Document Stores
CouchDB

Sergej Sintschilin
2133716

Inhaltverzeichnis

1 Document Stores

Dokumentorientierte Datenbanken (oder Document-Stores) sind eine der wichtigsten Kategorien der sogenannten NoSQL-Datenbanken. Wie von der NoSQL-Bewegung definiert, sind die Document Stores nicht-relational, was vor allem bedeutet, dass sie kein festes und vordefiniertes Schema verlangen, verteilt und horizontal-skalierbar sind. Das zentrale Konzept der dokumentorientierten Datenbank ist die Vorstellung von einem Dokument. Obwohl die Definition eines Dokuments von jeder dokumentorientierten Datenbank in Einzelheiten unterschiedlich formuliert wird, sind die Dokumente im Allgemeinen in sich abgeschlossene Daten, d.h. die Informationen über ein Dokument sind im Dokument selbst gekapselt. Die Daten sind meistens als Standard-Dateiformate (wie XML, YAML, JSON und BSON) oder auch als Binary-Large-Objects kodiert. [1,2]

1.1 Entwicklung

Das Konzept der dokumentorientierten Datenbank ist nicht neu. Die Wegbereiter dafür waren die in den 1970er Jahren für Gesundheits-und Finanz-Anwendungen eingesetzte Programmiersprache MUMPS [3] und das von IBM im Jahr 1989 entwickelte erste dokumentorientierte Datenbanksystem Lotus-Notes [4].

Die Sprache MUMPS ist mit den umfangreichen Tools für die Unterstützung von Datenspeicherung ausgestattet. MUMPS erzwingt keine Verwendung eines vordefinierten Datenmodells und die sogenannten Einträge können direkt, ohne die Notwendigkeit der förmlich definierten Daten, in die MUMPS-Datenbank geschrieben werden. Dieses Datenspeicherungsmodell gibt der MUMPS-Datenbank die Merkmale einer dokumentorientierten Datenbank. [3]

Neben CouchDB, MongoDB und Lotus Notes gibt es viele weitere Document-Stores, wie Amazon SimpleDB, Clusterpoint, OrientDB, RavenDB, Terrastore und ThruDB. Die Unterschiede zwischen den einzelnen Document-Stores bestehen hauptsächlich in Abfrage-Methoden und Zugriffschnittstellen.

1.2 Unterschiede zum RDBMS

Wenn man über eine lange Zeit mit den relationalen Datenbanken gearbeitet hat, ist das Konzept der dokumentorientierten Datenbanken zunächst schwer zu begreifen. Mit dem im Jahr 2003 eingeführten neuen Begriff Web 2.0 werden neue Anforderungen an die Datenbanken gestellt, wodurch die relationalen Datenbanken an ihre Grenzen stoßen [5].

1.2.1 Schemafreiheit

Eine dokumentorientierte Datenbank besteht aus einer Reihe von in sich abgeschlossenen Dokumenten. Dies bedeutet, dass alle Daten eines Dokuments im Dokument selbst gespeichert sind und nicht in der zugehörigen Tabelle, wie es bei einer relationalen Datenbank der Fall ist. Es gibt in der Tat überhaupt keine Tabellen, Zeilen, Spalten oder Beziehungen in einer dokumentorientierten Datenbank. In gewisser Weise können die Dokumente mit den Datensätzen verglichen werden. Im Gegensatz zu einer relationalen Datenbank, in der jede Zeile einer Tabelle die gleiche Menge von Feldern hat und jedes Feld einem Attribut mit einem festgelegten Datentyp und Definitionsbereich zugeordnet ist, sind die Dokumente weniger starr. Schemaänderungen, z.B. mittels `ALTER-TABLE`-Befehlen, setzen die Anwendungen oft längerfristig außer Betrieb. Solche Ausfälle sind für Internet-Anwendungen mit Nutzern in aller Welt nicht tragbar, da sich kein Zeitfenster mehr für solche Wartungsarbeiten findet. Die Schemafreiheit gehört sicherlich zu den interessantesten Aspekten einer dokumentorientierten Datenbank. So muss man sich nicht um Relationen zwischen verschiedenen Dokumenten kümmern, wenn man in einem Feld eine Liste speichern möchte. Außerdem kann man ohne Schwierigkeiten bei einem Dokument neue Felder nachträglich hinzufügen, die bisher nicht existiert haben, oder andere weglassen. Die Schemaverantwortung wird den Anwendungen und nicht dem Datenbankmanagementsystem überlassen. [6,7]

1.2.2 Datenzugriff

Ein weiterer Unterschied zwischen diesen zwei Arten von Datenbanken ist die Speicherung von den eindeutigen Bezeichnern. Die relationalen Datenbanken benutzen üblicherweise das Konzept der Primärschlüssel, die durch eine Auto-Increment-Funktion (wie z.B. bei MySQL) oder durch einen Sequenz-Generator (wie z.B. bei Firebird) erzeugt werden. Natürlich sind diese Bezeichner nur innerhalb der Tabelle bzw. innerhalb der Datenbank, in denen sie verwendet werden, eindeutig. Wenn eine Einfüge-Operation zur gleichen Zeit auf zwei Datenbanken auf getrennten Netzwerken ausgeführt wird, können sie sich nicht gegenseitig abrufen, um den nächsten eindeutigen Bezeichner festzulegen. Eine Auto-Increment-Funktion oder eine Sequenz-Generator reicht bei einem Document-Store nicht aus. Stattdessen wird jedes Dokument innerhalb der Datenbank über einen eindeutigen Schlüssel, der dieses Dokument repräsentiert, angesprochen. Oft ist dieser Schlüssel eine einfache Zeichenkette. In einigen Fällen ist diese Zeichenkette eine URL oder ein Pfad. Die meisten Document-Stores bieten ein spezielles Verfahren, um den Dokumentenschlüssel zu generieren, so dass es für eine andere Datenbank unwahrscheinlich ist den gleichen eindeutigen Bezeichner aus Versehen zu wählen. In der Regel stellt ein dokumentorientiertes Datenbankmanagementsystem zusätzlich eine Abfragesprache oder eine üblicherweise auf einem Map/Reduce-Algorithmus basierende Schnittstelle, die es erlaubt, die Dokumente auf der Basis ihres Inhalts abzurufen. [6]

