Social Media Plattform Facebook hat beispielweise nach eigenen Angaben im März 2014 über 802 Millionen aktive Benutzer gehabt. 609 Millionen Benutzer melden sich täglich mit ihren mobilen Geräten an, welche vier Terabyte an Daten pro Tag produzieren [Facebook15]. Bei Twitter zeigen die Statistiken, dass z.B. im Monat Mai 2014 mehr als 230 Millionen aktive Nutzer getwittert haben. Täglich werden 500 Millionen Tweets versendet [Twitter15]. Amazon ist sehr erfolgreich im Bereich des Online-Verkaufs. Das Portal erhält viele Anfragen und muss ebenso viele Buchungen pro Sekunde bewältigen.

Getrieben durch diese allgemeine Digitalisierung stehen immer mehr Unternehmen vor der Herausforderung, diese großen und zum Teil unstrukturierten Datenmengen zu verwalten und zu analysieren, um sie für ihre Geschäftsprozesse optimal zu verwenden. Jahrelang waren relationale Datenbanken der facto-Standard für alle Datenhaltungsprozesse. Jedoch haben Soziale Netzwerke wie Facebook, Twitter und Co. mit vielen Millionen, über den ganzen Globus verteilten Benutzern andere Anforderungen an Datenhaltung als zum Beispiel die Verwaltung einer Bank oder eine Versicherung. Bei Social Media müssen sehr viele Daten aus unterschiedlichen Quellen sehr schnell abgespeichert und abgerufen werden können. Dabei stoßen relationale Datenbanken an ihre Grenzen; im Bezug auf ihre Technologie sowie ihr Lizenzmodell.

Seit einigen Jahren heißt das neue Schlagwort NoSQL, übersetzt „nicht nur SQL“. NoSQL bietet eine Antwort auf diese neuen Herausforderungen; ein Ansatz der anstelle starrer Tabellenstrukturen eine flexible Anordnung von Daten vorsieht. Google mit dem spaltenorientierten Datenbank-Management-System Big Table und Facebook mit Casandra waren die Vorreiter dieser neuen Technologie. Jedoch gibt es mehr als 150 Datenbanken, die unter dem Schirm NoSQL stehen. Fast jede Woche wird etwas neues in dem Bereich freigegeben.

Im Hinblick auf diese Fülle an Datenbanken stehen IT Entscheider zunehmend vor der Herausforderung, die richtigen Datenbanken zu wählen, die der jeweiligen Anforderung gerecht werden sollen.

Ziel dieser Arbeit ist es, die NoSQL Technologien unter die Lupe zu nehmen, dabei ihre Fähigkeiten zu untersuchen und ihre optimale Nutzung für Big-Data darzustellen.

Diese Arbeit ist in fünf Teile gegliedert.

Teil1. Grundlagen: Hier geht es darum zu zeigen, dass NoSQL-Datenbanken die existierenden SQL Datenbanken zunehmend verdrängen, besonders in dem Bereich, wo man mit großen und komplexen Datenmengen zu tun hat und wo von verteilten Datenbankensystemen die Rede ist. Dabei werden die Grundlagen von NoSQL erläutert.

Teil2. Gibt einen Ausblick zu existierenden NoSQL Datenbanken. Zur Zeit gibt es mindestens 150 Datenbanken, die den Anspruch erheben, NoSQL Technologien zu sein. Nach ihren Funktionalitäten sind diese Datenbanken in vier Gruppen geteilt.  Jede dieser Gruppen ist anders entwickelt worden und besitzt verschiedentliche unterschiedliche Eigenschaften und Potentiale.

Teil3. Teil 3 beschäftigt sich mit dem Einsatz und Möglichkeiten von NoSQL-Datenbanken. Dabei sollte man definieren, welche Anwendungen bzw. Geschäftsmodelle NoSQL benötigen.

Anwendungen wie z.B. Webanwendungen, Transaktionsanwendungen und Rechnungsanwendungen haben ihre eigenen Anforderungen an Datenmodell sowie an Zugriffsart als auch an Konsistenzmodell.

Teil 4. Dieser Teil dient der praktischen Anwendung.

MongoDB, ein Dokument-Datenmodell ist die populärste NoSQL-Datenbank und ist sehr beliebt bei Webanwendungen. MongoDB ist einfach und leicht zu erlernen und bietet viele Features, die auch in SQL zu finden sind. BLOG oder CMS sind CRUD freundlich und können komplexen Dateien enthalten.

# Grundlagen

## Probleme traditioneller relationaler Datenbanken

Obwohl SQL weiter entwickelt wurde und über mächtige Features verfügt, kann das System nicht mit den neuen Anforderungen standhalten. Anhand der folgenden zwei Szenarien sollen die Probleme traditioneller relationaler Datenbanken verdeutlicht werden.

* **Szenario 1: Eine Unterhaltungs-App**

Ein Entwickler hat für ein Unternehmen eine Unterhaltungs-App entwickelt wie auf der Abbildung 2-1 dargestellt ist. Die Daten persistieren auf einer relationalen Datenbank. Diese App wird sehr bekannt und beliebt. Die Anzahl der Nutzer und die daraus hervorgegangenen Daten wachsen exponentiell, sodass die Kapazität des Datenbankspeichers schnell erschöpft ist. Der Entwickler und sein Team entscheiden sich, das Datenbanksystem zu skalieren[siehe 1.3.2], um die Verfügbarkeit zu gewährleisten.

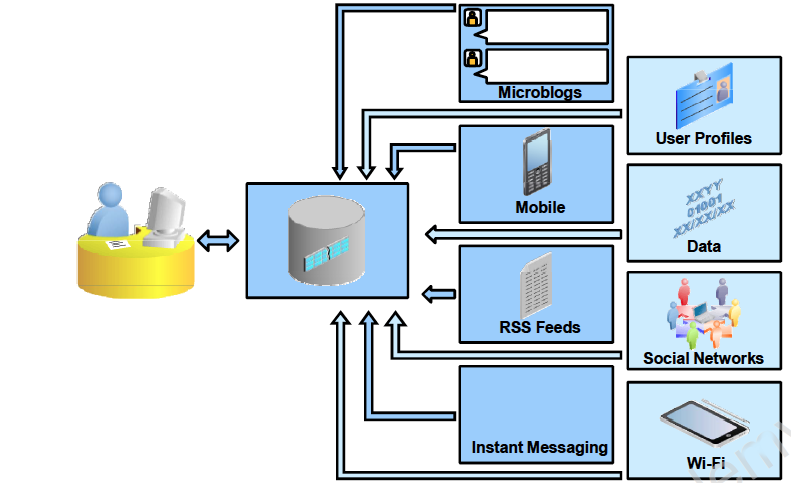


Abbildung ‑ Erfassung Polymorphe Daten im RDBS []

Das Team bekommt die zusätzliche Aufgabe, auch User-Profile unter bestimmten Bedingungen zu erfassen und diese so erfassten Daten in die App zu integrieren. Dies erfordert, dass **die** **Datenmodelle neu modelliert werden müssen**.

Um die Marketing-Abteilung besser zu unterstützen, schlägt der Vorstand dem Entwickler und seinem Team vor, zusätzliche “Social Activities“ über das Netz über jeden Benutzer (Mobil, Facebook , Twitter, YouTube etc.) aufzunehmen. Somit wächst mit der Beliebtheit dieser App die Herausforderung für die Entwickler. Die Daten müssen weiter gespeichert werden und mit der Zeit nehmen sie ein enormes Ausmaß an. Wie soll man das System skalieren, um eine bessere Antwortzeit zu gewährleisten? Wie soll man diese polymorphen Daten persistieren

* **Szenario 2: Webserver-Logdateien**

Ein Datenbank-Entwickler hat die Aufgabe, Daten von Webseiten zu erfassen, um den Aufbau und die Struktur einer Internetseite zu optimieren (Webcontrolling). Dafür werden wichtige Daten wie User-ID, URL, Time-Stamp und Additional-Infos erhoben und gespeichert. Wenn man diese Daten genau betrachtet, ist es leicht zu erkennen, dass sie nicht die gleiche Struktur haben. Zum Beispiel können Additional-Infos verschiedene Video- und Foto-Formate beinhalten, eine URL kann ungültig sein oder der Time-Stamp kann in einem unbekannten Format abgegeben werden.

In beiden Szenarien müssen sich die Entwickler Gedanken machen über die unstrukturierten Daten und die häufigen Änderungen. Relationale Modelle sind weniger flexibel für Entwickler [Kuba10], weil

* die Datenbank eine Zeit lang nicht nutzbar ist, während das Schema verändert wird.
* die Verteilung der Daten nicht möglich oder insofern teuer ist, als es in der Wartung und/oder Beschaffung zusätzlicher Lizenzen erheblich mehr als Alternative kostet.
* mit SQL eine zusätzliche Sprache eingeführt wird,
* SQL im Allgemeinen unelegant und komplex ist,
* Transaktionen bzw. Zeilen/Tabellenweise Sperren (“Locking“) nicht immer nötig sind, die Verarbeitungsgeschwindigkeit aber mindern.
* traditionelle relationale Datenbanken sehr oft vertikal skalieren (Scale up) und nur gut für Anwendungen funktionieren, bei denen das Nutzeraufkommen ein gewisses Maß an Elastizität nicht übersteigt.
* das ACID Konsistenzmodell eingehalten werden muss. Diese Eigenschaft lässt sich nur schlecht mit einer Anforderung nach horizontaler Skalierung vereinbaren, da ein großer Kommunikationsaufwand betrieben werden muss, um Konsistenz über mehrere Knoten zu gewährleisten.
* Ein Weitere Problem ist der Flexibilität: Schema Änderung bei relationalen Datenbanken sind insbesondere dann schwierig, wenn bereits große Datenmengen vorhanden sind

Die Geschichte von SQL Datenbanken ist zurückzuführen auf die Entwicklung von funktionellen Programmiersprachen wie Cobol[]. In der früheren Entwicklung von relationalen Datenbanken waren objektorientierte Sprachen, Internet, Big User, verteilte Systeme oder Big-Data unbekannte Begriffe. Damals gab es Installationen, die höchstens 1.000 Benutzer bedient haben und ein Server war ausreichend für die Datenhaltung. Heute laufen die meisten Geschäftsmodelle über das Web und sind täglich durch Millionen Benutzer belastet. Drei Trends sind Treiber für eine Massive Revolution des Status Quo in der Welt von Datenbanken: **Big-Data, Big-User und Cloud Computing.**

## Treibende Faktoren

### Big-Data

Unter dem Begriff Big-Data fassen Experten zwei Aspekte zusammen: zum einen immer schneller wachsende Datenberge, zum anderen IT-Lösungen und Systeme wie Hadoop, die Unternehmen dabei helfen, mit der Informationsflut fertig zu werden. Vor allem unstrukturierte Daten, beispielsweise aus sozialen Netzwerken, verursachen den Großteil von Big-Data [Lixe15] .

Durch Social Media wie Facebook, Twitter und viele andere ist es einfach geworden, Zugang zu Daten zu erhalten. Geo-Positionsdaten, Soziale Graphen, Persönliche Benutzerinformationen, nutzergenerierte Inhalte, Maschinendatenerfassung, sowie Sensoren generierte Daten sind nur einige Beispiele für die ständig wachsende Menge an Daten, die erfasst wird. Es überrascht daher nicht, dass Entwickler aus diesem Grund bestehende Anwendungen verbessern und neue erstellen möchten. Die Nutzung dieser Daten verändert vieles von Grund auf und in einem hohen Tempo: Kommunikation, Einkaufen, Werbung, Unterhaltung und die Kundenbeziehung. Apps, die diesem Trend nicht Rechnung tragen, werden schnell ins Hintertreffen geraten.

