**Hochschule Emden-Leer   
Fachbereich Technik**

SS 2014

Projektarbeit

**Prof. Dr. Martin Schiemann Lillie**

**Dipl. Ing. Jürgen Meyer**

Thema :

**NoSQL und MongoDB**

Pepin Armand Okotto  
Liebenauerstraße 47  
67549 Worms

Matr.-Nr:7001529

7. Semester

Datum der Abgabe: 21.10.2014

**Ehrenwörtliche Erklärung**

"Ich erkläre hiermit ehrenwörtlich, dass

1. ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe,
2. die aus fremden Quellen direkt oder indirekt übernommenen Gedanken als solche kenntlich gemacht sind,
3. dieses Exemplar mit der beurteilten Arbeit übereinstimmt und
4. diese Arbeit bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht wurde."

Worms, den 20.10.2014

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Unterschrift

Inhaltsverzeichnis

Abkürzungsverzeichnis V

Darstellungsverzeichnisse VI

1 Einleitung 1

2 Grundlagen Datenbanken 3

2.1 Definition und Aufgaben 3

2.2 Anforderungen an Datenbanken 3

2.3 Bestandteile von Datenbanken 4

2.4 Datenbankmodelle 5

2.4.1 Hierarchisches Datenbankmodell 5

2.4.2 Netzwerkartige Datenbanken 7

2.4.3 Relationale Datenbanken 8

2.4.4 Objektorientierte Datenbanken 9

2.5 Das Konsistenzmodell ACID 9

3 NOSQL Datenbanken 11

3.1 Motivation für NoSQL Datenbanken 11

3.1.1 Probleme (objekt-) relationaler Datenbanken 11

3.1.2 Neue Anforderungen an Anwendungen 14

3.2 Definition und Eigenschaften von NoSQL 15

3.3 Grundlegende Techniken und Theorien 16

3.3.1 CAP Theorem / BASE 16

3.3.2 MAP / Reduce 17

3.4 Skalierbarkeit 18

3.4.1 Vertikale Skalierung 19

3.4.2 Horizontale Skalierung 19

3.5 Arten von NoSQL-Datenbanksystemen 20

3.5.1 Graphen-Datenbanken 20

3.5.2 Spaltenorientierte Datenbanken / Wide Column Store 22

3.5.3 Dokumentenorientierte Datenbanken 23

3.5.4 Key Value Systeme 25

3.6 Resümee 26

4 MongoDB - Ein NoSQL Dokumenten-Modell für das Web 28

4.1 Einleitung 28

4.2 Überblick 28

4.3 MongoDB Konzepte 30

4.3.1 Datenbanken 30

4.3.2 Kollektionen 31

4.3.3 Dokumente 32

4.4 Entwerfen des Datenschemas 33

4.5 Installation 36

4.6 MongoDB-Shell 37

4.7 Die MongoDB-API 39

4.8 Replikation und Ausfallsicherheit 39

4.9 Einsatzgebiet 41

4.10 Zusammenfassung MongoDB 42

5 Fazit und Ausblick 44

Anhang 46

Quellenverzeichnis 47

Abkürzungsverzeichnis

|  |  |
| --- | --- |
| ACID | Atomacy, Consistency, Isolation, Durability |
| API | Applikation Programming Interface |
| BASE | Basically Available, Soft state, Eventual consistency |
| BSON | Binary JSON |
| CAP | Consistency, Availability, Partition Tolerance |
| CRUD | Create, Read, Update, Delete |
| DBMS | Database Management System (dt.: Datenbank-Verwaltungssystem) |
| HTML | Hypertext Markup Language |
| JSON | JavaScript Object Notation |
| LRU | Least Recently Used |
| NoSQL | Not only SQL |
| OQL | Object Query Language |
| RDBMS | Relational Database Management System (dt.: Relationales Datenbankmanagementsystem) |
| SQL | Structured Query Language |

Darstellungsverzeichnisse

Abbildungen

Abbildung 1 Hierarchische Datenbanken [Schi14] 6

Abbildung 2 Netzwerkartige Datenbanken [Schi14] 7

Abbildung 3 Parallelisierte Verarbeitung mit Map/Reduce [Holl12] 18

Abbildung 4 Graphen Datenbank Modell [Neo414] 22

Abbildung 5 Spaltenorientierte Datenbanken 23

Abbildung 6 Data-Modell einer dokumentenorientierten Datenbank [Mong14] 24

Abbildung 7 Key Value Datenmodelle 26

Abbildung 8 Visual Guide to NoSQL Systems [Hurs10] 27

Abbildung 9 Mongo DB Anspruch [Mong14] 30

Abbildung 10 Strukturelemente eines Mongo Servers 31

Abbildung 11 Blog Schema Design für eine relationale Datenbank 35

Abbildung 12 Blog Schema Design für MongoDB 36

Abbildung 13 Initialer Zustand des Replica Sets 40

Abbildung 14 Ausfall des Primary-Knotens 40

Abbildung 15 Wahl eines neuen Primary-Knotens 41

Tabellen

Tabelle 1 Hierarchisches Netzwerk [Schi14] 6

Tabelle 2 Vor- und Nachteile netzwerkartiger Datenbanken [Schi14] 8

Tabelle 3 Vor- und Nachteile relationaler Datenbanken [Schi14] 8

Tabelle 4 Vor-und Nachteile objektorientierter Datenbanken [Schi14] 9

Tabelle 5 Vergleich RDBMS und Graphen-Datenbanken 22

Tabelle 6 Vergleich RDBMS mit dokumentenorientierten Modellen 25

Tabelle 7 Aufbau einer ObjektID [Mong14] 33

Tabelle 8 Befehle 38

Tabelle 9 Vergleich CRUD Operation RDBMS und MongoDB [Mong14] 38

# Einleitung

“Man sagt, Daten seien das neue Öl. Wenn das so ist, sind Datenbanken die Ölfelder, die Raffinerien, die Bohrer und die Pumpen.“ [ReWi12]

Daten sind heute sehr wichtige Produktionsfaktoren. Sie werden erzeugt, gesammelt, manipuliert, analysiert und archiviert. 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 nur eine Ebene; das heißt, dass das Datenbankmanagementsystem selbst und die Daten sich am selben Ort befanden [MaCu03]. Das relationale Datenbank-Konzept war die Lösung für alle Datenbank-Probleme. Mit der Entwicklung des Web 2.0 im Jahr 2000 haben sich die Anforderungen an Datenbanken stark verändert. Das Web 2.0 bietet neue Technologien und Möglichkeiten, Daten zu verarbeiten. Diese Daten sind vielfältig und kommen aus verschiedenen Quellen und sind nicht auf einem Rechner konzentriert, sondern vielmehr über viele Rechner ortsunabhängig verteilt. In diesem Zusammenhang wird von Daten in der Cloud gesprochen. In sozialen Netzwerken kursieren Videos, Pinnwandeinträge etc. auf verschiedenen Geräten mit verschiedenen Betriebssystemen. Es gibt kaum etwas, das nicht mit dem Internet verbunden ist – es wird vom “*Internet of all Things“*gesprochen. 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 zählte beispielweise nach eigenen Angaben über 802 Millionen aktive Benutzer im März 2014, 609 Millionen melden sich täglich mit ihren mobilen Geräten an [Face14], welche täglich vier Terabyte an Daten produzieren. Bei Twitter zeigen die Statistiken, dass mehr als 230 Millionen aktive Nutzer im Monat Mai 2014 getwittert haben, pro Tag werden 500 Millionen Tweets versendet [Twit14]. Amazon ist sehr erfolgreich im Bereich des Online-Verkaufs. Das Portal erhält viele Anfragen und muss ebenso viele Buchungen pro Sekunde bewältigen. Amazon, Facebook, Twitter etc. -sozusagen alles Firmen, die Ihre Geschäftsmodelle online abwickeln - haben das gleiche Problem: “traditionelle“ relationale Datenbanken können diese Fülle von Anfragen nicht mehr effizient bewältigen. So entstand eine neue Technologie, bekannt als Not only SQL “NoSQL“. Google mit dem spaltenorientierten Datenbank-Management-System Big Table und Facebook mit Casandra waren die Vorreiter dieser neuen Technologie.

Die vorliegende Projektarbeit untersucht daher die Fragen, wie die NoSQL Technologie funktioniert und wie sie sich von traditionellen relationalen Datenbanken abgrenzt. Ist der Begriff “NoSQL“ nur ein Hype oder werden die relationalen Datenbanken aussterben?