1.2.3 Horizontale Skalierbarkeit

Die vertikale Skalierung der Datenbanken reicht nicht mehr aus, denn es ergeben sich Performance-Probleme bei Anfragen auf die im Terabyte- und Petabyte-Bereich befindenden Datenmengen. Das Problem der relationalen Datenbanken ist, dass sie eigentlich für „Ein-Server“-Hardware konzipiert wurden und deswegen nicht besonders gut horizontal skalierbar sind. Sie hängen von der Vorstellung von JOINs viel zu stark ab, um die Daten ohne komplizierte Verfahren aufzuteilen. Wenn die SQL-Performanz sich verschlechtert, wird das Problem nicht dadurch gelöst, wenn man einfach mehr SQL-Server einrichtet. In der Regel sind ziemlich leistungsstarke Computer, die mit den großen Datenmengen umgehen können, notwendig, was der Kauf von sehr teurer Hardware nach sich zieht. In relationalen Systemen werden gerade ändernde Datensätze gesperrt für andere lesende und manipulierende Zugriffe. Bei Anfragen bedeuten Sperren immer Wartezeiten für die Anwender. In verteilten Systemen und replizierten Daten kommt bei Datenänderungen erschwerend hinzu, dass auf verschiedenen Rechnern eine Konsistenz gefunden werden muss, damit die manipulierten Werte auf die Replikate verteilt werden können. Dabei müssen Nachrichten zum Aufbau der Sperre verschickt werden, Nachrichten für den geänderten Wert und dann auch wieder Nachrichten mit der Bestätigung, ob die Transaktion mit einem Commit oder einem Rollback endete. Solche Giganten, wie Wikipedia oder Facebook, die auf der auf vielen Servern verteilten relationalen Datenbank aufgebaut sind, lösen diese Probleme, indem sie die wesentlichen Funktionen einer relationalen Datenbank nicht nutzen und auf die JOINs nahezu verzichten [8,9]. Die dokumentorientierten Datenbanken sind auf die horizontale Skalierbarkeit ausgelegt. Da sie keine JOINs unterstützen, was die unmittelbare Folge davon ist, dass es keine Primär- und Fremdschlüsseln in Document-Stores gibt, lassen sich die Dokumente ohne große Probleme auf mehreren Servern trennen, das heißt, dass die Last auf viele Knoten, die aus billiger, nicht unbedingt ausfallsicherer Hardware bestehen können, verteilt wird. [6,7]

1.3 Fazit

Es ist wichtig zu beachten, dass, obwohl dokumentorientierte Datenbanken auf einer anderen Weise als relationale Datenbanken betrieben werden, sie kein realisierbarer Ersatz für sie sind. Document-Stores sind vielmehr eine Alternative für jene Projekte, bei denen ein dokumentorientiertes Konzept eine bessere Passform als eine gewöhnliche relationale Datenbank ist. Sie eignen sich besonders gut, wenn man nicht stark vernetzte Daten speichern muss und die einzelnen Daten unterschiedliche Eigenschaften aufweisen, sodass es undenkbar wäre, eine gemeinsame Struktur anzulegen, die allen Daten gerecht wird (z.B. Wikis, Blogs und Dokumentmanagementsysteme).

2 CouchDB

CouchDB ist ein dokumentorientiertes Datenbankmanagementsystem. Der Begriff „Couch“ steht für „Cluster of unreliable commodity hardware Data Base“ (zu Deutsch: „Datenbank auf einem Cluster aus unzuverlässiger Standardhardware“) und repräsentiert das Ziel der extremen Skalierbarkeit, hohen Verfügbarkeit und Zuverlässigkeit [10].

Eine Besonderheit von CouchDB liegt in ihrer einfachen Bedienbarkeit. Um damit zu arbeiten, wird nur das Wissen über HTTP und JavaScript benötigt. Die Interaktionen mit den Datenbanken erfolgen über eine HTTP-RESTful-Schnittstelle mittels JavaScript, welches von Mozillas JavaScript Engine Spidermonkey ausgeführt wird. [11]

Eine Datenbank besteht aus beliebig vielen Dokumenten, welche wiederum in beliebig viele Schlüssel-Werte-Paare gegliedert sind. Zusätzlich beinhaltet jedes Dokument seine ID und seine Versionsnummer. Wenn Änderungen an einem CouchDB-Dokument vorgenommen werden, werden diese Änderungen nicht wirklich an das vorhandene Dokument angehängt, sondern eine neue Version des gesamten Dokuments, „Revision“ genannt, wird erstellt. Dies bedeutet, dass ein vollständiger Verlauf der Dokumentenmodifikationen automatisch von der Datenbank verwaltet wird. Diese Revisionskontrolle wird Multiversion Concurrency Control genannt. [12]

CouchDB verfügt über eine leistungsfähige B-Baum-Speicher-Engine. Sie ist dafür verantwortlich ist, die Daten in CouchDB zu sortieren, und bietet einen Mechanismus für die Suche, das Einfügen und das Löschen in logarithmischen Zeiten. CouchDB verwendet diese Engine für alle internen Daten, Dokumente und Views. [12]

Wegen der schemafreien Struktur der Datenbank, ist CouchDB abhängig von der Verwendung von Views, um Anfragen zu beantworten und beliebige Beziehungen zwischen den Dokumenten zu erstellen. Die Ergebnisse dieser Views werden dynamisch mit Map/Reduce berechnet und haben keinerlei Auswirkungen auf andere Dokumente in der Datenbank. Das Map/Reduce-Modell kann in den Map-Schritt und in den Reduce-Schritt zerlegt werden. Im Map-Schritt wird jedes Dokument vom Meister-Knoten genommen und das Problem wird in Teilprobleme aufgeteilt. Diese Teilprobleme werden anschließend zur Arbeiter-Knoten, die das Problem lösen und die Ergebnisse an den Meister-Knoten zurückliefern. Im Reduce-Schritt nimmt der Meister-Knoten die Ergebnisse, die von den Arbeiter-Knoten empfangen wurden, und kombiniert sie zu einem Gesamtergebnis, um das ursprüngliche Problem zu beantworten. [12]

CouchDB unterstützt mehrfache Datenbank-Repliken, die auf unterschiedlichen Servern laufen können und ihre Daten synchronisieren. Dies ist nützlich in den zwei häufigen Szenarien: gelegentlich verbundene Anwendungen (z.B. PDA) und geschäftskritische Anwendungen (z.B. Cluster). Im ersten Szenario kann der Benutzer während des Offline-Modus arbeiten und seine Datenänderungen lokal speichern. Später, wenn er sich wieder zum Netzwerk verbindet, kann er seine durchgeführten Änderungen mit der Quelldatenbank synchronisieren. Im zweiten Szenario wird die Datenbank auf mehreren Servern repliziert, so dass die Zuverlässigkeit durch Redundanz und hohe Leistung durch den Lastausgleich erreicht werden können. [12]

Obwohl CouchDB ein noch sehr „junges“ Projekt ist, erfreut es sich immer größerer Beliebtheit bei Softwareprojekten, sowie Web-, Smartphone- und Facebook-Anwendungen [13]. Außerdem wird CouchDB als zentraler Bestandteil von Ubuntu eingesetzt und übernimmt dort die Aufgaben zum Synchronisieren von Adressen und Lesezeichen [14].