Unternehmen und Entwickler benötigen sehr flexible Datenbanken, die schnell neue Arten von Daten aufnehmen und die nicht durch Struktur und Content-Änderungen von Drittdatenanbietern beeinträchtigt werden können. Sie brauchen also einen neuen Ansatz für den Umgang mit dem hohen Volumen, der Geschwindigkeit und der Vielfalt von Big Data.

**Volumen:** Bis zum Jahr 2020 wird das digitale Universum auf 40.000 Exabyte oder 40 Billionen Gigabyte anwachsen (das sind über 5.200 Gigabyte pro Erdbewohner). Bereits jetzt müssen Daten im Petabytebereich schnell und mit möglichst wenig Aufwand verarbeitet werden[IDC14].

**Geschwindigkeit**: 90 Prozent aller Daten dieser Welt wurden in den vergangenen zwei Jahren erstellt. Der Datenstrom muss in Echtzeit verwaltet und analysiert werden, weil Unternehmen und Behörden es sich nicht leisten können, diese Aufgabe aufzuschieben. [IDC14].

**Vielfalt:** Die Daten sind außerordentlich vielfältig, und 80 Prozent aller Daten sind unstrukturiert. In einer Studie gaben 64 Prozent aller befragten Unternehmen an, dass der Hauptgrund für die Erwägung eines neuen Big Data-Ansatzes die Bewältigung der vielen unterschiedlichen neuen Echtzeitdatenquellen sei [IDC14].

### Big-User

Vor einigen Jahren waren 900 User für eine Anwendungen normal und mit 10.000 Usern waren Systeme überfördert. Heute ist es üblich, Anwendungen mit Millionen Usern am Tag zu sehen. Allein werden bei Twitter pro Tag 500 Millionen Tweets versendet [Twit15]. Das liegt vor allem an der Globalisierung vom Internet und der weiteren Entwicklung anderer Technologien. Nutzer verbringen mehr Stunden online und mobile Geräte werden populär.

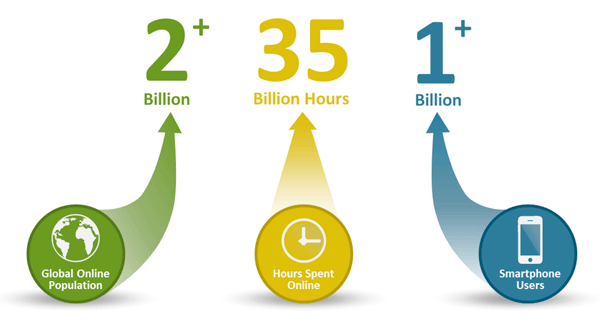


Abbildung ‑ Globale Bevölkerung online Nutzers und Smartphone Besitzer Weltweit[Couc15]

Es ist wichtig, dass eine große Anzahl an gleichzeitigen Usern unterstützt wird. Gerade aber weil die Anforderungen an die Nutzung der Apps so unvorhersehbar sind, ist es genauso wichtig, eine schnell wachsende (oder sinkende) Zahl an gleichzeitigen Usern dynamisch unterstützen zu können.

* Eine neu gestartete App kann viral werden und die Anzahl der User über Nacht buchstäblich von Null auf eine Million ansteigen.
* Manche User sind häufig aktiv, andere nutzen die App nur ein paar Mal und dann nie wieder.
* Saisonale Schwankungen, beispielsweise um Weihnachten oder den Valentinstag, können für kurze Zeit zu einer Spitzenauslastung führen.

Genau diese hohe Anzahl an Usern und das dynamische Nutzerverhalten sind der Grund dafür, dass der Bedarf an leichter skalierbaren Datenbanktechnologien steigt. Für viele Anwendungsentwickler ist es schwierig oder beinahe unmöglich, mit relationalen Technologien die nötige dynamische Skalierbarkeit und Größe zu erreichen und dabei gleichzeitig die benötigte konstant verfügbare Leistung aufrecht zu erhalten, so wie es das Nutzervolumen erfordert.

### Cloud Computing

Heutzutage basieren die meisten neuen Anwendungen (sowohl für Privatanwender als auch Unternehmen) auf einer 3-Schichten-Internetarchitektur. Sie laufen auf einer öffentlichen oder privaten Cloud und unterstützen eine große Anzahl an Usern.

In dieser Drei-Schichten-Architektur gelangt man über einen Webbrowser oder eine mobile App, die mit dem Internet verbunden ist, auf die Anwendungen. In der Cloud leitet ein Load-Balancer den eingehenden Datenverkehr zu einer vertikal skalierten Schicht von Web/Anwendungsservern, die die Logik der Anwendung verarbeiten. Auf der Web/Anwendungsschicht funktioniert die Scale-Out Architektur perfekt. Pro 10.000 (oder eine beliebige andere Anzahl) neue gleichzeitig aktive User wird einfach ein weiterer Standardserver zu der Webanwendung hinzugefügt, um die Last aufzufangen.

Auf der Datenbankebene waren relationale Datenbanken zunächst eine beliebte Wahl. Ihr Einsatz wurde jedoch zunehmend problematischer, da sie auf einer zentralisierten, Share-Everything Technologie basieren, die eher Möglichkeiten zum vertikalen Skalieren als zum horizontalen Skalieren bieten. Daher waren sie keine optimale Lösung für Anwendungen, die einfach und dynamisch skalierbar sein müssen. Deswegen müssen sich Entwickler die beste Alternative aussuchen, die die drei Säulen der Verteilten Systeme vereinbaren: Leistung, Skalierbarkeit und Konsistenz.

## Definition von NoSQL-Datenbanken

Eine feste Definition von NoSQL gibt es nicht, stattdessen wird der Begriff NoSQL verstanden als “Not only SQL“ oder “Not Relational“ und beschreibt dabei statt einer konkreten Technologie, verschiedene Ansätze im Bereich Persistenz. Alle diese Technologien besitzen als kleinsten gemeinsamen Nenner die Eigenschaft, nicht auf relationalen Datenbankenmodellen aufzubauen. NoSQL Datenbanken zeichnen sich durch folgende Merkmale aus [EdHa11]

* **Kein relationales Datenmodell.** Somit kann die Speicherung der Daten in NoSQL-Datenbanken flexibler gestaltet oder komplett der Anwendung überlassen werden.
* **Open Source**
* **Eignung für Systeme mit verteilter und horizontaler Skalierbarkeit**. Viele NoSQL bieten die Möglichkeit, verteilte Datenbanken zu verwenden und die Daten auf mehrere Server aufzuteilen.
* **(Fast) schemafrei oder schwächere Schemarestriktion.** Web 2.0 muss agile sein. Die Daten haben nicht mehr eine gleiche Struktur. Die Datenbanken müssen damit umgehen können. Darüber hinaus muss auch eine schnelle Reaktionszeit auf Schemaänderungen gewährleistet sein.
* **Einfache Programmierschnittstelle (API – Applikation Programming Interface).** SQL, die Standard Abfragesprache für relationale Datenbanken, ist sehr komplex, viele Joins, Unterabfragen ineinander geschachtelte Selects, rekursive Selects, machen die Abfragen teilweise schwer überschaubar und überfordern manche Programmierer. NoSQL Datenbanken bieten einfache Schnittstellen, die aber weniger mächtig sind. Die Mehrheit solcher Datenbanken machen komplexe Abfragen auf Map/Reduce Techniken.
* **Kein ACID als Konsistenz Modell.** ACID ist unverzichtbar für Anwendungen mit Auftrags- oder Finanzdaten, aber für Anwendungen mit unkritischen Daten beispielsweise im Social-Web reicht BASE aus. Hier liegt der Schwerpunkt auf der Verfügbarkeit und weniger auf der Konsistenz der Daten.
* **Aufgrund der verteilten Architektur** unterstützt das System eine einfache Datenreplikation. –einzelne Shards werden ständig gespiegelt.

Eine der populärsten Web-Seiten über NoSQL ist [www.nosql.org](http://www.nosql.org), gepflegt von Prof.Dr. Stefan Edlich. Hier findet man eine Fülle an Informationen über die unterschiedlichen Arten von NoSQL Technologie und die Seite ist stets aktualisiert. Der Autor listet mehr als 150 verschiedene Arten von Datenbanken, die unter dem Schlagwort NoSQL zusammengefasst werden können. Entsprechend ihrer Grundfunktionalität können die meisten NoSQL Systeme den vier folgenden Hauptgruppen zugeordnet werden.

* **Dokumentendatenbanken** koppeln jeden Schlüssel mit einer komplexen Datenstruktur, bekannt als Dokument. Dokumente können viele verschiedene Schlüssel-Wert-Paare oder Schlüsselfeldpaare oder sogar verschachtelte Dokumente enthalten.
* **Key-Values Store (Schlüssel/Wert)** kurz KV sind die einfachsten NoSQL-Datenbanken. Wie der Name andeutet, verknüpft ein KV-Speicher Schlüssel und Werte, wie eine MAP oder eine Hashtabelle.
* **Graphenbasierte Datenbanken** bestehen aus Knoten und Beziehungen zwischen diesen Knoten.
* **Spaltenorientierte Datenbanken** sind Datenbanken, in denen die Daten spaltenorientiert gespeichert werden.

Zum ausprobieren der verschiedene Dienste bietet Amazon ein recht großzugig bemessenes kostenfreies Nutzungskontinent für ein ganzes Jahr [Amaz].

## Grundlegende technische Konzepte von NoSQL

Um zu verstehen, wie NoSQL Datenbanken funktionieren und eingesetzt werden können, ist es wichtig, sich mit den Grundbegriffen und theoretischen Grundlagen auseinander zu setzen. Hier werden wesentliche Konzepte wie Map/Reduce, CAP-Theorem, Konsistenzmodell BASE, Replikation sowie Skalierbarkeit vorgestellt.

### Verteiltes System (CAP Theorem /BASE)

Big-Data Eigenschaften Volume und velocity sind noch möglich mit dem Einsatz von verteilten Systemen, in denen Last und die Daten auf viele einzelne Rechnerknoten verteilt werden. Jedoch besagt das CAP Theorem [Brew04], dass bei der Implementierung eines verteilten Datenbankensystems Konsistenz (alle Knoten haben jederzeit den gleichen Datenbestand), Verfügbarkeit (das System steht für Lese- und Schreibzugriffe zur Verfügung) und Toleranz (Daten sind immer vorhanden im Fall, dass ein Knoten oder das Netz ausfallen sollten) nicht gleichzeitig in vollem Maße erreicht werden kann. Eric Brewer vertritt die These, dass bei Skalierung von Datenbanken nur zwei der drei erreicht werden können Abbildung 1.3. Bei vielen NoSQL Technologien kann man aufgrund der hohen Verteilung nicht auf die Partition Tolerance (P) verzichten. Da die Verfügbarkeit (A) ebenfalls einen sehr hohen Stellwert einnimmt, bleibt nur der Ausweg, an der Konsistenz (C) zu sparen.