Diese Arbeit ist in zwei wichtige Teile gegliedert. Im ersten Teil wird auf die Herkunft des “NoSQL“ Begriffes eingegangen und dabei erläutert, wieso dieser Begriff eine entscheidende Rolle in der heutigen IT-Welt spielt. Weiterhin werden - neben allgemeinen NoSQL-Eigenschaften, die die NoSQL Datenbanken von relationalen Datenbanken unterscheiden - außerdem die zentralen NoSQL Datenbanktypen vorgestellt. Der zweite Teil schließt mit dem Beispiel von MongoDB ab. In Webanwendungen ist MongoDB eine sehr bekannte und beliebte Dokumenten-orientierte NoSQL Datenbank. MongoDB unterscheidet sich von relationalen Datenbank-Systemen durch die Flexibilität seiner Datenmodelle, die ihm erlaubt, eine einfache Skalierung und eine gute Performanz zu gewährleisten. MongoDB eignet sich sehr gut für Anwendungen wie Blogs. Es wird im letzten Teil dieser Arbeit ein Blog entwickelt werden, die Daten werden in MongoDB persistiert.

# Grundlagen Datenbanken

Dieses Kapitel dient als Einstieg in die Datenbanken. Es wird beschrieben, welche Funktionen zu einer Datenbank gehören und welche Werkzeuge eine Datenbank aufweisen sollte.

## Definition und Aufgaben

* Eine Datenbank ist eine strukturierte Sammlung von Daten, die verwaltet werden, um die Bedürfnisse einer Gruppe von Anwendern zu decken. Die Struktur entsteht durch die Organisation der Daten mit Hilfe eines Datenbankmodells.[Schi14]
* Eine Datenbank ist eine eigenständige und auf Dauer ausgelegte Datenorganisation, welche einen Datenbestand sicher und flexibel verwalten kann. [Schi14]

Eine Datenbank soll Daten verwalten: Eingabe von neuen Daten, Löschen veralteter Daten sowie das Nachführen bestehender Daten. Diese Daten können von verschiedenen Programmen und Anwendungen verarbeitet werden.

Eine Datenbank kann entweder lokal oder in einem verteilten System gespeichert werden. Verteilte Datenbank-Systeme haben den Vorteil, dass sie gleichzeitig von mehreren Benutzern und ortsunabhängig benutzt werden können.

## Anforderungen an Datenbanken

Eine ideale Datenbank sollte folgende charakteristischen Eigenschaften besitzen [Schi14]:

* Integritätssicherung:  
  Daten werden auf Korrektheit (bereits während der Eingabe) überprüft und Fehlmanipulationen verhindert.
* Redundanzarmut:  
  Es gibt keine ungeordnete Mehrfachspeicherung von Datenwerten.
* Datensicherheit:  
  Ungewollter Datenverlust wird durch interne Backup- und Prüfmechanismen verhindert.
* Datenschutz:  
  Zugriffskontrolle und spezifische Sichten sorgen für einen Zugang gemäß der Rechte des Nutzers.
* Mehrbenutzerbetrieb:  
  Viele Benutzer können parallel auf der Datenbank arbeiten.
* Datenunabhängigkeit:  
  Das DBMS ist nicht an die Daten der Datenbank gekoppelt, es kann unabhängig von den Daten weiterentwickelt werden.
* Zentrale Kontrolle:  
  Ein Administrator ist in der Lage, das gesamte System von einem Rechner aus zu verwalten.

## Bestandteile von Datenbanken

Gewisse Werkzeuge und Komponenten müssen bereitgestellt werden, dass die vorher beschriebenen Anforderungen erfüllt werden können.

**Datenbank-Verwaltungssysteme (DBMS)** verwalten Daten auf einem Computer. Sie kümmern sich um die Organisation der Daten und regeln den Zugriff auf die Daten. Es ist zu unterscheiden zwischen Desktop-Datenbank-Systemen wie Microsoft Access und Server-gestützten Datenbank-Systemen.

Die **Datenbanksprache** bildet die Schnittstelle zwischen dem Benutzer und dem Datenbank-Verwaltungssystem. Bei den meisten Datenbanken wird die Sprache SQL (Structured Query Language) verwendet.

## Datenbankmodelle

Unter einem Datenbankmodell versteht man eine abstrahierte Darstellung der Daten und der zwischen diesen Daten bestehenden Beziehungen. Man unterteilt Datenbanken in konzeptionelle und implementative Modelle.

Das konzeptionelle Modell beschäftigt sich mehr mit der Frage, was in die Datenbank aufgenommen werden soll, als damit wie dies zu geschehen hat (z.B. ER-Datenbankmodell).

Implementative Modelle kümmern sich darum, wie die Daten in Datenbanken gespeichert werden und wie die Strukturen untereinander in Beziehung stehen (z.B. Netzwerkmodelle oder das relationale Datenbankmodell).

Datenbanken lassen sich grundsätzlich nach drei Hauptkategorien einteilen:

* Hierarchische Datenbanken
* Relationale und Objektrelationale Datenbanken
* Objektorientierte Datenbanken

### Hierarchisches Datenbankmodell

Dieses Modell gilt als das älteste Datenmodell und wurde im vorigen Jahrhundert von der Firma North American Rockwell entwickelt. In diesem Modell werden Daten in Form einer einzelnen, sequentiellen Datei gespeichert wie die Abbildung 1 darstellt.

Beispiel: Aufbau des Verzeichnisses im Betriebssystem DOS

***C:\Emden\Studium\Projekt\Okotto\NOSQL\_MongoDb\version1.ppt***

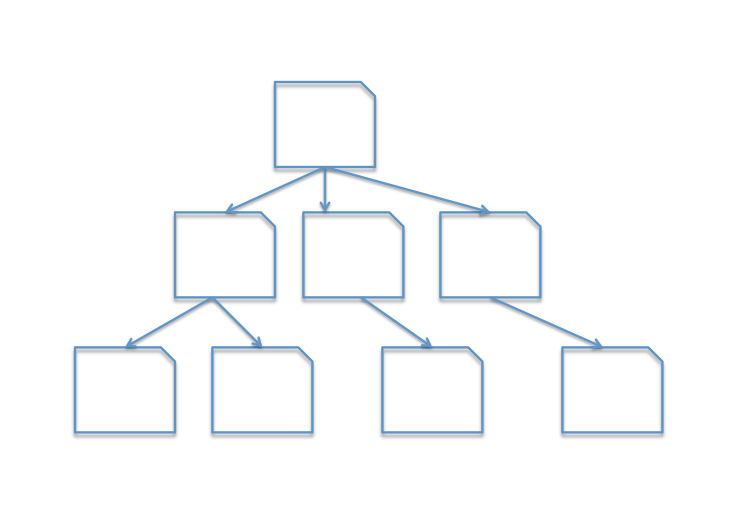


Abbildung Hierarchische Datenbanken [Schi14]

Verknüpfungen zwischen verschiedenen Bäumen oder über mehrere Ebenen innerhalb eines Baumes sind nicht möglich. Dieses Modell hat seine Grenzen gezeigt. 60% der gespeicherten Daten waren redundant. Dieses System wird heute kaum genutzt. Weitere Modelle sind entwickelt worden, wie z.B. die Netzwerk-Datenbanken und relationale Datenbankmodelle. Die Tabelle 1 fasst die Vor- und Nachteile dieses Modells kurz zusammen.

|  |  |
| --- | --- |
| **Vorteile** | **Nachteile** |
| * sehr kurze Zugriffszeiten * minimale Redundanz | * kaum änderbare Strukturen * komplexe Programmierung |

Tabelle Hierarchisches Netzwerk [Schi14]

### Netzwerkartige Datenbanken

Dieses Modell ist eine Erweiterung des hierarchischen Datenmodells und wurde zum ersten Mal auf der “Conference on Data Systems Language“ präsentiert. Dieses Modell besitzt keine strenge Hierarchie. Im Gegensatz zum hierarchischen Modell kann jeder Entitätstyp mehrere Nachfolger und mehrere Vorgänger haben.

Die Modellierung einer Netzwerkdatenbasis ist sehr komplex. Wie beim hierarchischen Datenmodell muss der Anwender die Struktur der gesamten Datenbank kennen, er muss Zugriffs-Algorithmen formulieren können. Seit den 1990er Jahren gewinnt das relationale Datenbankmodell mehr und mehr an Bedeutung.