2.1 Entwicklung

Die Entwicklung von CouchDB wurde 2005 begonnen [15] und wird bis heute mit der Herausgabe von neuen Versionen weiterhin unterstützt. Die letzte veröffentlichte Version ist 1.1.1 (31. Oktober 2011) [16]. Der Erfinder der CouchDB-Engine, Damien Katz, der vorher bei IBM als Senior-Developer der dokumentorientierten Datenbank Lotus-Notes beschäftigt war, hat an der Entstehung alleine gearbeitet und dann mit Hilfe anderer Programmierer [17].

CouchDB wurde ursprünglich in C++ geschrieben, aber im April 2008 zog das Projekt auf die Erlang-Plattform [18] um. Die Begründung dafür war der Wunsch nach dem Erreichen der Fehlertoleranz. Erlang ist eine funktionale Sprache, die sehr mächtig in der Entwicklung von verteilten Systemen ist [19]. Das Konzept dieser Sprache eignet sich sehr gut um die Parallelität des von CouchDB benutzten Map/Reduce-Modells umzusetzen. Auf welchen Plattformen CouchDB installiert werden kann, hängt lediglich davon ab, ob die Plattform einen nativen Erlang-Code-Compiler enthält. CouchDB kann auf den meisten POSIX-Systemen laufen – einschließlich Linux und Mac OS X [20]. Windows wird derzeit nicht offiziell unterstützt.

Im November 2007 wies die Organisation IANA der CouchDB offiziell die Portnummer 5984 (TCP/UDP) zu [21].

Die Veröffentlichung von CouchDB erfolgte unter der GNU General Public License. Im Februar 2008 wurde Damien Katz von Apache Software Foundation angestellt [22] und seitdem wird die Arbeit an CouchDB unter der Obhut dieser Organisation fortgeführt. CouchDB hat dort den Status des Top-Level-Projekts erlangt [23], das heißt die Datenbank steht prinzipiell auf einer Stufe mit dem Apache-Webserver oder Tomcat. Durch die Apache-Lizenz wurde CouchDB zu einem vollwertigen Open-Source-Projekt. Wie bei den meisten Open-Source-Lizenzen, ist es erlaubt den Quellcode zu modifizieren und in der anderen Software zu verwendet.

Im Januar 2012 hat Damien Katz bekanntgegeben, dass er sich in Zukunft nicht mehr beim CouchDB-Apache-Projekt engagieren möchte [24]. Der Grund dafür waren die Meinungsverschiedenheiten innerhalb des Teams bezüglich der von Damien Katz angestrebten dynamischen Entwicklung von CouchDB. Gegenwärtig sieht er die Zukunft in der Entwicklung seiner neuen Datenbank unter dem Namen Couchbase. Nichtsdestotrotz versichert die Apache-Community, dass sie auch in Zukunft auf das CouchDB-Projekt setzt und es nach Kräften unterstützt [25].

2.2 Anatomie der Dokumente

Wie bei allen dokumentorientierten Datenbanken bilden die Dokumente die zentrale Datenstruktur von CouchDB. Ein in der definitiven Referenz von CouchDB vorgestelltes Beispiel, wo der Vergleich zwischen Dokumenten und Visitenkarten gezogen wird, kann eine Motivation zum Verständnis dieses Designs sein. Jede Visitenkarte ist individuell und unterliegt keinem festen Schema. Darauf können sich Name einer Person oder einer Organisation, unterschiedliche Kontaktinformation, Firmenlogos und andere Inhalte befinden. Wie die Visitenkarten in sich abgeschlossene Daten sind, so können auch die Dokumente betrachtet werden. [26]

2.2.1 JSON

Die CouchDB-Dokumente sind sehr flexibel. Sie werden in einer Datenbank als Objekte im JSON-Format abgelegt. Aktuell beträgt die maximale Größe eines Dokuments 4GB [27]. Ansonsten gibt es keine Einschränkungen, wie ein Dokument strukturiert werden soll oder was es enthalten soll (solange ein Dokument gültiges JSON-Format aufweist). JSON (JavaScript Object Notation) ist ein kompaktes Daten-Austausch-Format. Es wird oft zum Serialisieren und Übertragen von strukturierten Daten über eine Netzwerkverbindung verwendet. JSON unterstützt sechs Datentypen: Nullwert, boolescher Wert, Zeichenkette, Zahl, Array und Objekt. Ein Nullwert wird durch das Schlüsselwort `null` dargestellt. Ein boolescher Wert ist entweder das Schlüsselwort `true` oder `false`. Eine Zeichenkette ist eine Folge von Unicode-Zeichen (standardmäßig UTF-8), die mit den doppelten Anführungszeichen " anfangen und enden. Eine Zahl ist eine Folge von Ziffern `0-9`, eine negative Zahl wird mit einem Minus − als Vorzeichen gekennzeichnet und eine reelle Zahl wird mit einem Punkt unterbrochen. Ein Array wird von eckigen Klammern umgeben und beinhaltet eine durch Komma getrennte, geordnete Menge von Eigenschaften vom beliebigen Datentyp. Ein Objekt wird von geschweiften Klammern umgeben und beinhaltet eine durch Komma getrennte, ungeordnete Menge von Schlüssel-Wert-Paaren, die durch einen Doppelpunkt separiert werden, wobei der Schlüssel eine Zeichenkette ist und der Wert vom beliebigen Datentyp sein kann. [28]

Um uns einen besseren Eindruck in die Dokumente zu gewähren, wandeln wir ein relationales Datenbankschema in das Dokumentenschema um. Dazu nehmen wir ein Adressbuch als Anwendungsfall, wo zu den Kontaktpersonen Telefonnummern und Anschriften zusammengetragen sind.

person				
id	**first name**	**last name**	**birth date**	**sex**
1	John	Smith	1980/09/29	m
2	Mei	Wang	1984/04/10	f

phone		
person	**type**	**number**
2	home	852 555-1234
2	mobile	924 555-4321

address						
person	**country**	**state**	**city**	**postal code**	**street**	**house**
1	United States	NY	New York	10017	Madison Ave	41
2	Hong Kong	null	Wan Chai	null	Hennessy Rd	529

Aufgrund der Normalisierung sind die zusammengesetzten Attribute, die zum Identifizieren der Adressen gehören, in einer eigenen Tabelle gespeichert – es besteht eine 1:1-Beziehung zu den Personen. Die Personen und ihre Telefonnummern weisen eine 1:N-Beziehung auf.