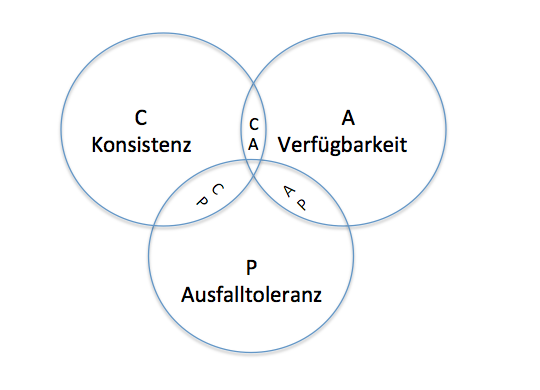


Abbildung ‑ CAP Eigenschaften [Schi14]

Eine Hauptanforderung von horizontal skalierten Datenbankensystemen ist, dass ein System aus mehreren Datenbanken immer erreichbar sein sollte. Viele NoSQL Datenbanken entsprechend PA Systeme.

BASE (Basicaly Available, Soft State, Eventually Consistent ) arbeitet ähnlich wie CAP. Mit BASE sind die verbundene Systeme verfügbare, aber können einen weichen beziehungsweise inkonsistenten Zustand – am Ende jedoch werden alle Knoten in einen Konsistenten Zustand gelangen. Bei manchen Anwendungen spielt die Konsistenz keine rolle oder ist Zweitrangig. z.B. Es ist nicht relevant, ob User z.B. gleichzeitig Tweets bekommen oder ob es eine Verzögerung an Informationen auf ihrem Facebook Account gibt. Wichtig ist, dass irgendwann die Informationen an den Empfanger gelangen.

### Skalierbarkeit und Leistungssteigerung

Unter Skalierbarkeit versteht man die Eigenschaft eines Software- oder Datenbanksystem mit der steigenden Anforderungen (z.B. durch eine größere Anzahl der Nutzer, größere Datenmengen) linear zu wachsen. Datenbanken können entweder vertikal (Scale-Out) oder horizontal (Scale-Up) skaliert werden. Beim vertikalen Skalieren wird zentralisiert aufgerüstet und in immer größere und leistungsfähigere Server investiert. Beim horizontalen Skalieren wird eine verteilte Strategie verfolgt, bei der viele Standardserver, physische oder virtuelle Server verbunden werden.

NoSQL-Datenbanken wurden von Grund auf als verteilte, auf Scale-Out-Technologie basierende Datenbanken entwickelt. Mehrere Standard-, physische oder virtuelle Server werden als Cluster zusammengeschlossen, um Daten zu speichern und Datenbankvorgänge zu unterstützen. NoSQL-Datenbanken bieten eine viel einfachere, lineare Lösung zur Datenbank-Skalierung. Wenn auf einmal 10.000 neue User ihre Anwendung nutzen, kann einfach ein weiterer Datenbankserver an das Cluster angeschlossen werden. Kommen weitere 10.000 User, fügen Sie den nächsten Server hinzu. Die Anwendung muss bei einer Skalierung nicht verändert werden, da die Anwendung immer nur eine einzige (verteilte) Datenbank wahrnimmt.

### Sharding und Replikation

NoSQL verteilt die Daten automatisch auf verschiedene Server, ohne dass dabei Anwendungen einbezogen werden. Server können der Datenebene hinzugefügt oder von ihr entfernt werden, ohne dass es zu Ausfallzeiten der Anwendung kommt. Die Daten (I/O) werden automatisch auf den Server verteilt. Die meisten NoSQL-Datenbanken unterstützen auch eine Replikation der Daten. Mehrere Kopien der Daten können innerhalb des Clusters, oder sogar über mehrere Rechen-Zentren gespeichert werden. Somit ist eine höhere Verfügbarkeit und Unterstützung für eine Disaster Recovery gewährleistet. Bei einem korrekt verwalteten NoSQL-Datenbanksystem ist es aus keinem Grund und zu keinem Zeitpunkt nötig, das System offline zu stellen, da der Betrieb aller Anwendungen konstant und rund-um-die-Uhr gewährleistet ist.

### Integriertes Caching

Um die Latenz niedrig zu halten und einen hohen Datendurchsatz zu gewährleisten, bieten hoch entwickelte NoSQL-Datenbanktechnologien ein transparentes Caching von Daten im Systemspeicher. Dies ermöglicht eine vollständige Transparenz für die Anwendungsentwickler und das Operations-Team, während bei relationalen Datenbanktechnologien die Caching-Ebene meist über eine eigene Infrastrukturebene verfügt, die auf separaten Servern entwickelt, eingesetzt und explizit von den Ops-Teams verwaltet wird.

### Map/Reduce

Dieses Framework wurde von Google Inc. im Jahr 2004 eingeführt, mit dem Ziel nebenläufige Berechnungen für enorm große Datenmengen auf Computerclustern zu gewährleisten. Das Prinzip hat seine Wurzeln in der funktionalen Programmierung. Beim Map/Reduce-Verfahren werden die Daten in zwei Phasen verarbeitet Dadurch lassen sich Berechnungen parallelisieren und auf mehrere Rechner verteilen Abbildung 1-4. Die erste Phase (Map) entspricht im wesentlichen der Auswahl von Datensätzen anhand bestimmter Kriterien, die Zweite (Reduce) führt Berechnungen, etwa Aggregation, auf den so erhaltenen Daten aus [Kirs15]. Bei sehr großen Datenmengen ist die Parallelisierung unter Umständen bereits schon deshalb erforderlich, weil die Datenmengen für einen einzelnen Prozess (und das ausführende Rechnersystem) zu groß sind



Abbildung ‑ MapReduce Prinzip [Holl12]

## Zusammenfassung

NoSQL Datenbanken können ihre Stärken am besten im Zusammenhang mit komplexen Daten, Stichwort Big-Data, und Cloud Umgebung zur Geltung bringen. Diese Technologien geben Lösungen auf die Herausforderungen der vier Facetten von Big-Data: Volume (Datenmenge), Variety (Quelle- und Formatvielfalt), Velocity (schnelle Datengenerierung) sowie Analytics (Analyse). Der Unterschied zwischen SQL und NoSQL liegt in der Performanz und in den Features. Viele NoSQL werden auf Funktionen wie ACID und SQL verzichten, um damit mehr Performanz und bessere horizontale Skalierbarkeit zu erreichen. Während SQL Speicher-orientiert sind, beziehen sich NoSQL primär auf Anwendungen.

# NoSQL-Datenbanken

Alle SQL Datenbanken (Oracle-SQL, Access, MySQL u.a.) haben einen gemeinsamen Nenner, die Abfragesprache SQL. NoSQL-Datenbankensysteme haben dagegen sehr viele Abweichungen in ihren Funktionalitäten und Anwendungen. Eine der großen Herausforderungen der NoSQL-Datenbanken ist die Schwierigkeit, das richtige Entwurfsmuster für das jeweilige Problem zu finden. In diesem Kapitel werden die vier wichtigsten Kategorien von NoSQL-Datenbanken: Key-Value, Graph, dokumentorientierte, sowie spaltenorientierte-Datenbanken anhand Ihrer Funktionen, Vor- und Nachteile und Einsatzgebiete behandelt.

## Key-Value-Datenbanken (Schlüssel/Wert-Speicher)

Key-Value (KV), auch als Key-Data Store bekannt, gehört zu den ältesten NoSQL-Datenbanken. Schon in den 1970er Jahren waren sie in verschiedenen Betriebssystemen im Einsatz, dabei jedoch nicht standardisiert. Die Entwicklung des Webs 2.0 hat dazu geführt, dass Key-Value zunehmend Anwendung finden [Selt15]. Redis, Tokyo Cabinet, Chordless, MencacheDB, Voldemort zählen zu den wichtigsten Vertretern.

### Key-Value Datenmodell

Key-Value Datenbanken referenzieren einen Schlüssel auf einen Wert, so wie es eine Map (oder Hashtabelle) bei einigen bekannten Programmiersprachen darstellt.

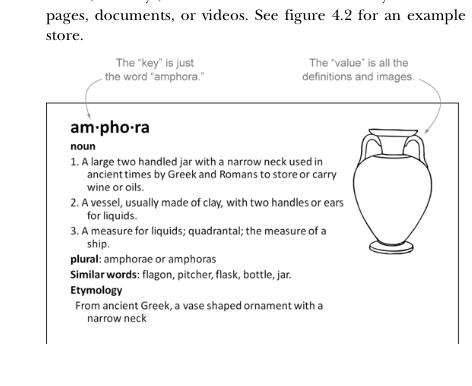


Abbildung ‑ Beispiel Wörterbuch. Schlüssel: Werte [McKe14]

Der Einsatz ist einfach und funktioniert wie bei einem Wörterbuch, wie die Abbildung 2-1 zeigt.

Der Schlüssel ist das Wort *amphora* und der Wert ist das Bild und alle Definitionen. Daten werden zusammen mit einem Schlüssel gespeichert als BLOB (binary large objects).

Moderne Arten dieser Datenbank können nicht nur Strings, sondern auch andere Datentypen von reinem Text, JSON, XML bis zu Images oder Video-Clips als Wert speichern (vgl. Abbildung 2-2). Key-Value Datenbanken haben keine Abfragesprache, sie bieten aber durch Methoden wie *put(key,value), get(key,value), remove(key,value)* die Möglichkeit, Schlüssel-Werte hinzuzufügen, zu entfernen oder zu suchen. Abfragen und Löschoperationen können nur über den eindeutigen Schlüssel des Wertes geschehen.

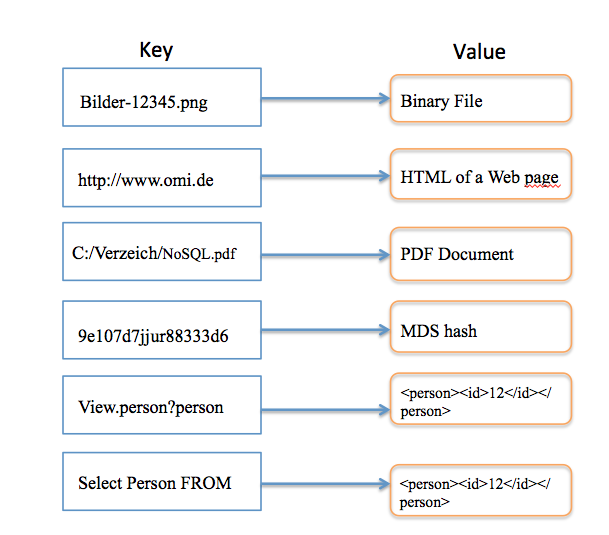


Abbildung ‑ Key-Value Datenmodell [McKe14]

Bei Key-Value ist die Wahl einer Datenstruktur dem Entwickler überlassen. Das System speichert alles wie ein BLOB und bei Abfrage mit *get(key, value)* enthält man das BLOB wieder zurück.

### Anwendungsfall Speichern von Webseiten

Täglich werden in Suchmaschinen Tausende von Abfragen gemacht. Es ist aber nicht trivial zu verstehen, nach welchen Prinzipien diese Suche funktioniert. Suchmaschinen wie z.B. Google benutzen Webcrawler (Spider, Searchboot), eine automatisierte Engine, die das WWW (World Wide Web) durchsucht und Webseiten analysiert. Wörter auf jeder Seite werden somit für eine schnelle Suche indiziert [Harb10]. In der Regel sind URL (Uniform Resource Locator) eindeutig und werden deswegen als Schlüssel und der Inhalt der Seite als Wert betrachtet. Das Web kann insofern als eine große Tabelle mit zwei Spalten gesehen werden (vgl. Tabelle 2-1).