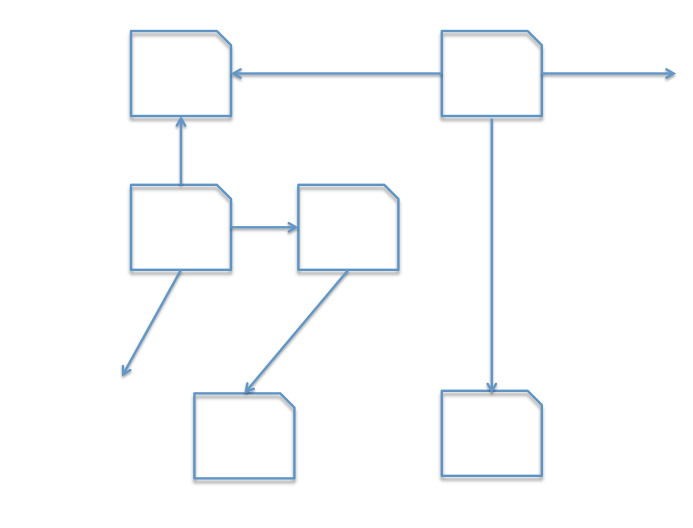


Abbildung Netzwerkartige Datenbanken [Schi14]

|  |  |
| --- | --- |
| **Vorteile** | **Nachteile** |
| * kurze Zugriffszeiten * geringe Redundanz | * änderbare Strukturen * aufwendige Programmierung |

Tabelle Vor- und Nachteile netzwerkartiger Datenbanken [Schi14]

### Relationale Datenbanken

Im Jahr 1970 entwickelte Dr. E. F. Codd, ein bei IBM angestellter Mathematiker, das relationale Datenbankmodell als grundlegendes Konzept, bei dem alles in Tabellen oder Relationen gespeichert wird. Die Verwendung miteinander verknüpfter Tabellen und Sichten erlaubt die Nutzung von verteilten Datenbanksystemen, was schwer in Netzwerken oder im hierarchischen Modell realisierbar war.

Ein sehr wichtiger Bestandteil einer relationalen Datenbank ist das Schema. Das Schema regelt, welche Daten in der Datenbank gespeichert werden und wie diese Daten in Beziehung zueinander stehen. Dieser Vorgang wird als Datenmodellierung bezeichnet.

Es gibt bei den relationalen Datenbanken auch Nachteile. Zugriffe erfordern oft das Lesen und Zusammenfügen von Daten aus mehreren Tabellen, was die Laufzeit verlängert und zu vielen Ein-und Ausgaben führt. Auch können einige Daten nur redundant in Tabellenform abgespeichert werden. Diese Vor- und Nachteile sind in Tabelle 3 zusammengefast.

|  |  |
| --- | --- |
| **Vorteile** | **Nachteile** |
| * leichte Änderbarkeit des Datenbankaufbaus * mathematisch fundiert * leicht programmierbar und zu verwalten | * häufig viele Ein-/Ausgaben notwendig * erfordert eine hohe Rechnerleistung |

Tabelle Vor- und Nachteile relationaler Datenbanken [Schi14]

### Objektorientierte Datenbanken

Ein objektorientiertes DBMS erlaubt es, zusammengehörende Einheiten auch zusammen abzuspeichern. Dabei werden Daten zusammen mit ihren Funktionen in einem Objekt gespeichert. Das Datenmodell richtet sich nach der objektorientierten Programmierung und im Einzelnen nach der jeweiligen Programmiersprache, die zum Einsatz kommt (z.B. C++, Java oder C#). Die Abfrage der Daten kann über die Funktionen des Objektes oder die Objektabfragesprache erfolgen. Hierbei kann auf die SQL-ähnliche Abfragesprache Object Query Language (OQL) zurückgegriffen werden. Die Vor-und Nachteile objektorientierter Datenbanken werden in Tabelle 4 aufgelistet.

|  |  |
| --- | --- |
| **Vorteile** | **Nachteile** |
| * objektorientierter Aufbau * universell einsetzbar * noch relativ einfach zu programmieren und zu verwalten * (teilweise) aufwärtskompatibel zu relationalen Datenbanken | * relativ viele Ein-/Ausgaben notwendig * erfordert eine hohe Rechnerleistung * teilweise recht komplexer Aufbau |

Tabelle Vor-und Nachteile objektorientierter Datenbanken [Schi14]

## Das Konsistenzmodell ACID

Bezüglich der Ausführung von Transaktionen, welche aus einer oder mehreren Operationen der jeweiligen DB-Sprache (z.B. SQL) bestehen, garantiert das DBMS die Einhaltung von vier grundlegenden Eigenschaften: nämlich Atomarität, Konsistenz, Isolation und Dauerhaftigkeit. Man spricht hierbei von den sog. ACID-Eigenschaften, abgeleitet von den Anfangsbuchstaben der englischen Begriffe Atomicity, Consistency, Isolation und Durability Diese im Folgenden näher erläuterten Eigenschaften charakterisieren zugleich das Transaktionskonzept [Rahm14]:

* **Atomarität** | (Atomicity, "Alles oder Nichts")

Transaktion wird entweder ganz oder gar nicht ausgeführt.

* **Consistency** | (Konsistenz/Integritätserhaltung)

Datenbank ist vor Beginn und nach Beendigung einer Transaktion jeweils in einem konsistenten Zustand.

* **Isolation** |(Isolation)

Nutzer, die mit einer Datenbank arbeiten, sollen den Eindruck haben, dass sie mit dieser Datenbank alleine arbeiten.

* **Durability** |(Dauerhaftigkeit/Persistenz)

Nach erfolgreichem Abschluss einer Transaktion muss das Ergebnis dieser Transaktion “dauerhaft“ in der Datenbank gespeichert werden.

# NOSQL Datenbanken

## Motivation für NoSQL Datenbanken

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. Durch dieses stetige Ansteigen der Datenmenge (Big Data) und den wachsenden Bedarf an Leistung und Verfügbarkeit der Dienste/Websites mit einem hohen Datenaufkommen, stehen IT Unternehmen vor großen Herausforderungen und beschäftigen sich zunehmend mit den folgenden Fragen:

* Wie kann das immer größer werdende Datenvolumen gemeistert werden?
* Wie wird ein hoher Grad an Verfügbarkeit erreicht?
* Wie lassen sich die gegebenen Probleme bestmöglich lösen?
* Wie lassen sich Daten bestmöglich partitionieren?
* Wie lassen sich Datenbanken mit neuen Frameworks kombinieren?

### Probleme (objekt-) relationaler Datenbanken

[Kuba10].

**Szenario 1 Eine Unterhaltungs-App**

Ein Entwickler hat für sein Unternehmen eine Unterhaltungs-App entwickelt, die Daten persistieren auf einer relationalen Datenbank. Diese App wird sehr populär 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, um die Verfügbarkeit zu gewährleisten.

Das Team bekommt die 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 sowie Aktivitäten von 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.

Es wird im Allgemeinen als Nachteil von RDBMS ausgeführt, dass einfestes Schema liegen muss.Relationale Datenbanken haben ein festes Schema und sind weniger geeignet für semi-strukturierte Daten [Kuba10]. In beiden Szenarien müssen sich die Entwickler Gedanken machen über die unstrukturierten Daten und die häufigen Änderungen. Relationale Modelle sind weniger flexibel für Entwickler, 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.

### Neue Anforderungen an Anwendungen

Das Web entwickelt sich rasant. Vor noch ein paar Jahren waren Smartphones noch kein Massenprodukt, Internet selbst wurde nur von Fachleuten benutzt, soziale Netzwerke und Geodienste waren noch weitgehend unbekannt und weder HTML5 noch NoSQL geläufige Begriffe. Heute hingegen sind diese Technologien wichtige Bestandteile von Webanwendungen. Benutzer fordern heute übersichtliche, interaktive, schnelle und benutzerfreundliche Websites, die auf ihre aktuellen Bedürfnisse unmittelbar eingehen. Persönlich, einfach, schnell – das sind die wesentlichen Erfolgsfaktoren aus Sicht des Benutzers.

Als Entwickler ist es sehr wichtig, mit dieser rasanten Entwicklung und den veränderten Benutzererwartungen Schritt zu halten.

**Neue Architekturen**

Damals hat man Daten irgendwie als Binär auf der Festplatte gespeichert oder ein paar Gigabit waren schon genug, um Daten zu persistieren. Heute hat sich die Infrastruktur, aus der die Daten gespeichert werden, geändert. Unternehmen nutzen Cloud Computing, Standard-Hardware, Virtualisierung oder Data-Center.