Auch wenn CouchDB keine Beziehungen zwischen den Dokumenten unterstützt, lassen sie sich manche mit den Arrays oder mit den eingebetteten Objekten realisieren. Die 1:N-Beziehung wird je nach Anwendungsfall entweder mit separierten Dokumenten oder mit einem eingebetteten Array umgesetzt. Im ersten Fall werden die Dokumente auf der N-Seite um einen Attribut, der die Referenz auf ein anderes Dokument darstellt und seine ID beinhaltet, erweitert. Der zweite Fall wird bei den schwachen Entitäten eingesetzt; die Dokumente vom schwachen Entitätstyp werden im referenzierten Dokument in einem Array gekapselt. Für die Erstellung einer M:N-Beziehung muss in den Dokumenten auf einer der beiden Relationsseiten ein Array mit den IDs, die sich auf die Dokumente auf der anderen Relationsseite beziehen, erstellt werden. Zu beachten ist, dass in den Fällen, wo in einem Dokument ein Attribut mit einer auf ein anderes Dokument beziehende ID angelegt wird, CouchDB zum keinen Zeitpunkt überprüfen kann, ob das Dokument mit dieser ID in der Datenbank existiert. [29]

```
{
  "type":"person",
  "firstName":"John",
  "lastName":"Smith",
  "birthDate":"1980/09/29",
  "sex":"m",
  "address":{
    "country":"United States",
    "state":"NY",
    "city":"New York",
    "postalCode":"10017",
    "street":"Madison Ave",
    "house":41
  },
  "phones":[]
}
```

```
{
  "type":"person",
  "firstName":"Mei",
  "lastName":"Wang",
  "birthDate":"1984/04/10",
  "sex":"f",
  "address":{
    "country":"Hong Kong",
    "city":"Wan Chai",
    "street":"Hennessy Rd",
    "house":529
  },
  "phones":[
    {"home":"852 555-1234"},
    {"mobile":"924 555-4321"}
  ]
}
```

Da es keine Tabellen in CouchDB gibt, so wird die Zugehörigkeit eines Dokumentes zu der jeweiligen „Tabelle“ üblicherweise mit dem `type`-Attribut gekennzeichnet.

Betrachten wie die Verwendung von NULL-Werten in relationalen Datenbanken; in einer Tabelle würden wir die Adressen in den Ländern, die keine Postleitzahlen verwenden, üblicherweise mit dem Setzten des PLZ-Attributs auf NULL repräsentieren. In CouchDB und in anderen ähnlichen Systemen, würde das Fehlen einer Postleitzahl einfach durch die Abwesenheit ihres Feldes im Dokument selbst bezeichnet werden. Das ist eine größere Übereinstimmung mit dem, was wir intuitiv aus praktischer Erfahrung erwarten.

Die Schemafreiheit von CouchDB ermöglicht weitere Design-Entscheidungen, die dazu helfen können den Speicherverbrauch zu verringern. So haben wir im unseren Beispiel die zwei Attribute von Telefonnummern zu einem zusammengefasst. Je welches Feld im Objekt existiert, kann der Wert dieses Feldes einem bestimmten Typ zugeordnet werden.

```
"phones":[
  {"type":"home","number":"852 555-1234"},
  {"type":"mobile","number":"924 555-4321"}
]
```

```
"phones":[
  {"home":"852 555-1234"},
  {"mobile":"924 555-4321"}
]
```

2.2.2 ID

Ein weiterer wichtiger Punkt, der bei der Gestaltung der Dokumentenstruktur zu beachten ist, ist die Definition der für die Dokumente zu verwendeten ID. Die ID ist eine Zeichenkette – es gibt keine Einschränkungen welche Zeichen man dafür verwendet. Allerdings soll man beachtet, dass auf die Dokumente per URL mit der Angabe der Dokumenten-ID zugegriffen wird. Deswegen soll man die Sonderzeichen, die die URL ungültig machen können, vermeiden [30]. Die ID muss nicht nur innerhalb ihrer Datenbank einzigartig sein, sondern auch in allen Instanzen dieser Datenbank. CouchDB verwendet Dokument-IDs, um Änderungen zwischen den Servern zu replizieren. Es wird empfohlen, dass man die natürlichen Schlüssel für die IDs der Dokumente verwendet. Der natürliche Schlüssel ist ein beliebiges Feld (oder eine Kombination von Feldern) in einem Dokument, der dieses Dokument eindeutig identifiziert. Die Datenbankgröße wird von den Dokumentengrößen aber auch mehrfach von den ID-Größen abgeleitet. Das `_id`-Attribut ist nicht nur im Dokument vorhanden, sondern wird auch als Teil der B-Baum-Struktur von CouchDB dupliziert, um in der Datei bei der Suche nach den Dokumenten zu navigieren. Als ein Beispiel aus der Praxis – wenn der Benutzer die Größe der IDs von 16 Bytes auf 4 Bytes reduziert, so reduziert sich auch die Größe der Datenbankdatei mit 10 Millionen Dokumenten von 21GB auf 4GB (mit der Annahme, dass alle Dokumente insgesamt 2GB groß sind) [31].

Im unseren Adressbuch-Beispiel ist die Kombination aus dem Vor- und Nachnamen, dem Geburtsdatum und dem Geschlecht eine ausreichende Lösung für die Festlegung der ID. Dabei müssen aber die Attribute, aus denen sich die ID zusammensetzt, unveränderbar gemacht werden, weil die ID eines Dokuments vor Änderungen gesperrt ist (Abschnitt 2.6.2 Validierungsfunktionen).

```
{
  "type":"John:Smith:19800929:m",
  "firstName":"John",
  "lastName":"Smith",
  "birthDate":"1980/09/29",
  "sex":"m",
  ...
}
```

```
{
  "type":"Mei:Wang:19840410:f",
  "firstName":"Mei",
  "lastName":"Wang",
  "birthDate":"1984/04/10",
  "sex":"f",
  ...
}
```