Diese Eigenschaft von Key Value, URL als Schlüssel zu speichern, erlaubt es, alle statischen Komponenten einer Webseite zu speichern.

|  |  |
| --- | --- |
| **Key**  **(Schlüssel)** | **Value**  **(Werte)** |
| <https://moodle.oncampus.de/login/index.php> | <html>....... |
| <http://www.hs-emden-leer.de/startseite.html> | <html>....... |
| <http://www.codecademy.com/learn> | <html>....... |
| <https://www.mongodb.org/?_ga=1.145625718.351634351.1424968924> | <html>....... |

Tabelle ‑ URL als Key und Seite als Value [McKe14]

### Vor- und Nachteile

Einer der Vorteile von Key-Value liegt in der enormen Geschwindigkeit. So erreicht z.B. redis Laufzeiten von O(1) für [Lese](http://redis.io/commands/get)- und [Schreiboperationen](http://redis.io/commands/set). Dafür muss man allerdings auch auf komplexe Suchalgorithmen oder Indizes verzichten; der Zugriff auf einen Datensatz erfolgt lediglich über dessen Schlüssel.[Edli12].

Key-Value bieten eine einfache Partitionierung und Replikation. Da auf jeden Eintrag nur über den Schlüssel zugegriffen werden kann, ist eine Verteilung des ganzen Datensatzes auf mehrere kleinere Server kein Problem.

### Key Value Vertreter

Unter Key-Value existiert eine Reihe von Lösungen. Redis und Riak sind die bekanntesten. Jedoch sollte man folgende Fragen berücksichtigen, wenn man sich für Key Value entscheidet.

* **Datenstrukturen**: Werden über einfache Key-Values hinweg weitere Datenstrukturen unterstützt (z.B. Listen oder Sets)?
* **Speicherung:** Werden die Daten volatil, z.B. im RAM oder persistent auf der Festplatte vorgehalten?
* **Sharding und Partition:** Wird über Sharding, Master-Slave oder Master-Master skaliert?
* **Konsistenzmodell:** Bietet die Implementierung strenge Konsistenz oder nur eventuelle Konsistenz?

### Fazit

Schlüssel/Wert ist das einfachste Modell in der Gattung der NoSQL-Datenbanken. Diese Technologie bildet einfache Schlüssel auf komplexe Datenstrukturen ab. Schlüssel/Wert sind sehr gut für verteilte Systeme geeignet und lassen sich gut skalieren.

Key-Value-Datenbanken eignen sich besonders für Anwendungen, in denen Daten effizient über einen Key aus einer großen Datenmenge selektiert bzw. gespeichert werden müssen. Beispiel dafür sind die Speicherung und Abfrage von Nutzer Profilen, Warenkorbdaten im Web. Auch für gute Speicherung von Session-Informationen sind solche System gut ausgerüstet Tabelle 2-3 fasst alle Eigenschaften von Schlüssel/Wert zusammen.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Key-Value** | | | | | |
| **Darstellung/**  **Format der Information** | **Präzise**  **Abfragen/ Sek.**  **Indizes** | **Konsistenz-**  **Methoden** | **Skalier-barkeit** | **Performanz** | **Schema** |
| sehr verschieden, häufig  BLOB | i.d.R. keine | i.d.R.  schwach  (Memory optimistic  concurency control) | sehr hoch  Replikation oder  Scharding | sehr hoch | schemalos Ausnahme Redis (String, List, Sets) |

Tabelle ‑ Key-Value Überblick [Edli12]

## Graph-Datenbanken

Als Edgard F. Codd 1970 relationale Datenbanken entwickelt hat, modellierte er alles als Tabelle mit Beziehungen. Heute ist diese Darstellung nicht mehr ausreichend. Wenn man Objekte analysiert, seien es Computer, Software, Moleküle und Gene, soziale Netze, Ereignisse aus Geschichte usw. zeigt sich, dass diese miteinander in Beziehungen stehen und somit ein Netzwerk bilden. Graph-Datenbanken sind dazu optimiert, solche einander verbindenden Informationen effizient abzuspeichern und greifbar zu machen.

Seit Leonard Euler die Graphentheorie (1735) über die Sieben Brücken von Königsberg [Brüc17] erläutert hat, ist bekannt, dass Graphen in vielen Bereichen anwendbar sind. Besonders in der Mathematik haben Graphen eine größere Bedeutung gewonnen. Graphen-Algorithmen, wie die Wege-Suche, werden genutzt, um praktische Probleme zu lösen.

Graph-Datenbanken verwenden Graphen, um Informationen abzuspeichern sowie darzustellen. Statt herkömmlicher Datensätze erstellt man hier Knoten, die durch die Beziehungen (Kanten), die man zwischen ihnen definiert, miteinander verknüpft werden []. In der Informationstechnik lassen sich Netzwerke als Graph darstellen, wobei Knoten Computer, Switch oder Router und Kanten Verbindungen entsprechen können.

Wie Tabelle 3.3 darstellt, ist es möglich, eine Graph-Datenbank in eine relationale Datenbank zu modellieren. Jedoch sollte man dabei mit vielen Joins rechnen, die sich bei steigender Komplexität der Beziehung und des Datenaufkommens nur sehr zeitintensiv ausführen lassen.

|  |  |
| --- | --- |
| **Relationale Modelle** | **Graph-Modelle** |
| Tabelle | Knoten/Kanten |
| Reihe | Eckpunkte |
| Spalten | Schlüssel/Wert-Paare |
| Joins | Kanten |

Tabelle ‑ Relationale Modelle im Vergleich zu Graph-Modellen

Komplexe Abfragen lassen sich schnell mit Graph Datenbankmodellen effizient beantworten. Die Abbildung 2-3 stellt ein Graphen-Datenmodell für Schauspieler und deren Filme dar. Dieses Modell kann in ein RDBMS transformiert werden, in dem die folgenden drei Tabellen gebildet werden: “Person“, “Film“ (Movie) und für die Kante eine Tabelle “Beziehungen“. Kanten als auch Knoten können Key-Value Paare enthalten.

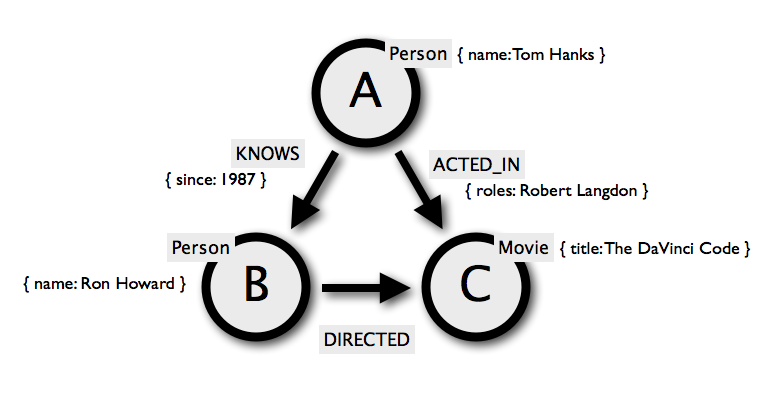


Abbildung ‑ Graphen Datenbank Modell [Neo415]

Es gibt eine Reihe von unterschiedlichen Graph-Datenmodellen. Im Folgenden werden zwei der am häufigsten eingesetzten Vertreter dargestellt: das RDF-Triplestores (Ressource Description Framework) und das Property Graph-Model.

### Datenmodelle

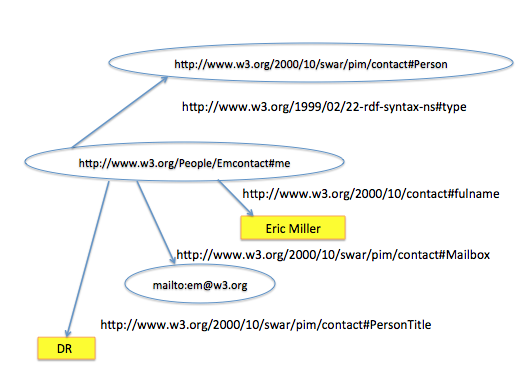
Das RDF-Triplestores und das Property Graph-Model unterscheiden sich in der Art und Weise, wie Eigenschaften und Kanten zwischen Knoten benannt und behandelt werden [Geye15]. Das Konzept von „Semantic Web“ oder „Linked Data“ basiert auf einem Graphen-Modell, das implizit aus Fakten entsteht. Dabei stellt jeder Fakt ein Tripel aus Subjekt, Prädikat und Objekt dar. Elemente werden als URIs oder Links abgebildet, damit man sie gut identifizieren kann. Ein RDF ist immer vollständig normalisiert, da jede Information als Referenz modelliert wird.

Abbildung ‑ Linked-Data-(RDF) – Repräsentation einer Kontaktinformation [NeHu15]

Das Property Graph-Model ist eine pragmatische Form von Graph-Datenmodellen. Es handelt sich dabei um eine Erweiterung des allgemeinen Graph Modells, in dem jede Kante ein Label trägt, die den Typ der Relation zwischen den Knoten beschreibt. Außerdem erhalten sowohl Knoten als auch Kanten einen eindeutigen Identifikator und können Eigenschaften („properties“) tragen, welche durch eine Zuordnungstabelle des Typs <String, Objekt> verkörpert werden. Je nach Implementierung können Knoten und Kanten streng typisiert werden, sodass der Typ des Knotens oder der Kante explizit erfasst oder implizit aus seinen Eigenschaften bestimmt werden kann [Geye15].

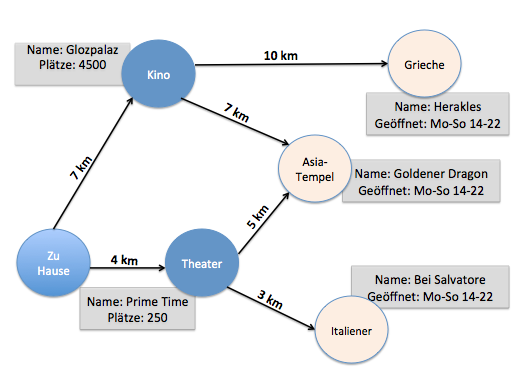


Abbildung ‑ Property-Graphen mit Knoten und Kanten [Geye15]

### Abfragesprache

Mit einer SQL-Abfrage kann man alle Knoten oder Kanten (Beziehungen) mit vorgegebenen Eigenschaften herausfinden. Um alle indirekten Beziehungen zu finden oder Pfade zwischen zwei Knoten zu bestimmen, können Methoden wie Common Table Expression (CTEs) eingesetzt werden. Dagegen verwenden die Graphen-Datenbanken performante traversierte Algorithmen (Breitsuche, Tiefsuche, kürzester Pfad) zur Selektion bestimmter Knoten []. Ausgehend von einem oder mehreren Knoten werden alle oder ausgewählte ausgehende Kanten traversiert. Es gibt unterschiedliche Wege, Daten abzufragen und zu manipulieren, jedoch bieten die meisten Graph-Datenbanken APIs, um mit den Anwendungen zu kommunizieren.