**Neue Entwicklungsmethoden**

Damals war Cobol als Programmiersprache Standard. Mittlerweile sind viele Programmiersprachen entstanden. Die Methoden, die angewendet werden um Programme zu erstellen, ändern sich ständig. Es gibt neue Frameworks, die zusammen arbeiten sollen, um effiziente Ergebnisse zu erzielen.

**Neue Apps**

Die Anwendungen kommen nicht mehr in CD oder DVD Form. Zum Beispiel bietet Google eine ganze Palette von Software, um Dokumente in Teams zu bearbeiten. Big Data, soziale und mobile Anwendungen sind die neue Form.

## Definition und Eigenschaften von NoSQL

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. In ihrem Buch “NoSQL - Einstieg in die Welt Nichtrelationaler Web 2.0 Datenbanken“ geben [EdHa11] folgende Eigenschaften als Definition:

* **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 setzt 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.

## Grundlegende Techniken und Theorien

In diesem Kapitel werden verschiedene theoretische Grundlagen berücksichtigt, die viele der Vorteile von NoSQL Datenbanken erst ermöglichen. Darüber hinaus wird das CAP-Theorem und BASE betrachtet, da sie eine wichtige Basis für verteilte Systeme darstellen.

### CAP Theorem / BASE

Das Akronym CAP steht für Consistency (Konsistenz), Availability (Verfügbarkeit) und Partition Tolerance (Ausfalltoleranz).

Bei relationalen Datenbanksystemen muss die Konsistenz streng gehalten werden. Diese Anforderung wird problematisch, wenn man eine große Datenmenge hat, wo eine horizontale Skalierung nicht mehr reichen kann. Die Gewährleistung der Konsistenz über mehrere verteilte Datenbankinstanzen hinweg ist mit relationalen Systemen in der Praxis mit erheblichem Wartungs- und Entwicklungsaufwand verbunden. Zusätzlich erreicht so die Reaktionszeit des Systems einen Wert, der für Webanwendungen in einem nicht mehr akzeptablen Bereich liegt.

Über dieses Problem hat [Brew00] im Jahr 2000 in seinem Vortrag “Towards Robust Distributed Systems“ Vermutungen formuliert, welche 2002 von Gilbert und Nancy Lynch bewiesen wurden. Dieses Theorem besagt, dass es in einem verteilten System unmöglich ist, gleichzeitig die drei Eigenschaften Konsistenz, Verfügbarkeit und Ausfalltoleranz zu garantieren.

* **Konsistenz (Consistency)** | Im verteilten System mit replizierten Daten muss sichergestellt sein, dass nach Abschluss einer Transaktion auch alle Replikate des manipulierten Datensatzes aktualisiert werden. Alle Knoten sehen zur selben Zeit dieselben Daten. Diese Konsistenz sollte nicht verwechselt werden mit der Konsistenz aus den [ACID](http://de.wikipedia.org/wiki/ACID)-Transaktionen, die nur die innere Konsistenz eines Datenbestandes betrifft.
* **Verfügbarkeit (Availability)** | Alle Anfragen an das System werden stets beantwortet.
* **Ausfalltoleranz (Partition Tolerance)** | Das System arbeitet auch bei Verlust von Nachrichten einzelner Netzknoten oder Partition des Netzes weiter.

### MAP / Reduce

MAP/Reduce ist ein Framework, um Berechnungen großer Datenmengen durchzuführen. Dieses Framework hat seine Wurzeln in der funktionalen Programmierung und wurde unter anderem durch google zur Bearbeitung von großen Datenmengen populär.

Beim MAP/Reduce-Verfahren werden die Daten in zwei Phasen verarbeitet. Dadurch lassen sich Berechnungen parallelisieren und auf mehrere Rechner verteilen (Abbildung 3). 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 Parallelisierte Verarbeitung mit Map/Reduce [Holl12]

## Skalierbarkeit

“A service is said to be scalable if when we increase the resources in a system, it results in increased performance in a manner proportional to resources added“ [Voge06].

Skalierbarkeit meint die Eigenschaft eines Software- oder Datenbanksystems mit den steigenden Anforderungen (z.B. durch eine größere Anzahl der Nutzer, größere Datenmengen) linear zu wachsen. Es gibt zwei verschiedene Arten, Systeme zu skalieren: vertikale Skalierung (intern) sowie horizontale Skalierung (extern).

### Vertikale Skalierung

Unter vertikaler Skalierung (“Scale-up“) wird das Aufrüsten vorhandener Server durch “bessere“ Hardwarekomponenten (CPU, Hauptspeicher, Netz, Platten, etc.) verstanden. Je nach Ziel des Aufrüstens sind mit “besser“ schnellere Komponenten, mehr Kapazität und/oder mehr Qualität für mehr Ausfallsicherheit gemeint. Hier spielt eine Rolle, dass die Kosten für solche besseren Komponenten nicht linear verlaufen und im Hochpreissegment sehr wohl exponentiell steigen können. Beschränkt sich die Anzahl der Server auf einen oder wenige, so ergibt sich ein weiteres Problem aus den Ausfallzeiten des Servers, in denen er nicht verfügbar ist, z.B. aufgrund dieser Aufrüstzeiten oder des Ausfalls von Komponenten.

### Horizontale Skalierung

Unter horizontaler Skalierung (“Scale-out“) wird die Erhöhung der Anzahl der verfügbaren Server, die in einer verteilten Datenbank miteinander vernetzt werden, verstanden. Die Anzahl der Server kann dabei sehr hoch sein (einige tausend Rechner sind kein Sonderfall), wobei der einzelne Rechner kein ausfallsicherer Server sein muss. Die neuen Server müssen nicht zwingend so leistungsfähig sein, wie die bereits vorhandenen Serverknoten, um das Gesamtsystem zu erweitern. Bei horizontalskalierten Systemen liegt der Vorteil in der Verfügbarkeit und Ausfalltoleranz. Der Ausfall eines einzelnen Serverknotens oder einer Verbindung zwischen zwei Serverknoten wird durch die Redundanz des gesamten Systems kompensiert.

## Arten von NoSQL-Datenbanksystemen

Auf der Webseite von [Edli14] zählt man zurzeit ungefähr 150 verschiedene Arten von Datenbanken, die unter dem Schlagwort NoSQL zusammengefasst sind. Diese Datenbanksysteme lassen sich in vier Kategorien unterteilen:

* Key Value Stores,
* Spaltenorientierte Datenbanken,
* Dokumentenorientierte Datenbanken,
* Graphen-Datenbanken.

Es wird im Folgenden jedes System näher erläutert und besonders die Grenze zu traditionellen Datenbanken gezeigt.

### Graphen-Datenbanken

Die Welt ist nicht ein Tabelle sondern ein Graph. Graphen gibt es überall, zum Beispiel entspricht ein Netzwerk einem Graph, wobei jeder Knoten einem Computer, Switch, Router entspricht; jede Kante einer Verbindung. Eine Beziehung zwischen Menschen kann auch als Graph dargestellt werden. Eine Graphen-Datenbank verwendet Graphen, um Informationen abzuspeichern sowie darzustellen. Statt herkömmlicher Datensätze erstellt man hier Knoten, die durch die Beziehungen, die man zwischen ihnen definiert, miteinander verknüpft werden. Sowohl an Kanten als auch an Knoten können Key-Value Paare angehängt werden, die Daten enthalten. Diese 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. Ein Beispiel sind Social Network Webanwendungen. In solchen Anwendungen spielen die Beziehungen zwischen den einzelnen Nutzern eine große Rolle und bilden die Grundlage einiger Funktionen.

Es ist möglich, ein Graphen-Datenbank-Modell in eine relationale Datenbank zu modellieren. Jedoch sollte man mit vielen Joins rechnen, die sich bei steigender Komplexität der Beziehung und des Datenaufkommen nur sehr zeitintensiv ausführen lassen. Die Abbildung 4 stellt ein Graphen-Datenmodell für Schauspieler und deren Filme dar. Dieses Modell kann in ein RDBMS transformiert werden, in dem man 3 Tabellen bildet, nämlich die Tabelle “Person“, “Film“ (Movie) und für die Kante eine Tabelle “Beziehungen“. Mit einer SQL-Abfrage kann man alle Knoten (Person, Movie, Beziehung) oder Kanten (Beziehung) mit vorgegebenen Eigenschaften herausfinden. Um alle indirekten Beziehungen zu finden oder Pfade zwischen zwei Personen 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. Graphen-Modelle bieten eine bessere Performanz als SQL. Graphen-Datenbanken haben keine Standards für Abfragen. Jede Datenbank dieses Modells bietet seine eigene Query.