2.2.3 Versionsnummer

Jedes Mal, wenn ein Dokument angelegt oder bearbeitet wird, wird ihm von CouchDB eine neue Versionsnummer, `_rev`-Attribut, automatisch zugewiesen (Abschnitt 2.9 B-Tree-Datenstruktur und MVCC). Die Versionsnummer ist ein MD5-Hashwert des Dokuments mit einem N-Präfix, der angibt, wie oft das Dokument bereits aktualisiert wurde. Der MD5-Hashwert setzt sich aus allen Attributen außer dem ID-Attribut zusammen. So haben zwei Dokumente mit dem gleichen Inhalt und unterschiedlichen IDs denselben Hashwert. Diese Nummern werden bei den Leseoperationen dazu genutzt, um die letzten Änderungen des Dokuments zu finden. [32,33]

```
{
  "_id":"John:Smith:19800929:m",
  "_rev":"1-9fa8fd74d0321a25d3f661632ccfb57d",
  ...
}
```

2.3 RESTful-JSON-API

CouchDB bietet eine nackte HTTP-Schnittstelle um die Daten aus der Datenbank abzurufen. Auf diese Schnittstelle wird mit Hilfe von Anfragen zugegriffen. Als Rückantwort erhält man die Daten im JSON-Format. Somit können Web-Dienste und Anwendungen, die über eine HTTP-Schnittstelle verfügen, die Datenbankoperation unabhängig von der verwendeten Sprache durchführen.

REST (Representational State Transfer) ist ein Programmierparadigma für Webanwendungen. Es fordert die Entwickler auf, die HTTP-Methoden einheitlich der Protokoll-Definition zu verwenden. Dafür gibt es grundlegende Operationen: `POST`, `GET`, `PUT` und `DELETE`, die bei den HTTP-Methoden eingesetzt werden. Mit `POST` erstellt man eine Ressource auf dem Server. Mit `GET` wird eine Ressource vom Server abgerufen. Mit `PUT` bearbeitet man eine Ressource. Mit `DELETE` wird eine Ressource vom Server entfernt. [34]

Aus Gründen der Einfachheit und zur Veranschaulichung der rohen JSON-Antworten, die die RESTful-JSON-API liefert, verwendet CouchDB das Kommandozeilentool `curl`. Mit Hilfe eines der unterstützten Protokolle (DICT, FILE, FTP, FTPS, GOPHER, HTTP, HTTPS, IMAP, IMAPS, LDAP, LDAPS, POP3, POP3S, RTMP, RTSP, SCP, SFTP, SMTP, SMTPS, TELNET und TFTP) ermöglicht `curl` die Übertragung der Daten von und zu einem Server. [35]

Eine REST-Web-Service-Anwendung (oder Client) umfasst im HTTP-Header und im Körper einer Anfrage alle Parameter, den Inhalt und die Daten, die von der Seite des Servers benötigt werden, um eine Antwort zu generieren. Die Komponente auf dem Server wird über ihre URL angesprochen. Der `authority`-Teil der URL besteht aus der Hostadresse, die auf die Serverressource bezieht, und der Angabe der Portnummer von CouchDB 5984.

```
HOST="http://127.0.0.1:5984"
```

Ist CouchDB nicht mehr freizugänglich – falls es durch die Administrationsrechte geschützt ist – dann müssen im `authority`-Teil der URL zusätzlich der Benutzername und das Passwort für die Authentifizierung mitangegeben werden.

```
HOST="http://username:password@127.0.0.1:5984"
```

Mit einer einfachen `GET`-Anfrage an die CouchDB-Adresse ohne zusätzlichen Parameter kann der Status des Systems ermittelt werden. Dabei kann der `GET`-Befehl ausgelassen werden, da `curl` die Anfragen standardmäßig als Leseoperationen annimmt [35].

```
curl $HOST/
```

Falls CouchDB aktiviert ist, erhält man eine Begrüßung und die aktuell installierte Versionsnummer von CouchDB.

```
{"couchdb":"Welcome","version":"1.1.1"}
```

2.3.1 HTTP-Datenbank-API

Die Liste aller verfügbaren Datenbanken kann über die spezielle `_all_dbs`-URL beschafft werden.

```
curl $HOST/_all_dbs
```

CouchDB gibt ein Array zurück, in dem die Namen aller vorhandenen Datenbanken als Zeichenketten repräsentiert sind.

```
["db1","db2"]
```

Falls noch keine Datenbank angelegt wurde, wird ein leeres Array zurückgegeben.

```
[]
```

Will man eine benutzerdefinierte Request-Methode verwendet, um mit dem HTTP-Server zu kommunizieren, wird es bei dem `curl`-Tool mit der `-X`-Option spezifiziert. Die spezifizierte Anfrage wird anstelle von der standartmäßigen GET-Methode verwendet. Die üblichsten zusätzlichen HTTP-Methoden sind `POST`, `PUT` und `DELETE`. [35]

Eine neue leere Datenbank lässt mit dem PUT-Befehl zusammen mit der Angabe des gewählten Datenbanknamens, der als Pfad an die Adresse angehängt wird, erzeugen. Der Name darf nur Kleinbuchstaben (a-z), Ziffern (0-9) und die Zeichen _$()+-/ beinhalten und muss mit einem Kleinbuchstaben anfangen [36].

```
curl -X PUT $HOST/db
```

Wurde die Datenbank erfolgreich angelegt, erhält man ein "ok" als Antwort. Dieses Argument wird bei den meisten erfolgreich abgeschlossenen Manipulierungsoperationen im Rückgabeobjekt mitgeliefert.

```
{"ok":true}
```

Konnte die Operation nicht abgeschlossen werden, z.B. wenn die Datenbank mit diesem Namen bereits vorhanden ist, erscheint eine Fehlermeldung mit der Fehlerart und der Begründung, warum dieser Fehler entstanden ist.

```
{"error":"file_exists",
 "reason":"The database could not be created, the file already exists."}
```

Um eine Datenbank wieder zu löschen, wird anstelle von PUT die DELETE-Operation verwendet.

```
curl -X DELETE $HOST/db
```

Beim erfolgreichen Löschen der Datenbank mitteilt CouchDB eine Bestätigung.

```
{"ok":true}
```

Man erhält einen Fehler, wenn die Datenbank mit dem angefragten Namen nicht existiert.

```
{"error":"not_found","reason":"missing"}
```

2.3.2 HTTP-Dokument-API

Die spezielle _all_docs-URL beschafft die Auflistung aller Dokumente einer Datenbank.

```
curl $HOST/db/_all_docs
```