### Anwendungsfälle

Graph-Datenbanken sind geeignet für Anwendungsfälle, bei denen die Beziehung zwischen den Datensätzen eine große Rolle spielt und eine Navigation zwischen den Knoten ermöglicht werden soll [Hung15]. Folgende Szenarien lassen sich einfach mit Graph-Datenbanken abbilden:

* Geokoordinaten: zwei- oder dreidimensionaler Baum (R-Tree), erlaubt Zugriff auf Koordinaten oder Flächen über Bounding Boxes auf verschiedenen Ebenen.
* hierarchische Strukturen (z.B. regionale Geographie, Organisationsstrukturen, Produktkategorien): einfache Baumstrukturen.
* Schlüsselworte, Hashtags: Netzwerk von Tags, die Entitäten markieren und auch miteinander verknüpft sein können
* Versionierung: verschiedene Ansätze, zum Beispiel versionierte Teilbäume, Versionsinformationen auf Beziehungen, Repräsentation von Änderungsoperationen (Git)
* Newsfeed eines Nutzers: Update aller Nutzer bilden eine verlinke Liste, pro Nutzer gibt es einen eigenen Beziehungstyp, der die Updates der von ihm abonnierten Nutzer oder Gruppen verknüpft.

Die Tabelle XX gibt einen Überblick über Funktionen und Eigenschaften von Graph-Datenbanken.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Graph-Datenbanken** | | | | | |
| **Darstellung/ Format der Information** | **Präzise Abfragen/ Sek. Indizes** | **Konsistenz-Methoden** | **Skalier-barkeit** | **Performanz** | **Schema** |
| Knoten und Kanten | Präzise Abfragen möglich / Sek. Indizes werden unterstützt | Hoch (MVCC aber auch mittels Write-Locks) | Sehr hoch:  Replikation oder  Scharding | Sehr hoch (oft auch bei höherer Serverzahl u. tiefen Anfragen) | Schemafrei |

Tabelle ‑ Überblick Graph-Datenbanken [Edli12]

## Spaltenorientierte Datenbanken

Wie funktionieren spaltenorientierte Datenbanken? Und wie gehen sie mit den Problemen größerer Datenmenge um? Um diese Fragen zu beantworten, ist es wichtig zu verstehen, wie relationale Datenbanken funktionieren.

### Zeilenorientierter Ansatz (relational Datenbanken)

Daten (Objekte oder Instanzen) werden in relationalen Datenbanken in Zeilen gespeichert (siehe Tabelle XY). Außerdem werden diese Daten wie in Tabelle XZ auf den Disk gespeichern.

|  |  |  |
| --- | --- | --- |
| **Name** | **Geschlecht** | **Alter** |
| Foo | weiblich | 22 |
| Max | männlich | 33 |
| Karl | männlich | 25 |
| Laureta | weiblich | 32 |
| Lara | weiblich | 10 |
| ... | ... | ... |

Tabelle ‑ Zeilenorientierter Ansatz Objekt „Person“

Das Objekt „Person“ wird, wie in unten dargestellter Tabelle, auf den Speicher repräsentiert werden

|  |  |
| --- | --- |
| Stelle 00 | | F o o \0 w e i b l i c h \0 22 |
| Stelle 25 | | M a x \0 m ä n n l i c h \0 33 |
| Stelle 35 | | K a r l \0 m ä n n l i c h \0 25 |
| Stelle 49 | | L a u r e t a \0 w e i b l i c h \0 32 |
| Stelle 80 | | L a r a \0 w e i b l i c h \0 10 |

Tabelle ‑ Objekt Person im Speicher (relationale Datenbanken)

Es ist hier schwer vorherzusehen, wo ein neuer Datensatz beginnt und an welcher Stelle eine bestimmte Eigenschaft zu finden ist. An diesem Beispiel lässt sich schwer erahnen, an welcher Position im Speicher z.B. Lara zu finden ist. Um solche Operationen durchzuführen, ist es notwendig, den ganzen Speicherbereich durchzusuchen. Ein Versuch, diese Daten zu analysieren, kann sehr teuer und ineffizient sein. Wenn man z.B. versucht, das durchschnittliche Alter zu berechnen, muss die ganze Datei Person.text gelesen und geparst werden, obwohl jedoch nur ein Bruchteil der Informationen dieser Datei, nämlich das Alter, interessiert. Spaltenorientierte Datenbanken bieten einen besseren Ansatz, um mit diesen Problemen effizienter umzugehen.

### Spaltenorientierter Ansatz

Das Prinzip des spaltenorientierten Ansatzes ist zu vergleichen mit einem Container oder Array (Spalte). Daten werden als Schlüssel-Wert-Relation, auch Key/Value oder Tupel genannt, abgelegt. Die spaltenorientierten Datenbanken speichern jedes Attribut in einer eigenen Tabelle hintereinander und nicht wie bei relationalen Modellen, die untereinander speichern (Reihe) [Edli12]. Ähnliche Inhalte werden gruppiert und jede Zelle ist eindeutig mit einem Schlüssel identifizierbar. Spaltenorientierte Datenbanken können mehrere Spalten haben und im Gegensatz zu den Tabellen in relationalen Datenbankensystemen, muss sie nicht die gleiche Struktur haben wie die andere Zelle, die der gleichen Familie angehören. Normalweise sind die spaltenorientiert Datenbanken Schemafrei.

Im Speicher wird das Objekt „Person“ auf den Beispiel xx wie unten repräsentiert

|  |  |
| --- | --- |
| Stelle 00 | | F o o \0 M a x \0 K a r l \0 L a u r e t a \0 L a r a \0 |
| Stelle 25 | | w e i b l i c h \0 m ä n n l i c h \0 m ä n n l i c h \0 w e i b l i c h \0 w e i b l i c h \0 |
| Stelle 35 | | 22 33 25 32 10 |

Tabelle ‑ Objekt Person im Speicher (spaltenorientierte Datenbanken)

Im Gegensatz zur relationalen Darstellung ist es einfach vorherzusagen, wo die Person mit Namen Lara zu finden ist (Lara in Array Index 4). Sobald Lara gefunden ist, kann berechnet werden, wo genau ihr Alter zu finden ist. Um eine Analyse durchzuführen, können die Dateien Name.txt, Alter.txt, Geschlecht.txt abgebildet werden (siehe Abbildung xy).

Datei Name.txt

Foo

Max

Karl

Lauretta

Lara

Datei Geschlecht.txt

Weiblich

Männlich

Männlich

Weiblich

Weiblich

Datei Alter.txt

22

33

25

32

10

Abbildung ‑ Beispiel Datei in spaltenorientierten Datenbanken

### Vor- und Nachteile

Wenn man versucht, große Datenmengen zu analysieren oder zu komprimieren, hat dieses Verfahren viele Vorteile. Es müssen weniger Daten gelesen werden und die Wahrscheinlichkeit für einen Cache-Hit ist höher. Diese Datenbanken haben sehr gute Möglichkeiten, Daten zu aggregieren und werden gern von OLAP und Data-Warehouse in Anspruch genommen.

Dagegen kann die Suche und das Einfügen von Daten sehr aufwendig sein. Weiterhin kann das Lesen von Objektstrukturen (zusammengehörige Spalten) auch zeitintensiv sein, da der Speicher detailliert durchsucht werden muss.

### Einsatzbereiche

Spaltenorientierte Datenbanksysteme finden ihren Einsatz für analytische Informationssysteme, bei denen eine direkte Verarbeitung von Informationen benötigt ist.

Beispiele hierfür sind u.a. Data-Mining Systeme, Business Reporting für Vertrieb/Marketing, Business Process Management Systeme sowie Systeme für Finanzberichte und Budget-Projektionen.

Cassandra und HBase dominieren zurzeit den Markt. HBase ist als Open-Source bei Google implementiert worden und galt als erstes System dieser Kategorie.

### Fazit

Spaltenorientierte Datenbanken genießen einen guten Ruf im Bereich der NoSQL-Datenbanken, aufgrund ihrer Fähigkeiten gut zu skalieren und große Datenmengen zu speichern. Sie überzeugen durch robuste Skalierbarkeit, Hochverfügbarkeit und können massive Schreibvorgänge gut leisten. Sie sind auch bekannt wegen ihrer Fähigkeiten mit Frameworks wie Map/Reduce gut aggregieren zu können. Die Tabelle xx zeigt eine Zusammenfassung von Eigenschaften und Funktionen spaltenorientierter Datenbanken.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Spaltenorientierte Datenbanken** | | | | | |
| **Darstellung/ Format der Information** | **Präzise Abfragen/ Sek. Indizes** | **Konsistenz-Methoden** | **Skalier-barkeit** | **Performanz** | **Schema** |
| Spalten und Attribute (Rows) / oft JSON oder BSON | Präzise Abfragen möglich / Sek. Indizes werden unterstützt | Mittel – Hoch (variiert auf Grund des Einsatzes versch. Methoden) | Sehr hoch:  Replikation oder  Scharding | Sehr hoch | Schemafrei |

Tabelle ‑ Überblick Spaltenorientierte Datenbanken [Edli12]

## Dokumentorientierte Datenbanken

Während relationale Datenbanken im Wesentlichen auf Entitäten beruhen - dies entspricht einer Tabelle mit Spalten und Zeilen und Relationen, also Beziehungen zwischen diesen Entitäten - arbeiten dokumentorientierte Datenbanken dagegen mit Kollektionen, die Dokumente enthalten (siehe Abbildung). Mit Dokument sind nicht Textdateien oder Word-Dokumente gemeint, sondern strukturierte Datensammlungen wie JSON, YAML oder RFD-Dokumente. Das Datenbankmodell ist schemalos, dass bedeutet, dass im Vorfeld keine Aussagen getroffen werden müssen, wie ein Datenobjekt aufgebaut ist.

Ein Dokument ermöglicht es, komplexere Strukturen in einem Objekt zu speichern. Selbst für umfangreichere Anwendungen kann dies bedeuten, dass für einen Datensatz, welcher in einer relationalen Datenbank auf mehrere Tabellen verteilt abgelegt werden muss, ein Dokument ausreicht, um die Informationen zu speichern. Dadurch werden ressourcenhungrige Joins über mehrere Tabellen hinfällig [Jans10]. Vertreter dieser Datenbank sind u.a. MongoDB, CouchDB, wobei beide in starker Konkurrenz auf dem Markt stehen. Im Kapitel 5 wird an praktischen Beispielen die Datenbank MongoDB erläutern.

### Anwendungsfälle

Dokumentorientierte Datenbanken sind besonders geeignet, sobald eine große Menge und zusammenhängende Datenstrukturen mit unbestimmten Formaten gespeichert werden müssen. Anwendungsfälle wären beispielweise Blog, CMS oder Wiki.

Blogs bestehen aus Blogeinträgen, die sich aus langen Texten, Kommentaren, Tags, Video, oder Fotos zusammen setzen und in einzelnen Dokumenten gespeichert werden können. Dadurch könnten Blogeinträge aus der Datenbank gelesen werden, indem man nur das passende Dokument betrachtet wird.

Solche Operationen sind in relationalen Datenbanken relativ schwer zu realisieren. Hierfür müsste man verschiedene Blogeinträge auf verschiedene Tabellen verteilen und diese bei einer Abfrage wieder durch Joins-Operationen abfragen. Darüber hinaus wäre der Versuch, Blogeinträge zu entfernen, ebenfalls komplizierter als mit dokumentorientierten Datenbanken.