Die derzeit bekanntesten Vertreter sind:

* Node4j mit der Abfrage Cyber. Dieses System ist transaktionsorientiert mit ACID-Konsistenz.
* ArangoDB – Opensource Dokumenten- und Graphen-Datenbank
* Cayley – Opensource
* DEX/Spaksee – Close Source, geschrieben in C++.

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

Tabelle Vergleich RDBMS und Graphen-Datenbanken

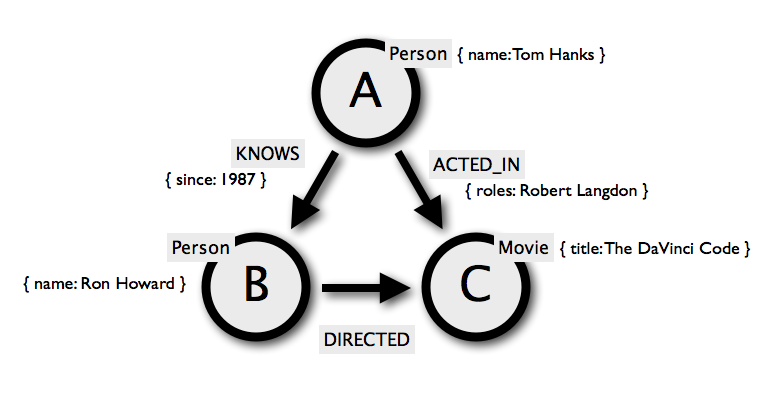


Abbildung Graphen Datenbank Modell [Neo414]

### Spaltenorientierte Datenbanken / Wide Column Store

Bereits seit den 80er Jahren gibt es schon die Idee der spaltenorientierten Datenbanken, die lange Zeit von Synbase eingesetzt worden sind. Die Idee dabei ist, mehrere Einträge in Spalten zu speichern. Die Struktur einer Spalte besteht aus dem Namen der Spalte, den Daten und einem Zeitstempel. Leseprozesse funktionieren in so organisierten Systemen schnell, da keine unnötigen Daten wie bei zeilenorientierten Datenbanken gelesen werden müssen. Schreibprozesse sind aber nur dann ebenso schnell, wenn nur eine Spalte betroffen ist. Gut geeignete Einsatzbereiche für solche Datenbanken sind daher [Reporting](http://www.computerwoche.de/schwerpunkt/r/Reporting.html), [Business Intelligence](http://www.computerwoche.de/schwerpunkt/b/Business-Intelligence.html) (BI) und [Data-Mining](http://www.computerwoche.de/schwerpunkt/d/Data-Mining.html) [Pürn13].

Einige der erfolgreicheren Implementierungen sind die Open Source Produkte Hbase, Cassandra und Hypertable. Aber auch kommerzielle Anbieter veröffentlichen nach und nach eigene Produkte. So hat zum Beispiel Amazon das eigene Cloudportfolio um das Produkt “Amazon SimpleDB“ erweitert, welches als Webservice zur Verfügung steht.

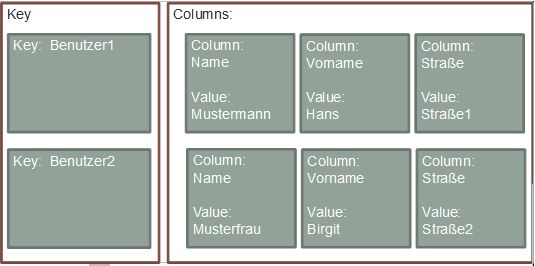


Abbildung Spaltenorientierte Datenbanken

### Dokumentenorientierte 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 dokumentenorientierte Datenbanken dagegen mit Kollektionen, die Dokumente enthalten. 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 MongoDB, CouchDB, Apache Cassandra. MongoDB und CouchDB stehen stark miteinander in Konkurrenz. Anhand von MongoDB wird im nachfolgenden Kaüitel ein Einblick in die Funktionalitäten eines dokumentenorientierten Datenbanksystems gegeben.

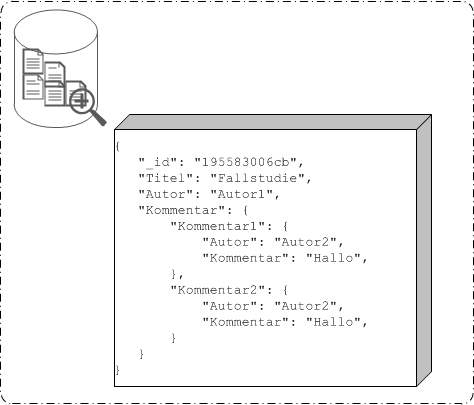


Abbildung Data-Modell einer dokumentenorientierten Datenbank [Mong14]

Die folgende Tabelle zeigt einen Vergleich zwischen den klassischen relationalen Modellen und den dokumentorientierten Modellen.

|  |  |
| --- | --- |
| **Relationale Modelle** | **Dokumentenorientierte Modelle** |
| Tabelle | Kollektion |
| Reihe | Dokument |
| Spalten | Schlüssel/Wert-Paare |
| Joins | Embedded |

Tabelle Vergleich RDBMS mit dokumentenorientierten Modellen

### Key Value Systeme

Key Value Systeme zählen zu den einfachsten NoSQL Datenbankmodellen und gehören zu den ältesten NoSQL Datenbanken. Grundsätzlich ist dieses System eine sehr einfache Speicherform, wie man es bereits von Konfigurationsdateien kennt. Bei der Speicherung ist jedes Objekt einem Schlüssel zugeordnet, wie Abbildung 7 zeigt.

Abfragen oder Operationen werden nur über die Schlüssel gemacht. Die Datenstruktur ist dem Entwickler frei überlassen (XML, JSON, etc.).

Diese Modelle verfügen nur über die vier Abfrage-Operationen “**C**reate“, “**R**ead“, “**U**pdate“ und “**D**elete“ (CRUD).

* **Create:** legt ein neues Objekt an mit seinem Schlüssel create(key, value).
* **Read**: liest ein Objekt mit seinem Schlüssel als Reference read(key).
* **Update**: aktualisiert ein Objekt über Schlüssel update(key).
* **Delete**: löscht ein Objekt über Schlüssel delete(key).

Diese einfache Art der Datenspeicherung ist in der Regel sehr performant und gut skalierbar, allerdings ist ein solches System nicht gut geeignet, um komplexe Abfragen zu realisieren, da die Datenbank meist nichts über die Struktur der Objekte weiß. Bekannte Key Value Datenbanken sind Membase, Redis, Voldermort und Amazon Dynamo.

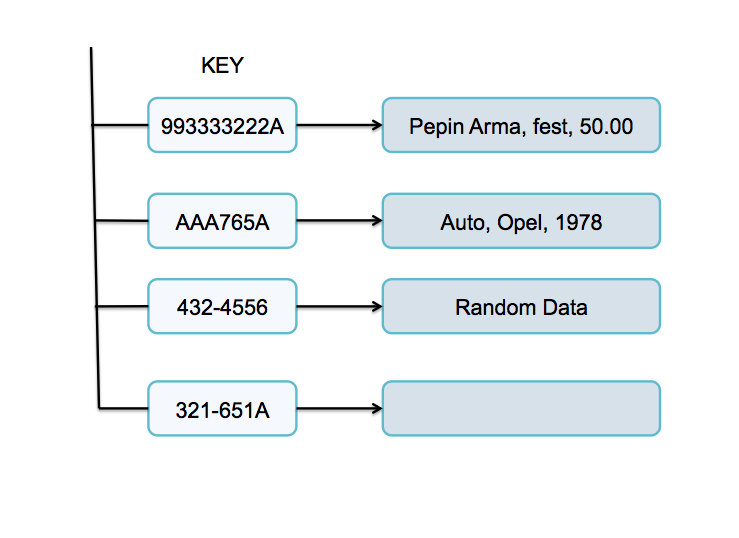


Abbildung Key Value Datenmodelle

## Resümee

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 8 hat [Hurs10]eine Entscheidungspyramide zu NoSQL aufgestellt, welche das CAP-Theorem visuell erklärt. Die meisten relationalen Datenbanken erfüllen die Konsistenz und die Verfügbarkeit (CA) und werden bei der Ausfallsicherheit 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. Eine andere Kritik an dieser Datenbank ist, dass sie nicht standardisiert ist.