Als Rückgabe wird eine Map/Reduce-View über alle Dokumente zusammen mit Angabe ihrer aktuellen Versionsnummer generiert (Abschnitt 2.6.1 Map- und Reduce-Funktionen). Die Dokumente in dieser View sind nach der ID aufsteigend sortiert (Klein-/Großschreibung wird beachtet).

```
{"total_rows":3,"offset":0,"rows":[
  {"id":"doc1","key":"doc1","value":{"rev":"1-23202479"}},
  {"id":"doc2","key":"doc2","value":{"rev":"2-0732d121"}},
  {"id":"doc3","key":"doc3","value":{"rev":"3-35f70da1"}}
]}
```

curl bietet die Möglichkeit bestimmte Daten in einem Request an den HTTP-Server zu versenden, auf die gleiche Weise wie es ein Browser tut, wenn ein Benutzer ein HTML-Formular ausfüllt und auf den Absenden-Button drückt. Mit der aktivierten -d-Option können die Daten im JSON-Format an den Server weitergereicht werden. [35]

Es gibt zwei Möglichkeiten ein Dokument in einer Datenbank anzulegen. Die POST-Operation wird dafür benutzt, um ein neues Dokument mit einer vom Server generierten ID zu erstellen. Dazu muss die URL auf den Speicherort der Datenbank verweisen. In den Körper der Anfrage wird ein JSON-Objekt, das den Inhalt des Dokuments repräsentiert, hineingeschrieben.

```
curl -X POST $HOST/db -d '{"a":1}'
```

Die dabei generierte ID ist ein Universally Unique Identifier (UUID). Es ist eine zufällige 16-Byte-Zahl, die hexadezimal formatiert ist. Es gibt 3×10^{38} Möglichkeiten für diese Zahl [37]. Obwohl die Wahrscheinlich sehr gering ist, dass die generierte UUID in der Datenbank schon vorkommt, empfiehlt sich trotzdem, die Dokumenten-ID selber zu wählen. Zu diesem Zweck wird die PUT-Operation statt POST benötigt. Dabei muss die URL auf den Speicherort des Dokuments verweisen; die gewählte ID wird an den Pfad zur Datenbank angehängt.

```
curl -X PUT $HOST/db/doc -d '{"a":1}'
```

Um ein Dokument zu aktualisieren oder zu löschen muss das `_rev`-Attribut mitübergeben werden. Damit stellt CouchDB sicher, dass sich das Dokument in der Zwischenzeit nicht geändert hat. Stimmen die Versionsnummern nicht überein, wird CouchDB die Änderung oder das Löschen des Dokuments ablehnen und einen Konfliktfehler werfen.

```
{"error":"conflict","reason":"Document update conflict."}
```

Man kann die Felder eines Dokuments nicht einzeln ändern, sondern muss das neue Dokument vollständig angeben. Die Aktualisierung des Dokuments verläuft ebenfalls mit dem `PUT`-Request. In diesem Fall muss das JSON-Objekt das `_rev`-Attribut im Körper beinhalten.

```
curl -X PUT $HOST/db/doc -d '{"_rev":"1-dcf48548","a":2,"b":3}'
```

Ausführung einer `DELETE`-Operation löscht das vorhandene Dokument. Dabei muss die aktuelle Versionsnummer als `rev`-Parameters zusammen mit der URL weitergereicht werden.

```
curl -X DELETE $HOST/db/doc?rev="2-5ef23287"
```

Um Eventual Consistency zu realisieren, bleiben die gelöschten Dokumente in der Datenbank für immer bestehen. Sie werden mit dem speziellen Feld `"_deleted":true` belegt und dadurch als gelöscht markiert, während alle anderen Felder in dieser Version des Dokuments entfernt werden. Wird ein Dokument, mit der gleichen ID nochmal angelegt, so beginnt seine Versionsnummer nicht mit einer 1, sondern wird ab der Versionsnummer des gelöschten Dokuments wiederfortgesetzt.

Jedes Mal wenn CouchDB die Änderung, die Erstellung oder die Löschung des Dokuments akzeptiert, dann weist es ihm eine neue Versionsnummer zu und gibt die ID und diese Versionsnummer des Dokuments zurück.

```
{"ok":true,"id":"doc","rev":"3-5a0de159"}
```

Dokumente lassen sich mit einer `GET`-Operation, angewendet auf den Pfad zum gefragten Dokument, einzeln anzeigen.

```
curl $HOST/db/doc
```

CouchDB gibt die aktuelle Version des Dokuments als JSON-Objekt vollständig mit allen Attributen, inklusive der ID und der Versionsnummer, zurück.

```
{
  "_id":"doc",
  "_rev":"3-5a0de159",
  "a":1
}
```

Will man auf die früheren Versionen zugreifen, muss der `rev`-Parameter weitergereicht werden.

```
curl $HOST/db/doc?rev="1-dcf48548"
```

Um herauszufinden, welche Versionen für das jeweilige Dokument zur Verfügung stehen, wird die Abfrage des Dokuments mit dem Parameter `revs_info=true` spezifiziert.

```
curl $HOST/db/doc ?revs_info=true
```

Dies gibt die aktuelle Version des Dokuments zurück, allerdings mit dem zusätzlichen Feld `_revs_info`, dessen Wert ein Array aus Objekten ist – ein Objekt pro Revision.

```
{
  "_id":"doc",
  "_rev":"3-61632ccf",
  "a":1,
  "_revs_info":[
    {"rev":"3-61632ccf","status":"available"},
    {"rev":"2-e8155f13","status":"deleted"},
    {"rev":"1-32876f95","status":"missing"}
  ]
}
```

Der Status `available` bedeutet, dass der Inhalt dieser Revision in der Datenbank gespeichert ist und darauf zugegriffen werden kann. Die anderen Werte zeigen an, dass die Revisionsinhalte nicht verfügbar sind. [38]

2.4 Futon

In den meisten CouchDB-Installationen ist die freie Fernwartungssoftware Futon integriert. Futon ist eine grafische JavaScript-Anwendung, die, wenn CouchDB auf dem lokalen Rechner installiert ist, in einem Webbrowser unter der `_utils`-URL erreicht werden kann [39].