### Fazit

Dokumentorientierte Datenbanken sind geeignet, wenn man Daten nicht stark vernetzen muss und die einzelnen Daten unterschiedliche Strukturen haben. Dieses schemalose Datenmodell erlaubt, dass jedes Dokument verschiedene Schlüssel enthalten und auch flexible und unterschiedliche Strukturen besitzen kann. Das bedeutet für den Entwickler eine große Freiheit. Jedoch bringt diese Freiheit auch viele Nachteile mit sich. Zum Beispiel muss die Struktur der Dokumente vom Entwickler selbst festgelegt und kontrolliert werden. Die Tabelle xx zeigt eine Zusammenfassung von Eigenschaften und Funktionen dokumentorientierter Datenbanken.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Dokumentorientierte Datenbanken** | | | | | |
| **Darstellung/ Format der Information** | **Präzise Abfragen/ Sek. Indizes** | **Konsistenz-Methoden** | **Skalier-barkeit** | **Performanz** | **Schema** |
| Dokumente und Anhänge (JSON oder BSON) | Präzise Abfragen möglich / Sek. Indizes werden unterstützt | Mittel hoch (häufig Update-in place oder MVCC) | Sehr hoch:  Replikation oder  Scharding | I. d. R. hoch (abhängig von Konsistenz- Methode) | Generell schemalos, oft zusätzl. unterstützende  Gruppierungs-funktion wie Collections oder Views |

Tabelle ‑ Überblick Dokumentorientierte Datenbanken [Edli12]

## Zusammenfassung

NoSQL Datenbanken lassen sich nach ihrer Basistechnologie und nach dem CAP-Theorem kategorisieren. Diese Datenbanken unterscheiden sich hinsichtlich ihrer Speicherstrukturen, wobei die wichtigsten Strukturen Dokumenten-DB, Key-Value-DB, Graphen-DB und Spaltenorientierte-DB sind.

Im Gegenteil zu klassischen “one-size-fits-all“ relationalen Datenbanken, sind NoSQL anwendungsbezogen. Sie stellen bei der richtigen Verwendung eine sehr gute Verfügbarkeit, Ausfalltoleranz und horizontale Skalierbarkeit bereit. In der Darstellung XX hat [Hurs10] eine Entscheidungspyramide zu NoSQL aufgestellt, welche das CAP-Theorem visuell erklärt. Die meisten relationalen Datenbanken erfüllen die Konsistenz (C) und die Verfügbarkeit (A) und werden bei der Ausfallsicherheit (P) einbüßen. Dagegen spielt bei NoSQL der Einsatzzweck eine wichtige Rolle bei der Auswahl. Es resultiert, dass die meisten NoSQL nicht ACID konform sind. Jedoch basieren solche Systeme stark auf dem Konsistenzmodell BASE.

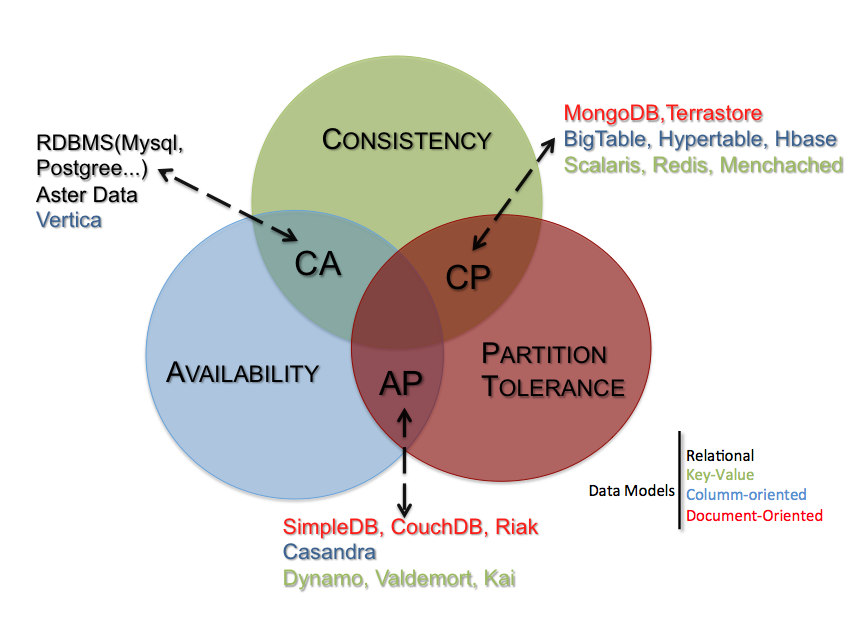


Abbildung ‑ Entscheidungspyramide nach Hurst [Hurs10]

Die Tabelle XX stellt eine –nicht vollständige– Übersicht über verschiedene NoSQL-Datenbanken, ihre Funktionen, Vor- und Nachteile sowie Einsatzgebiete dar.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **CouchDB** | **MongoDB** | **Redis** | **Cassandra** | **Riak** | **HBase** |
| **Programmiert in** | Erlang | C++ | C/C++ | Java | Erlang & C | Java |
| **Lizenz** | Apache | AGPL / Apache | BSD | Apache | Apache | Apache |
| **Hauptvorteil** | DB-Konsistenz & einfache Benutzung | Verwendet Abfragen und Indizierung auf SQL-Basis | Geschwindigkeit | Große Tabellen | Fehlertolerenz | Riesige Tabellenstrukturen |
| **Protokoll** | HTTP /REST | proprietär | Ähnlich Telnet | Proprietär und Thrift | HTTP/ REST | HTTP/ REST  Und Thrift |
| **Art** | Dokumentorientiert | Dokumentorientiert | In-Memory | Spaltenorientiert | Spaltenorientiert | Key-Valueorientiert |
| **Besonderheit** | Bidirektionale Replikation | Master Slave Replikation | Master Slave Replikation | Trade-offs für Verteilung und Replikation | Trade-offs für Verteilung und Replikation | Entworfen wie Google BigTable |
| Konflikt Erkennung | Abfrage sind JavaScript Ausdrücke | Einfache Schlüssel Werte | Trade –Offs Anpassbar | Trade –Offs Anpassbar | Vordefinierte Abfragen |
| MVCC Blockieren keine Lesezugriff | Komplexe Daten Operationen | Datenüberprüfung | Optimierung von Echtzeitabfragen |
| Versionierung der Daten | Serverseitige  JavaScript | Verwendet Sets, Lists, Hashes | Abfrage nach Spalten | Datensicherheit | Performance Thrift Gateway |
| Serverseitige Validierung der Dock | Unterstützt Transaktion |  |
| Authentifizierung | Verteilte Daten (Sharding) | Ablaufdatum für Daten | Abfrage nach Indexbereichen | Datensicherheit | XML mit HTTP Unterstützung |
| Echtzeit Updates | Sortierte Sets | Volltextsuche |
| Attachement Verwaltung | Teilweise Ablage der Daten im Speicher | Überwachen von Datenänderungen | Schreibt schneller als es liest | Commits | Wahlfreier Zugriff auf Daten |
| JQuery Bibliothek |  | Enterprise und Open Source vers |
| **Nachteile** | Wiederholte Komprimierung notwendig | Nach einem Crash müssen die Kollektion repariert werden | Erst Vers 2.0 kann auf die Disk auslagern | Schreibt schneller als es liest | Verteilte Daten nur mit der Enterprise Version | Performance des Wahlfreien Zugriffs ähnlich SQL |
| Datenbankgöße sollte vorhersehbar sein | Übernimmt Komplexität von Java |
| **Einsatz Gebiete** | CRM | Dynamische Abfragen | Häufige Schreibzugriffe | Echtzeit Datenanalyse | Hochverfügbarkeit | Große Tabellen mit Milliarden Datensäzten und Millionen Feldern |
| Wenig Schreibzugriff | Indizierung der Daten | Schnelle Änderung der Daten | Alle Systembestandteile in Java sein sollen | Schnelle Schreibzugriffe |
| Häufige Lesezugriffe | Große DB mit Performance | Echtzeitverarbeitung | Seltene Lesezugriffe | Echtzeitzugriff auf die Daten |
| Versionierung | Ähnlich SQL ohne definierte Spalten | Statistiken | Loggin | Datenanalyse | Wahlfreier Datenzugriff |

Tabelle ‑ Überblick über populärsten NoSQL –Datenbanken und Einsatz[Gull11]

# Einsatzmöglichkeiten und Grenzen von NoSQL im Hinblick auf Big-Data

[**www.infoq.com/presentations/choosing-NoSQL-NewSQL**](http://www.infoq.com/presentations/choosing-NoSQL-NewSQL)

## Auswahlkriterien für eine NoSQL - Datenbank

Bei der Auswahl einer Datenbank muss man zuerst untersuchen, ob eine NoSQL-Datenbank benötig wird oder ob nicht eine normale relationale Datenbank die beste Lösung ist. In beiden Bereichen muss man sich jedoch in der Vielfalt an Angeboten zurechtfinden.

NoSQL ist eine sehr junge Technologie und hat dementsprechend auch Nachteile. Viele Angebote sind oft noch nicht ausgereift. Während man noch überlegt oder versucht zu vergleichen, gibt es wieder etwas Neues oder bestehende Features sind obsolet (deprecated). Deswegen sollte man das technische Risiko durch eine Vielzahl von Kriterien bei der Auswahl reduzieren.

### Kriterienkatalog

Grundsätzlich muss der Entscheidung eine gründliche Analyse der Anwendung im Hinblick auf das erwartete Datenvolumen (Gigabyte bis Terabyte), die Komplexität von Daten, die Art der Navigation zwischen Daten, die Konsistenzanforderungen, die erwarteten Abfragen sowie die Performance-Anforderungen bezüglich Latenz und Skalierbarkeit vorausgehen. Darüber hinaus sind auch nicht-funktionale Kriterien wie Lizenzfragen, Kosten, Sicherheit, Support etc. zu berücksichtigen [KuSt15].

In einem Feld wie NoSQL, das sich schnell ändert, ist es sehr wichtig zu untersuchen, wie das System migrierbar ist, d.h. wie man ohne großen Aufwand zu einer anderen Datenbank wechseln kann.

### Performance

Auf Grund von Big-Data (große Datenmenge) spielt die Performance von Datenbanken eine wichtige Rolle. Bei den traditionellen relationalen Datenbanksystemen existieren etablierte Benchmarks, die für unterschiedliche Anwendungsszenarien entsprechende Datenbankschemata, Testdatenprofile und Abfragen definieren. Leider existieren solche Benchmarks nicht für NoSQL-Datenbankensysteme. Das liegt daran, dass die Technologie noch in den Anfängern steht und es keine Gremien gibt, die für Standards zuständig ist.

## Transaktionsanwendungen

### Anforderungen an Datenmodelle

### Datenzugriffsanforderung

### Fazit

## Rechnungsanwendungen

### Anforderungen an Datenmodelle

### Datenzugriffsanforderung

### Fazit

## Webanwendungen

In die Webanwendungen sollten Applikation folgende Eigenschaften haben

Moderne Webanwendungen müssen im Idealfall zudem echtzeitfähig sein, um den Benutzer so schnell wie möglich mit neuen Informationen zu versorgen.

Alle diese Variante müssen für eine Moderne Technologie erfüllt werden.