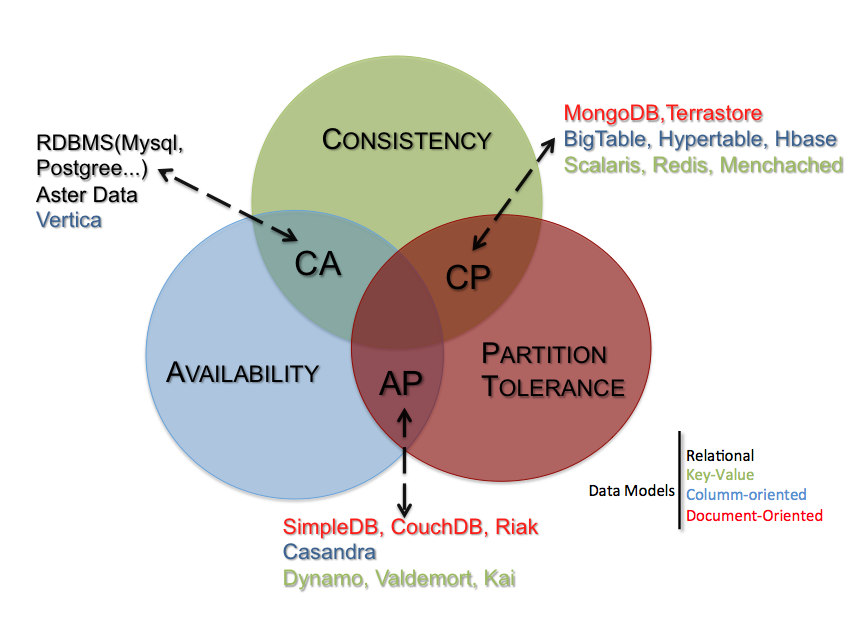


Abbildung 8 Visual Guide to NoSQL Systems [Hurs10]

# MongoDB - Ein NoSQL Dokumenten-Modell für das Web

## Einleitung

Wenn man bei Google 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]. Dieser Teil der Arbeit wird sich mit vielen Aspekten von MongoDB beschäftigen. Es wird gezeigt, warum diese Datenbank so beliebt ist - trotz des Verzichts auf die vielen Features traditioneller relationaler Datenbanken. Wie MongoDB seine Daten modelliert und mit CRUD umgeht, wird anhand eines Blogs (App) veranschaulicht.

## Überblick

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 9 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 oder auf dem privaten Rechner. [Mong14] bietet auf seiner Webseite ein Tutorial, wie man MongoDB in kurzer Zeit zum Einsatz bringen kann.

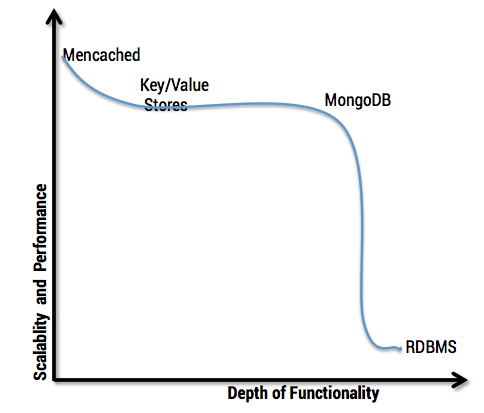


Abbildung Mongo DB Anspruch [Mong14]

## MongoDB Konzepte

### 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.

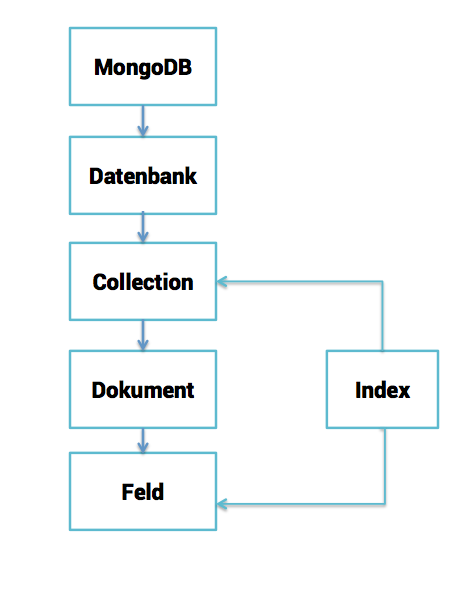


Abbildung Strukturelemente eines Mongo Servers

### Kollektionen

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)

### Dokumente

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. [BSON14]

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 ObjektID [Mong14]

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]

## Entwerfen des Datenschemas

Das Datenschema einer dokumentorientierte DB wie MongoDB folgt anderen Prinzipien als ein Datenschema einer relationalen Datenbank. Während Tabellen und Kollektionen sich einigermaßen entsprechen, verzichtet MongoDB auf Features wie Joins. Bei relationalen Datenbanken ist man gewöhnt, alles zu normalisieren und möglichst viele Informationen auszulagern, um Redundanzen zu vermeiden oder effizient mit dem Speicher umzugehen. Danach sind viele Joins notwendig, um die entsprechenden Daten zu adressieren. Diese Joins Operationen können die Abfragegeschwindigkeit nach unten ziehen. Um dieses Problem zu umgehen, werden Dokumente verschaltet oder referenziert. Es gibt zwei Formen des Referenzierens:

* **Embedded-Document:** Dies sind Teildokumente eines Top-Level-Dokumentes. Es ist eine einfache Methode, ohne Joins zu referenzieren. Dadurch werden die Anfragen performanter. Ferner wird Speicherplatz, der für Referenzen benötigt wird, gespart. Die Gefahr besteht bei unsinnigen oder zu komplexen Verschachtelungen, die Übersichtlichkeit zu verlieren. Außerdem muss die logische Dokumenten-Größenbeschränkung beachtet werden.
* **DBRef(erence):** DBRef ist ein Datentyp, der in etwa dem Fremdschlüssel in RDBMS entspricht. DBRef verweist auf ein Dokument ([ObjectI](http://wikis.gm.fh-koeln.de/wiki_db/Datenbanken/MongoDB#ObjectId)D) in der eigenen oder einer anderen Kollektion (Name) und erwartet zwei Parameter, den Namen der Kollektion und die [ObjectI](http://wikis.gm.fh-koeln.de/wiki_db/Datenbanken/MongoDB#ObjectId)D des Dokuments, auf die sich die Referenz bezieht. Referenzieren funktioniert ausschließlich mit der [ObjectI](http://wikis.gm.fh-koeln.de/wiki_db/Datenbanken/MongoDB#ObjectId)D.

Obwohl MongoDB eine schemafreie Datenbank ist, unterliegt das Design eines Datenschemas auch wichtigen Regeln. Das Design wird vor allem vom Verhältnis der Lese- und Schreibzugriffe geleitet. Wenn die Anwendung oder die Software so konzipiert ist, dass die Lesezugriffe ein Vielfaches der Schreibzugriffe beträgt, so sollte das Schema für die Lesezugriffe optimiert werden, was einer Demoralisation der Daten entspricht. Falls Lese-und Schreibzugriffe gleich sind, sollte man sich Gedanken über ein normalisiertes Datenschema machen.