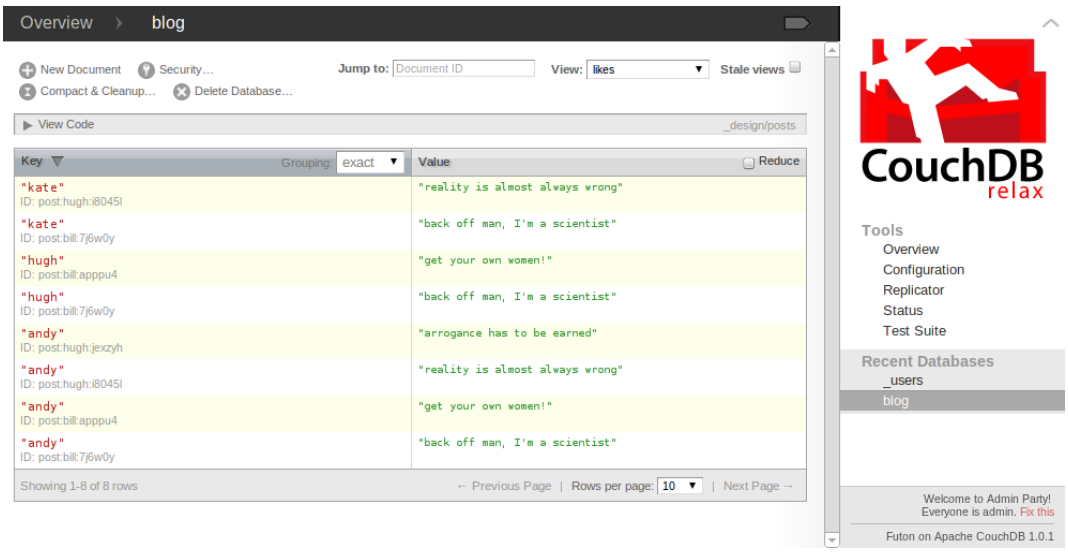
```
http://127.0.0.1:5984/_utils/
```

Es können auch Datenbanken auf fremden Rechnern über eine Netzwerkverbindung administriert werden. Dazu muss auf dem eigenen lokalen Rechner ein SSH-Tunnel eingerichtet werden.

```
$ ssh -L 5984:127.0.0.1:5984 ssh.example.com
```

Dieser Tunnel leitet die Anfragen, die an den lokalen Port 5984 adressiert waren, zum entfernten Server-Port 5984 weiter. [10]

Futon bietet den vollen Zugang zu sämtlichen Funktionen von CouchDB und macht es einfach mit den komplexeren Features von CouchDB zu experimentieren. Mit einfachen Mausklicks kann man die Datenbanken und die Dokumente anzeigen, anlegen, löschen und bearbeiten. Des Weiteren kann man die Abfragen in Form einer Map/Reduce-View gegen eine Datenbank ausführen, sowie die Replikation zwischen den Datenbanken erzeugen.



2.5 Sicherheit

Bei der Standardeinstellung ist jeder Benutzer ein Administrator. Wenn CouchDB gerade installiert wurde, darf jeder Request von jedem ausgeführt werden. In CouchDB, wo jeder alle Privilegien besitzt, nennt man das die Admin-Party [40]. Wird CouchDB mit solcher Konfiguration an das öffentliche Internet angeschlossen, besteht die Gefahr, dass ein Angreifer die Admin-Rechte für das komplette System erlangt. Um dies zu vermeiden, soll man den Zugriff zu den Datenbanken beschränken.

Als Erstes muss ein Serveradministratorkonto angelegt werden. Nur der Serveradministrator kann Datenbanken erstellen und löschen.

```
curl -X PUT $HOST/_config/admins/admin -d '"password"'
```

Zum Schutz von dem, wer die Dokumente lesen und aktualisieren kann, hat CouchDB einen einfachen Mechanismus, der erweitert werden kann, um benutzerdefinierte Sicherheitskonzepte zu implementieren. Pro Datenbank können Datenbankadministratoren und Datenbankleser angelegt werden. Datenbankleser können alle Dokumente außer den Design-Dokumenten innerhalb ihrer Datenbank abfragen, bearbeiten und erstellen. Die Datenbankadministratoren haben alle Privilegien innerhalb ihrer Datenbank. Außerdem können sie weitere Datenbankleser und -administratoren einfügen und entfernen. [41]

Datenbankadministratoren und Datenbankleser werden im Sicherheitsobjekt einer Datenbank definiert. Es befindet sich unter der Datenbank-URL am Pfad `_security`.

```
curl $HOST/db/_security
```

Das Sicherheitsobjekt ist ein JSON-Dokument. Zu beachten ist, dass es kein regulär versionisiertes CouchDB-Dokument ist (es unterliegt keinen MVCC-Regeln) [41]. Es ist nur eine Design-Entscheidung um die Autorisierungsüberprüfung zu beschleunigen. Dieses Dokument kann mit Hilfe der Futon-Webanwendung oder mit dem Kommandozeilentool `curl` editiert werden.

```
{
  "admins":{
    "names":["anna","finn"],
    "roles":["boss"]
  },
  "readers":{
    "names":["lisa"],
    "roles":["employee","guest"]
  }
}
```

Nachdem die Rollen und die Namen der Benutzer vordefiniert wurden, können im weiteren Verlauf neben den Serveradministratoren auch diese Benutzer, die die zugelassenen Rollen haben, auf die Datenbank zugreifen.

Damit sich diese Benutzer authentifizieren können, müssen ihnen Rollen und Passwörter zugeteilt werden. Diese Informationen werden in einer Authentifizierungsdatenbank mit dem Namen `_users` gespeichert. Beim jeden Dokument in dieser Datenbank kann der `type`-Attribut nur auf den Wert `"user"` gesetzt werden. Im Feld `roles` werden die Rollen des Benutzers als ein Array aus Zeichenketten festgelegt. Passwort wird in zwei Feldern kodiert gespeichert.

```
{
  "_id":"org.couchdb.user:lisa",
  "type":"user",
  "name":"lisa",
  "roles":["guest"],
  "password_sha":"fe95df1ca59a9b567bdca5cbaf8412abd6e06121",
  "salt":"4e170ffeb6f34daecfd814dfb4001a73"
}
```

Die Felder `password_sha` und `salt` werden von CouchDB automatisch erstellt. Dazu muss das `password`-Feld mit dem drin angegebenen Passwort im Dokument präsent sein. Wenn das Benutzer-Dokument in die Datenbank geschrieben wird, überprüft CouchDB die Existenz des `password`-Feldes – wenn es vorhanden ist, wird automatisch eine zufällige 128-bit-UUID erzeugt, mit der das Passwort verkettet und gehasht wird. Anschließend werden das gehashte Passwort im Feld `password_sha` und die generierte UUID im Feld `salt` abgelegt. Das `password`-Feld wird inzwischen entfernt. Dieser komplette Vorgang hat den Vorteil, dass die Benutzer keine Salz- und Hashwerte für das Passwort manuell zu berechnen brauchen. [40,41]