Webanwendungen laufen am meisten über das Internet und sind von einer unbekannten Anzahl von Benutzer Nutzbar, woraus Anforderungen an diese Anwendungen und ihre Umsetzung resultieren.

**Hohe Flexibilität** (Webseite ändern sich ständig, und die Datenstruktur muss auch dementsprechend angepasst werden.)

**hohe Verfügbarkeit** (Webseite um erfolgt zu haben müssen immer Verfügbar sein)

**hohe Skalierbarkeit** ( Eine Webseite kann schnell mit Daten gefüllt werden, besondere bei selbst-Content editieren Webseite wie Blog). Die müssen das C10K-Problem lösen und 10.000 Clients gleichzeitig bedienen zu können.

**Moderne Webanwendungen** müssen eine hohe Performance Aufweisen, so dass sie einer nativen Anwendung weder in Komfort noch in Reaktivität nachstehen

**Sicherheit**

### Anforderungen an Datenhaltung

### Datenzugriffanforderung

### Zusammenfassung

Überblick Anwendungsfälle

# Webanwendung mit MongoDB

Wenn man in Suchmaschine nach dem Begriff NoSQL sucht, trifft man am häufigsten das Wort MongoDB. Dieses Datenbanksystem ist neben CouchDB ein starker NoSQL Vertreter der dokumentenorientierten Datenbanken. Diese Datenbanken lassen sich für große und kleine Anwendungen leicht einsetzen und sind sehr beliebt in der Webanwendung. MongoDB bietet für Webapplikationen, in denen eine große Anzahl von Daten, die nicht transaktionsbasiert vorgehalten werden müssen, eine hohe Performanz, skalierbare, schemafreie und dokumentenorientierte Lösungen. [Alve11]. MongoDB ist sehr beliebte bei Webentwickler wegen seine Ähnlichkeiten an SQL.

Blog sind allgemein selbst-User generated Content Anwendungen, das bedeutet jeder in der Lage ist Interactive mit den Anwendungen zu interagiert. Außerdem auf Blog verkehren nicht nur großen Datenmenge sondern auch unstrukturierte Daten. In diese Kapitel soll untersucht werden wie solche Anwendungen einfach mit MongoDB persistieren werden sollen. Dabei werden die CRUD Operation durchgeführt, die Funktionalität von Map/Reduce vorgestellt werden.

## Wer nutz MongoDB

MongoDB hat sich rasant in der Datenbankwelt etabliert und ist sehr beliebt und gefragt bei Web Unternehmen. Wie bereits Abbildung 9 im Kapitel 4.2 andeutet, bietet MongoDB eine Fülle von Features und positioniert sich zwischen den schnellen, aber feature-ärmeren und unflexibleren Schlüssel-Wert-Speichern und den komplexen, schwergewichtigeren relationalen Datenbanken.

Viele bekannte Unternehmen vertrauen auf MongoDB für verschiedene Geschäftsmodelle:

* **ING Entertainment**: Die Computerspiele- und Unterhaltungsseite analysiert in Echtzeit den Seiten-Verkehr mit MongoDB.
* **GitHub:** Die social Coding-Plattform setzt MongoDB für internes Reporting ein.
* **Sourceforge:** Die Projekt-Webseiten und die Download-Seiten aller Projekte werden mit MongoDB verwaltet.
* **New York Times:** Hier wir MongoDB für ein frei konfigurierbares Foto-Upload-Formular verwendet. Auf Grund der Schemalosigkeit der Dokumente kann jeder Anwender sein eigenes, maßgeschneidertes Formular erstellen.
* **Foursquare:** Ein ortsabhängiges soziales Netzwerk, mit dem Nutzer an Orten virtuell einchecken und zu diesen Orten Tipps und Bewertungen schreiben. Foursquare setzt MongoDB exklusiv für die Eincheck-Funktion (“Who’s here?“) ein.
* **SAP und MTV:** Nutzen MongoDB für CMS.
* **Springer:** Speichert Artikel und Bücherkapitel für den real time-Download und Anzeigen in verschiedenen Formen.

Von Anfang an wollten die Gründer von MongoDB besonders die Webapplikation adressieren, deswegen lassen sich mit dieser Datenbank Aufgaben wie Dokumentenmanagement, das Verwalten von Anwender- und Sitzungsdaten, Logging, Echt-Zeit-Analysen und generell Aufgaben mit einem hohen Datenaufkommen einfach und bequem lösen. MongoDB ist nicht geeignet für Systeme mit komplexen Transaktionen und traditionelle Business-Intelligence-Projekte, da die dafür nötigen Strukturen nicht zur Verfügung stehen [Alve11].

## Überblick über MongoDB

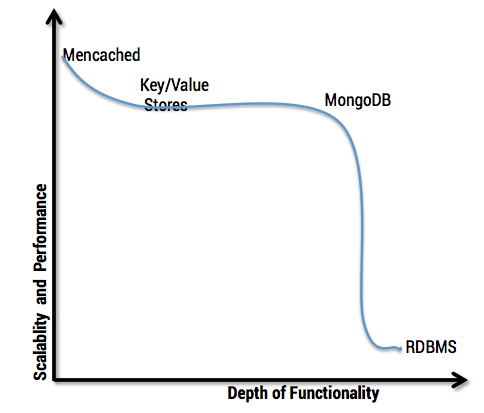
Eliot Horowitz, CTO und CO-Founder von MongoDB, hat die Philosophie dieses Datenbanksystems wie folgt erläutert:

*„MongoDB wasn’t designed in a lab. We built MongoDB from our own experiences building large scale, high availability, robust systems. We didn’t start from scratch, we really tried to figure out what was broken, and tackle that. So the way I think about MongoDB is that if you take MySQL, and change the data model from relational to document based, you get a lot of great features: embedded docs for speed, manageability, agile development with schema-less databases, easier horizontal scalability because joins aren’t as important. There are lots of things that work great in relational databases: indexes, dynamic queries and updates to name a few, and we haven’t changed much there. For example, the way you design your indexes in MongoDB should be exactly the way you do it in MySQL or Oracle, you just have the option of indexing an embedded field.“[Horo11].*

Der Name MongoDB stammt ursprünglich von dem englischen Begriff “humanous“, was einfach “gigantisch“ oder “wahnsinnig groß“ bedeutet. MongoDB ist eine quelloffene, dokumentenorientierte, verteilte NoSQL-Datenbank mit Ausfallsicherheit und horizontaler Skalierung. MongoDB wurde 2009 von der Firma 10gen als Open-Source-Datenbank unter der Affero General Public Licence veröffentlicht. [affe14]

Der Fokus beim Design von MongoDB lag darauf, Flexibilität, Mächtigkeit, Geschwindigkeit/Skalierbarkeit und einfache Benutzbarkeit in einem System zu vereinigen.

* Flexibilität: Das Ziel hierbei war ein schemaloses Datenmodell anzubieten, dass mit der Datenbankanwendung mitwachsen und sich verändern kann. Der Zugang zum Datenbanksystem sollte auch für den Programmierer vereinfacht werden.
* Mächtigkeit: Wie Abbildung XX verdeutlicht, stellt MongoDB den Anspruch an sich, den Funktionsumfang von SQL anzubieten, gleichzeitig aber auch die Geschwindigkeit und Skalierbarkeit von NoSQL Datenbankensystemen wie Key Value Stores und Mencached zu erreichen. Bei relationalen Datenbankensystemen wird durch Datenbanksperren und Konsistenzbedingungen oft Geschwindigkeit eingebüßt.
* Benutzbarkeit: Es soll einfach, ohne großen Aufwand, ohne Training und praktisch überall möglich sein, eine MongoDB-Instanz laufen zu lassen, sei es auf eigenen Servern, in einem Cloud-Service



### Datenbanken

Ein MongoDB Server enthält eine Reihe von Datenbanken, die als Container von Daten fungieren und unabhängig voneinander sind. Eine MongoDB Datenbank enthält eine oder mehrere Kollektionen (Sammlungen). Jede einzelne Kollektion enthält wiederum Dokumente. Die folgende Abbildung zeigt, wie MongoDB strukturiert ist.

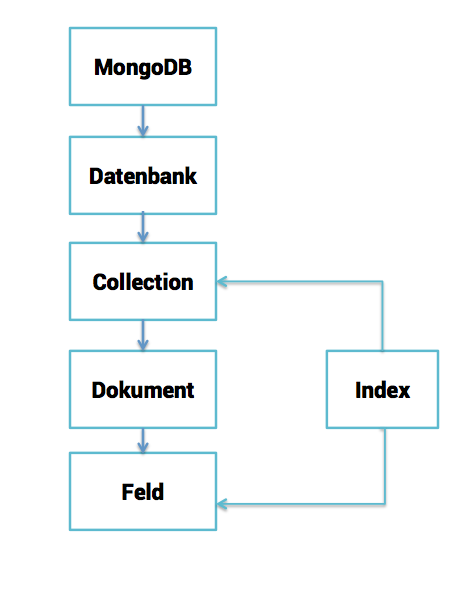


Tabelle ‑ MongoDB Struktur

### Installation und Treiber

Die Einstiegshürde in MongoDB ist sehr niedrig. Die Software ist für jedes Betriebssystem erhältlich und kann ohne großen Aufwand installiert werden. Darüber hinaus bietet MongoDB als erstes Werkzeug eine sehr gute Dokumentation, jedoch nur auf Englisch[Mong15].

Um die Software zu installieren, lädt man sie von der Download-Seite von MongoDB herunter und entpackt sie. Das in C++ implementierte executable mongod (bzw. mongod.exe) für den Datenbankenserver liegt im Unterverzeichnis bin. MongoDB verlangt, dass ein Verzeichnis (db/data) anlegt wird, in dem seine Dateien abgelegt werden. Um MongoDB zu starten, muss man nur das hinterlegte Verzeichnis für die Datenbankdateien auswählen und MongoDB mit folgendem Befehl starten: mongod. Danach horcht der Server Prozess bereits auf Default-Port 27017. Jetzt kann man unter [http://localhost:27017](http://localhost:28017) auf ein rudimentäres Admin-GUI zugreifen. Automatisch ist man mit einer leeren Datenbank, genannt “test“ verbunden. Mit mongod –help kann man mehr über weitere Kommandozeilenoptionen erfahren.

**Storage**

MongoDB verwaltet Nutzdaten und Indizes grundsätzlich im RAM und synchronisiert den Arbeitsspeicher mittels des Betriebssystemdienstes mmnap[Mong15] periodisch mit dem Dateisystem. Beim Erstellen einer Datenbank werden drei Dateien in /data/db angelegt:

*<DBName>.0, <DBName>.1 sowie <DBName>.ns.*

Die <DBName>.ns ist einzigartig für jede Datenbank und enthält Meta-Informationen sowie eine Reihe von Dateien mit Nutzdaten.