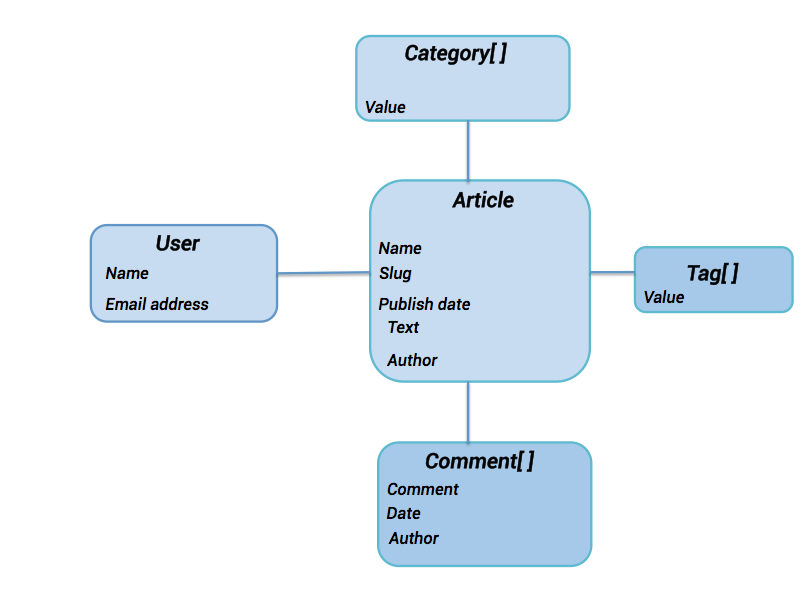


Abbildung Blog Schema Design für eine relationale Datenbank

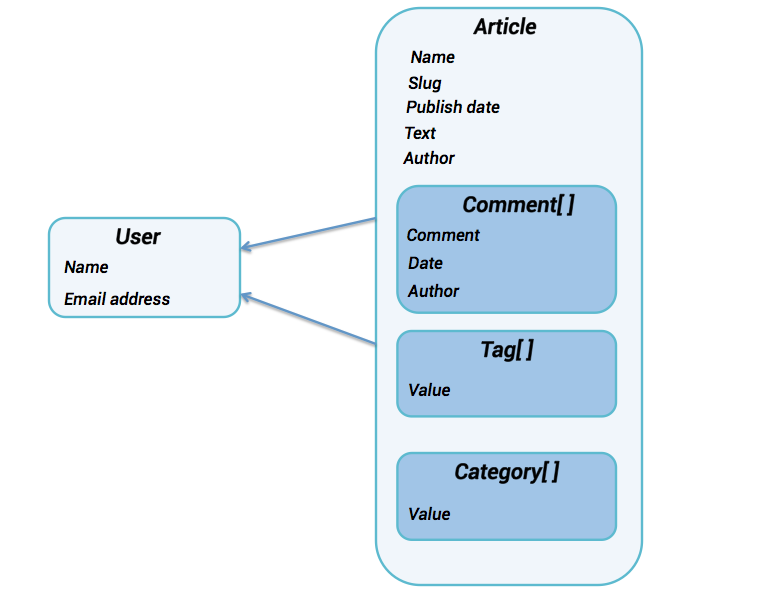


Abbildung Blog Schema Design für MongoDB

## Installation

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.

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[] 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-Shell

In dem Verzeichnis Mongo /bin befindet sich nach der Installation von MongoDB, eine Datei mongo. mongo ist die interaktive Shell, mit der man direkt in MongoDB CRUD Operationen machen, als auch administrative Aufgaben erledigen kann. Die Standardfunktionen von JavaScript können auch hier erledigt werden, da die Shell eine vollständige Java-Script Umgebung ist. In Tabelle 8 werden wichtige Admin-Befehle aufgelistet. Die Tabelle 9 stellt einige MongoDB CRUD Befehle vor und stellt sie ihrem SQL-Pendant gegenüber.

|  |  |
| --- | --- |
| **Befehle** | **Funktion** |
| show dbs | zeigt den Name der vorhandenen Datenbanken |
| show collections | zeigt die Sammlung in den aktuellen Datenbanken |
| show users | zeigt den aktuellen Benutzer der Datenbanken |
| use <db name> | stellt die aktuellen Datenbanken zu <db name> |

Tabelle Befehle

|  |  |
| --- | --- |
| **CREATE TABLE USERS (x Number, y Number)** | **Implizit** |
| INSERT INTO USERS values(1,1) | db.users.insert({a:1,b:1}) |
| SELECT x,y FROM users | db.users.find({}, {x:1, y:1}) |
| SELECT \*FROM users where age = 18 ORDER by name | db.users.find({age:33}).sort({name:1}) |
| UPDATE users set x=1 where y=’l’ | db.users.update({b:’l’},{$set:{a:1},false,true}) |
| CREATE INDEX myindexname ON users(name) | Db.users.ensureIndex({name:1}) |
| Select DISTINCT last\_name FROM users | Db.users.distinct(’last\_name’) |

Tabelle Vergleich CRUD Operation RDBMS und MongoDB [Mong14]

## Die MongoDB-API

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. [Mong14]

## Replikation und Ausfallsicherheit

Wie auf Abbildung 13 zu sehen ist, wird die Ausfallsicherheit über den Einsatz von Replica Sets sichergestellt. In einem Replica Set gibt es keinen Master, sondern eine Primärdatenbank, die als Master gilt, und eine beliebige Anzahl von Sekundärdatenbanken, welche die Slaves darstellen. Der Primary ist der einzige Knoten, der Schreibzugriffe verarbeitet und an die Secondaries repliziert. Lesezugriffe können wahlweise an beiden Arten von Knoten dargestellt werden. Der Primärdatenbank wird von Cluster ausgewählt. Fällt sie aus, wird bei allen verfügbaren Sekundärdatenbanken geprüft, welche als Letztes mit der Primärdatenbank synchronisiert wurde, die so Ermittelte wird automatisch zur neuen Primärdatenbank. (Abbildung 14)

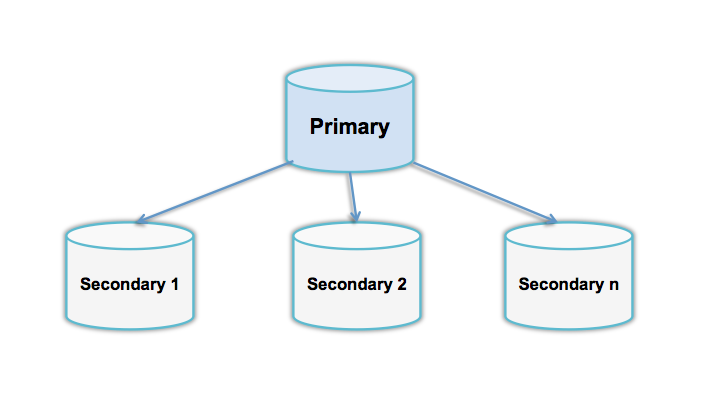
Ein Master wird mit dem Befehl **mongod -master** gestartet, der zugehörigen Slave mit **mongod -slave -sourcemaster\_adresse.** Automatisch werden die Daten des Masters auf den laufenden Slaves repliziert. Im Prinzip ist die Anzahl der Slaves unbegrenzt, kann sich jedoch bei einer größeren Anzahl auf die Performance des Master-Knotens negativ auswirken.

Abbildung Initialer Zustand des Replica Sets

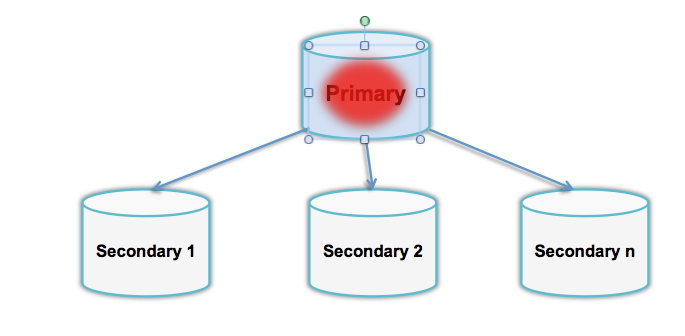


Abbildung Ausfall des Primary-Knotens

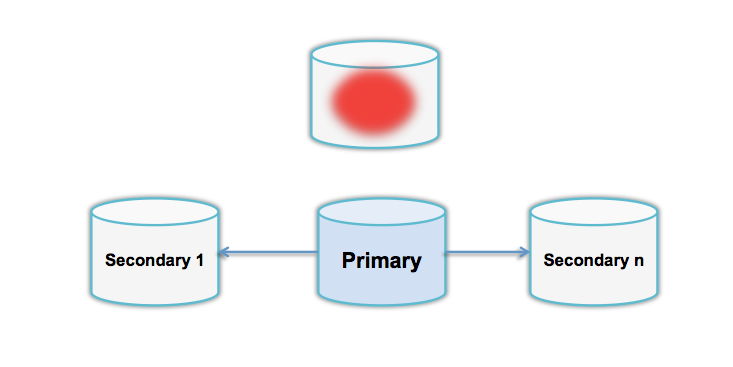


Abbildung Wahl eines neuen Primary-Knotens

## Einsatzgebiet

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].