MongoDB bietet native Treiber für alle gängigen Programmiersprachen und Frameworks. Ein grundlegender Unterschied im Vergleich zu relationalen Datenbanken ist, dass das MongoDB Abfragemodell als Methode implementiert wird oder Funktionen innerhalb der API von einer bestimmten Programmiersprache zur Verfügung stellt, im Gegensatz zu einer völlig separaten Sprache wie SQL. Dies, gekoppelt mit der Affinität zwischen dem MongoDB JSON-Dokument-Modell und den Datenstrukturen der objektorientierten Programmierung, macht die Integration mit Anwendungen einfach. Damit sollten sich die Entwickler weniger Gedanken machen über Kompatibilität und müssen nicht zusätzliche Tools zum Einsatz bringen. MongoDB unterstützt dabei alle im Web häufig verwendeten Programmiersprachen, darunter JavaScript, PHP, Perl, Ruby, Python und Java. All die verfügbaren Client-Drivers stehen auf der [MongoDB-Website](http://www.mongodb.org/display/DOCS/Drivers) zum kostenlosen Download bereit. Eine komplette Liste der Treiber ist in der MongoDB Treiber Seite zu finden[Mong15]. Für unsere Zweck werden wir die API PHP benutzen.

### Kollektion und Dokument

Die logischen Namensräume zur Sammlung mehrerer Dokumente heißen Kollektionen (Sammlungen) und sind grob mit der Tabelle einer relationalen Datenbank vergleichbar. Kollektionen werden vor dem ersten Schreibzugriff automatisch angelegt. Jede Kollektion liegt in genau einer Datenbank, die wiederum ein Container für mehrere Kollektionen ist. Eine Abfrage operiert genau auf einer Kollektion. Ein Zugriff auf Daten mehrerer Kollektionen ist nicht möglich. MongoDB bietet zwei spezielle Formen von Kollektionen: Die System-Kollektion und die Capped-Kollektion.

* **Die System-Kollektion** enthält die Informationen (Metainformationen, aktuelle Konfiguration) des Systems der Datenbank.

<dbname>.system.namespaces = speichert alle Namespaces

<dbname>.system.indexes = speichert Indizes

<dbname>.system.profile = verwaltet Datenbankprofile

<dbname>.system.users = enthält alle Benutzer, die auf die Datenbank zugreifen dürfen

* **Die Capped-Kollektion** bietet die Möglichkeit, den Speicherplatz einer Kollektion zu beschränken. Nach dem Prinzip LRU (Least Recently Used) werden die ältesten Einträge nach einer bestimmten Anzahl von Einträgen oder einer festlegten Zeitspanne überschrieben.

Zum Beispiel wird mit dem folgenden Befehl eine Kollektion mit dem Namen mycoll mit 10.000.000 Bytes auf dem Speicher erzeugt.

db.createCollection(“mycoll“,true, size:10000000)

In MongoDB werden einzelne Datensätze Dokumente genannt. Dokumente entsprechen Zeilen (ROW) in einer Tabelle in RDBMS. Es handelt sich um eine geordnete Menge von Key Value Paaren (die man Felder nennt), mit der Eigenschaft, das Werte auch Arrays und eingebettete Dokumente sein können. Dokumente sind schemalos, d.h. sie haben keinen festen Aufbau, sondern werden dynamisch mit den Daten gefüllt, die benötigt werden. Dokumente werden in BSON, binäres JSON Format, abgelegt. BSON ist ein binäres Format, das im Vergleich zu JSON über mehr Datentypen und eine bessere Traversierbarkeit verfügt[BSON15].

Jedes Dokument muss unbedingt über ein Feld namens \_id verfügen. Es kennzeichnet den Primärschlüssel, der pro Kollektion eindeutig sein muss. Allgemein generieren die jeweiligen Treiber eine so genannte ObjektID. Die ObjektID ist genau 12 Byte lang und setzt sich folgendermaßen zusammen.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Byte | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| Wert | Timestamp | | | | System-ID | | | Prozess\_ID | | Inkrement. Counter | | |

Tabelle ‑ Aufbau einer ObjekID[Mong15]

Die maximale Größe eines Dokuments beträgt 16 MB, so dass sich auch große Geschäftsobjekte darin ablegen lassen. Größere Dokumente lassen sich nur über Umwege des GridFS [Mong14] speichern. Mit GridFS werden Metadaten in einzelne Chunks aufgeteilt, damit beispielsweise eine große Binärdatei ausgelesen werden kann. [Holl12]

## Schema-Design – Blog Modellierung

Schema-Design in MongoDB ist der Prozess der Abbildung eines fachlichen Modells auf Datenbanken und Collections unter Berücksichtigung der wesentlichen Abfragen auf diese Collections[Trel14].

### Analyse

### Modelle

## CRUD (create, reads, update, delete)- Operationen Blog-User Interface

## Session Manager –Benutzer Profile

## Fortgeschrittene Abfragen und Map/Reduce (Blog-Cloud-Tag, Bewertung pro Autor)

## Web Analytik mit MongoDB

## MongoDB Admin

### Replikation

### Ausfallsicherheit

## Bewertung

# Zusammenfassung und Ausblick

# Anhang A1: Übersicht von NoSQL-Datenbanken

Abbildung ‑ Überblick über verschiedenen NoSQL-Datenbanken

## A.2

# Anhang B: Quellcode

# Glossar

**Datei[HDFS]:** Datei-Basierte Sicht auf Daten (in Hadoop typisch nicht kleiner als 100 Mbyte) ist für viele Anwendungen (z.B. Auslieferung von Audio- und Video Dateien) immer noch ein vorzüglicher Ansatz und die bevorzugte Persistenz Schicht für alle folgenden. Es muss nicht immer eine Datenbanken sein.

**In-Memory:** auch und gerade Programme, die Daten in erster Linie im RAM halten und dort manipulieren, derzeit auch als In-Memory-Datenbanken gepriesen, müssen von der Ressourcenüberwachung verwaltet werden. CPU und RAM sind die Hauptressourcen, die YARM im Blick hält. In-Memory-Speicher haben den Vorteil, dass schnelles Processing auch ohne spezielle Vorbereitung der Daten möglich ist, ein Vorteil gegenüber Ad-hoc-Abfrage-Absätzen

**Graphen:**

# Literaturverzeichnis

**[Amaz15]** Amazon <http://aws.amazon.com/de/pricing/> abgerufen am 21.05.2015

**[Brew00]** Eric A. Brewer. „Towards Robust Distributed Systems“.In: Proceedings oft he Ninetenth Annual ACM Syposium on Principles of Distributed Computing. PODC ‚00.New York, NY, USA: ACM 2012

**[Brew04]** Eric A. Brewer **CAP Theorem** <http://www.cs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf> abgerufen am 04.05.2015

**[Brew12]** Eric A. Brewer.„CAP Twelve Years Laters: How the „Rules“ Has changed „ In: Computer 2012

**[Couc12]CouchDB** <http://radar.oreilly.com/2012/01/what-is-big-data.html>, abgerufen am 10.05.2015

[**[**](http://de.wikipedia.org/wiki/Big_Data#cite_note-President_BG14-1)**Dumb12]** Edd DumbillWhat is Big Data Introduction to the Big Data landscape <http://radar.oreilly.com/2012/01/what-is-big-data.html>, abgerufen am 10.05.2015

**[EdHa11]** Stephan Edlich, Jens Hampe, “NoSQL- Einstieg in die Welt Nichtrelationaler Web 2.0 Datenbanken“, Carl Hanser Verlag München, 2011

**[Edli15]** Stephan Edlich, <http://nosql-databases.org/>, abgerufen am 01.04.2015

**[Face15]** Facebook, <https://newsroom.fb.com/company-info/> abgerufen am 04.05.2015

**[GaRe11**]John Ganz, David Reinse. “Extracting Value from Chaos“, [www.http://germany.emc.com/collateral/analyst-reports/idc-extracting-values-from-chaos-ar.pdf](http://www.http://germany.emc.com/collateral/analyst-reports/idc-extracting-values-from-chaos-ar.pdf), 2011, abgerufen am 05.05. 20145

**[Geye15] Jens Geyer**, Graphdatenbanken im Vergleich von Knoten und Kanten. Web & Mobile Developer Alternative Datenbanken 05.2015.

**[gnu]** [**GNU** Betriebssystem](http://www.gnu.org/), <http://www.gnu.org/licences/agpl-3.0.html> , abgerufen am 26.05.2015

**[Gull11] Clemens Gull,** Web-Applikationen entwickeln mit NoSQL 2011 Franzis Verlag

**[Harb10]** [**Ronny Harbich**](http://www-e.uni-magdeburg.de/harbich/ronny_harbich.php)**,** Webcrawling – Die Erschließung des Webs <http://www-e.uni-magdeburg.de/harbich/webcrawling/webcrawling.pdf>abgerufen am 21.05.2015

**[Kirs15]**Christian KirschTraditionelle und Moderne Datenbanktechnik Hase und Igel IXDevelopper Aufgabe 02.02.15

**[Kund09]** Vivek Kundra, Federal CIO, 21. Juli 2009, Open Government and Innovations Conference

**[KuSt15] Thomas Kudraß, Uta Störl** Taschenbuch Datenbanken Carl Hanser Verlag München 2015

**[Lixe15]**Christoph Lixenfel5 häufige Fragen zu Big Data<http://www.computerwoche.de/a/5-haeufige-fragen-zu-big-data,3100789> , abgerufen am 10.05.2015

**[MaCu03]** Ramon A. Mata-Toledo, Pauline K. Cushman, “Relationale Datenbanken“, mitp-Verlag / Bonn, 2003

**[McKe14]**Dan McCreary and Ann Kelly „Making Sense of NoSQ“ a guide for managers and the rest. Manning Publications Co. 2014

**[NeHu15] Peter Neubauer, Michael Hunger** Abfrage Sprachen für Graphendatenbanken (Die Vernetze Welt) IX Developer Auszug vom 02.2015.

**[Selt15]** Margo SeltzerBerkley DB: A Retrospective<http://www.sites.computer.org> **abgerufen am 19.05.2015**

**[Trel14] Tobias Trelle** MongoDB der praktische Einstiegdpunkt.verlag 2014

**[Twit15] Twitter**, <https://about.twitter.com/de/company>, abgerufen am 04.05.2015

*[Watz15] Paul Watzlawick.*

# Eidesstattliche Versicherung

|  |  |  |  |
| --- | --- | --- | --- |
| Name: Okotto |  | Vorname: Pepin Armand |  |
| Matrikel-Nr.:7001520 |  | Studiengang: Medien Informatik |  |

Hiermit versichere ich, .Pepin Armand Okotto., an Eides statt, dass ich die vorliegende Bachelorarbeit mit dem Titel **Einsatz und Möglichkeiten von NoSQL-Datenbanken in Hinblick auf große Datenmenge(Big-Data)** selbständig und ohne fremde Hilfe verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Die Stellen der Arbeit, die dem Wortlaut oder dem Sinne nach anderen Werken entnommen wurden, sind in jedem Fall unter Angabe der Quelle kenntlich gemacht. Die Arbeit ist noch nicht veröffentlicht oder in anderer Form als Prüfungsleistung vorgelegt worden.

Ich habe die Bedeutung die strafrechtlichen Folgen (siehe unten) einer unrichtigen oder unvollständigen eidesstattlichen Versicherung zur Kenntnis genommen.

Auszug aus dem Strafgesetzbuch (StGB)

***§ 156 StGB*** Falsche Versicherung an Eides Statt

Wer von einer zur Abnahme einer Versicherung an Eides Statt zuständigen Behörde eine solche Versicherung falsch abgibt oder unter Berufung auf eine solche Versicherung falsch aussagt, wird mit Freiheitsstrafe bis zu drei Jahren oder mit Geldstrafe bestraft.

Ort, Datum Unterschrift