## Zusammenfassung MongoDB

In der NOSQL-Welt gehört die Datenbank MongoDB zu Recht zu den Stars. In sehr kurzer Zeit lässt sich ein neuer Server in ein bestehendes Cluster einbinden und nutzen. MongoDB bringt auch sehr viel Funktionalität, die man auch bei relationalen Datenbanksystemen finden kann. Die mächtige Query, Indexe auf Dokumentinhalte, Map/Reduce oder Features wie Replikation und einfaches Scharding machen die Benutzung von MongoDB sehr attraktiv. Diese Datenbank kann vor allem für neue noch nicht ausgereifte oder wachsende und dadurch sich oft ändernde Datenstrukturen (Schema) gut genutzt werden, da Änderungen an der Datenstruktur/Schema kein Problem darstellen.

Es gibt auch Nachteile bei MongoDB. Um eine hohe Geschwindigkeit zu erreichen, wird die gesamte Datenbank im RAM gehalten. Kommt es zum unsauberen Beenden einer Master-Datenbank, sind die Daten oftmals korrupt. Deswegen ist es sehr wichtig, MongoDB mit Replikation laufen zu lassen. Es ist auch sehr empfehlenswert, MongoDB auf einem 64-Bit-System installieren zu lassen, um die RAM-Beschränkung von einem 32-Bit-System zu vermeiden.

# Fazit und Ausblick

NoSQL Datenbanken werden sehr gepriesen, nämlich in der Webanwendungen und Entwicklung. Fast jeden Monat wird eine neue Technologie in diesem Bereich entwickelt. NoSQL Datenbanken bieten viele Möglichkeiten und versprechen noch mehr in der Zukunft, was ihre Weiterentwicklung betrifft.

Es ist ratsam dafür Kriterien zu finden, die für NoSQL Datenbanken sprechen. Dafür müssen zwei große Aspekte betrachtet werden, zum einen der funktionale Aspekt und zum anderen der Aspekt der Big-Data Problematik. NoSQL löst einige dieser Probleme und kann auch dazu führen, dass die Produktivität der Entwickler steigt. NoSQL bringt auch mehr Agilität in den Prozess hinein, insbesondere der Verzicht auf Schemata auf der Datenbank führt dazu, dass deutlich schneller Änderungen an bestehenden Systemen vorgenommen werden können. Dazu kommt Funktionalität, die ohne den Einsatz von NoSQL Datenbanken fast gar nicht möglich gewesen wäre. Zum Beispiel bringen Graphen-Datenbanken Funktionalität für Routen-Datenplanung, die der Entwickler sonst mühsam hätte programmieren müssen. Es gibt auch den Vorteil bei NoSQL Datenbanken, große Mengen an Data, sogenannte Big Data, auf verschiedenen Rechnern zu verteilen oder in ein Partition-System laufen zu lassen. Es stellt sich auch weiterhin die Frage, ob man bestehende SQL (relationale Datenbanksysteme) verlassen sollte, um komplett auf NoSQL einzusteigen?

Beide Systeme stehen nicht in direkter Konkurrenz zueinander, sondern können sich gegenseitig ergänzen. Die Tendenz in vielen Unternehmen spricht für polyglotte Systeme; Systeme, die alles bestens kombinieren lassen. NoSQL Datenbankensysteme und etablierte SQL Datenbanken sollen gemeinsam einen Datenpool bereitstellen, um besser mit Daten umzugehen. Ein gutes Beispiel ist ein großes Shop-System wie Amazon. Für die Speicherung von Daten mit sehr starker ACID Konformität wie Finanzen und Reporting kann es sich lohnen, ein RDBMS wie MySQL, Oracle etc. einzusetzen. Eine dokumentenorientierte Datenbank wie MongoDB wäre für das Content-Management-System und Produktkataloge gut einsetzbar. Redis kann zum Einsatz kommen, wenn es darum geht, Informations-Sessions für die Webanwendung bereit zu stellen. Mit einem Graphen-Datenbanksystem wie Neo4J ist es möglich, Empfehlungen von Produktkatalogen bereit zu stellen. So kann jede Datenbank optimal in dem gesamten System positioniert werden.

Obwohl NoSQL Datenbanken im Vergleich zu relationalen Datenbanken nicht standardisiert sind, besteht viel Potential für Fortschritte und neue Ideen. Viele bekannte Unternehmen, wie z.B. Oracle, versuchen, ihr SQL System an die neuen Anforderungen anzupassen, um Defizite auszugleichen.

Anhang

|  |  |
| --- | --- |
| Anhang A | MongoDB Installationsanleitung |
|  |  |

Quellenverzeichnis

Buchquellen

**[Ande10]** Lehnardt Slater Anderson, ”CouchDB - The Definitive Guide“, O’Reilly Verlag, 2010

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

**[Holl12**] Arno Hollosi, “Von Geodaten bis NoSQL. Leistungsstarke PHP-Anwendungen“, Carl Hanser Verlag, München 2012

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

**[Rahm14]** Erhard Rahm, “Mehrrechner-Datenbanksysteme“, Addison-Wesley-Verlag, 2014

**[ReWi12]** [Eric Redmond, Jim R. Wilson](http://www.oreilly.de/catalog/sevendatabasesger/author.html), “Sieben Wochen, sieben Datenbanken“, O’Reilly Verlag, 2012

**[Schi14]** Edwin Schicker, “Datenbanken und SQL eine praxisorientierte Einführung mit Anwendungen in Oracle, SQL Server und MySQL“, Springer Fachmedien Wiesbaden 2014

Online Publikationen

**[Alve11]** Markus Alvermann, “Einführung in MongoDB“, 2011, <http://www.sigs-datacom.de/fileadmin/user_upload/zeitschriften/js/2011/01/alvermann_JS_01_11.pdf>, abgerufen am 15.09.2014

**[Brew00]** E.A. Brewer, “Towards Robust Distributed Systems“, [www.cs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf](http://www.cs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf), 2000, abgerufen am 18.10.2014

Weblinks

**[affe14]** affero, <http://www.affero.org/oagf.html>, abgerufen am 25.06.14

**[BSON14]** BSON, <http://bsonspec.org/>, abgerufen am 19.09.2014

**[Edli14]** Stephan Edlich, <http://nosql-databases.org/>, abgerufen am 19.10.2014

**[Face14]** Facebook, <https://newsroom.fb.com/company-info/> abgerufen am 28.05.2014

**[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 06.09. 2014

**[heis14]** heise, <http://www.heise.de/newsticker/meldung/Big-Data-Rohstoff-der-Informationsgesellschaft-1659622.html>, abgerufen am 03.05.14

**[Horo11]** Eliot Horowitz, “MongoDB Architecture guide”, <http://www.mongodb.org/about/introduction/>, abgerufen am 25.06.14

**[Hurs10]** Nathan Hurst, “Thoughts on Software, Technology and Startup“ [www.blog.nahurst.com/visual-guide-to-nosql-systems](http://www.blog.nahurst.com/visual-guide-to-nosql-systems), 2010, abgerufen am 18.10.14

**[Jans10]** Rudolf Jansen, “CouchDB-angesagter Vertreter der NoSQL Datenbanken“, <http://heise.de/-929070>, 2010, abgerufen am 17.08.2014

**[Json14]** JSON, <http://json.org/>, abgerufen am 17.06.2014

[JSON14] JSON, <http://json.org/>, abgerufen am 19.09.2014

**[Kuba10]** W.Mark Kubacki, “SQL vs NoSQL“, <http://www.se.uni-hannover.de/priv/lehre_2010sommer_wwwseminar/11-SQL_vs_NoSQL-Ausarbeitung.pdf> abgerufen am 17.08.2014

**[Mong14]** MongoDB, <http://docs.mongodb.org/manual/tutorial/getting-started/>, abgerufen am 25.06.14

**[Mong14]** MongoDB, [www.mongodb.org/display/DOCS/GriFS](http://www.mongodb.org/display/DOCS/GriFS), abgerufen am 17.06.2014

**[neo414]** neo4j**,** <http://www.neo4j.org/>**,** abgerufen am 20.10.2014

**[Pürn13]** Heinz Axel Pürner, “NoSQL – die neue(alte) Datenbank-Generation“, 2013, <http://www.computerwoche.de/a/nosql-die-neue-alte-datenbank-generation,2497315>, abgerufen am 10.09.2014

**[Twit14]** Twitter, <https://about.twitter.com/de/company>, abgerufen am 28.05.2014

**[Voge06]** Werner Vogels, “A word on Scalability”, [www.althingsdistributed.com/2006/03/a-word\_on\_scalability.html](http://www.althingsdistributed.com/2006/03/a-word_on_scalability.html), 30.03.2006, abgerufen am 28.10.14