Die `_user`-Datenbank unterliegt bestimmten Regeln, wie die Dokumente strukturiert werden sollen (Abschnitt 2.6.2 Validierungsfunktionen). Ein anonymer Benutzer kann nur neue Dokumente erstellen. Ein authentifizierter Benutzer kann nur sein eigenes Dokument aktualisieren. Ein Server- oder Datenbankadministrator kann auf alle Dokumente zugreifen und sie bearbeiten. Die ID des Dokuments muss sich aus dem Namen des Benutzers mit dem Präfix `"org.couchdb.user:"` zusammensetzen. Wenn ein Dokument nicht von einem Serveradministrator erstellt wird, muss das Attribut `"roles"` ein leeres Array `[]` sein. Wenn ein Dokument nicht von einem Serveradministrator bearbeitet wird, muss das Attribut `"roles"` unverändert zurückgelassen werden. Die Namen der Rollen und der Benutzer dürfen nicht mit einem Unterstrich _ beginnen, da solche Namen vom System reserviert sind. Die Benutzerdokumente, die die Serveradministratoren repräsentieren, dürfen keine Attribute `"password_sha"` und `"salt"` besitzen, weil ihre Passwörter in der Konfigurationsdatei gespeichert sind. [41]

2.6 Design-Dokumente

Ein Design-Dokument ist ein gewöhnliches CouchDB-Dokument mit dem Unterschied, dass dessen ID mit `_design/` beginnt. Ansonsten verhalten sich Design-Dokumente genau wie jedes andere Dokument in CouchDB. Sie werden mit den anderen Dokumenten versionisiert und repliziert. Design-Dokumente werden dafür benutzt um den Quellcode für Anwendungen zu enthalten.

```
{
  "_id":"_design/doc",
  "_rev":"4-9fa8fd74",
  "views":{
    "view1":{
      "map":"function(doc) {...}"
    },
    "view2":{
      "map":"function(doc) {...}",
      "reduce":"function(keys, values) {...}"
    }
  },
  "validate_doc_update":"function(newDoc, oldDoc, userCtx) {...}"
}
```

Neben den Map/Reduce- und Validierungsfunktionen gibt es eine große Menge anderer Funktionalitäten, welche die Design-Dokumente bieten und die mit den neuen Versionen von CouchDB erweitert werden. In Attributen `show` und `list` können Funktionen enthalten sein, um JSON in HTML, XML oder andere Ausgabeformate zu konvertieren. In Anhängen können JavaScript-Bibliotheken und HTML-Templates abgelegt werden. Damit lassen sich komplette Standalone-Anwendungen implementieren. In diesem Abschnitt wird es nicht auf alle Features von Design-Dokumenten eingegangen, da der Umfang sehr groß ist und es in die gröbere Richtung geht. [42]

2.6.1 Map- und Reduce-Funktionen

Neben den Dokumenten sind die sogenannten Views ein anderes wichtiges Konzept von CouchDB. Die Views werden dazu verwendet, um die relevanten Dokumente aus der Datenbank zu filtern und sie mit benötigten Daten zu extrahieren. Dementsprechend sind die Views für CouchDB das gleiche was SQL für relationalen Datenbanken ist.

CouchDB-View-Indizien werden generiert, wenn die View zum ersten Mal aufgerufen wird. Nach diesem ersten Erstellen der Views, werden sie jedes Mal aktualisiert, wenn sie wiederholt aufgerufen werden. Es empfiehlt sich, die Views so früh wie möglich zu erstellen, weil dieser Prozess ziemlich langsam ist (z.B. mit 10 000 Dokumenten kann die Generierung der Views eine Stunde dauern). [43]

Map-Funktionen sind nebenwirkungsfreie Funktionen, die ein Dokument als Argument annehmen und Schlüssel-Wert-Paaren mit Hilfe der `emit`-Funktion ausgeben. CouchDB speichert die ausgegebenen Zeilen durch den Aufbau der sortierten B-Baum-Indizien, so dass die Suche nach einer Zeile über einen Schlüssel ohne einen großen Speicherverbrauch durchgeführt werden kann. Das Erstellen einer View verläuft in einer linearen Zeit $O(N)$, wobei N die Gesamtzahl der Zeilen in der View ist. Allerdings erfolgt die Abfrage innerhalb einer View sehr schnell, da der B-Baum flach bleibt, auch wenn es sehr viele Schlüssel enthält. Reduce-Funktionen arbeiten mit den sortierten Zeilen, die von den Map-Funktionen geliefert wurden. Eine Reduce-Funktion berechnet das endgültige, reduzierte Ergebnis und gibt dieses mit der `return`-Anweisung zurück. [43]

Der Quellcode für Map- und Reduce-Funktionen wird in einem Design-Dokument unter dem Attribut `views` angelegt. In diesem Attribut wird ein JSON-Objekt gespeichert, deren Attribute die Namen der Views sind. Somit kann jedes Design-Dokument beliebige Anzahl von View-Definitionen beinhalten. Der Wert jeder dieser View ist auch ein JSON-Objekt, das das Attribut `map` besitzt und optional mit dem Attribut `reduce` ergänzt werden kann. Die Werte von `map` und `reduce` sind Zeichenketten, in denen sich die zugehörigen Funktionen befinden. Das Ergebnis einer View wird über die `_view`-URL erreicht.

```
http://127.0.0.1:5984/db/_design/doc/_view/view
```

Die nächstfolgenden Funktionalitäten der Design-Dokumente werden anhand anwendbarer Beispiele erläutert. Dafür wird im Voraus eine Datenbank `blog` vorbereitet, wo die Benutzer ihre Blogeinträge veröffentlichen können. Jeder Blogeintrag hat einen Autor, ein Erstelldatum und einen kurzen Text als Inhalt. Zusätzlich können andere Benutzer diese Einträge mit „gefällt mir“ markieren und somit deren Bewertung steigern. Im unseren Beispiel bietet sich die Kombination aus dem Autonamen und dem Erstelldatum als eine gute Passform für den natürlichen Schlüssel. Dabei speichern wir das Erstelldatum als eine Zahl in einem Stellenwertsystem zur Basis 36, um die ID möglichst kurz zu halten.

_id	author	content	postedAt	likes
post:hugh:i79opo	hugh	humanity is overrated	2004/11/16 11:20:12	
post:bill:apppu4	bill	get your own women!	1990/07/13 08:54:04	hugh, andy
post:hugh:i8045l	hugh	reality is almost always wrong	2004/11/30 17:51:21	andy, kate
post:bill:7j6w0y	bill	back off man, I'm a scientist	1984/06/08 01:02:58	kate, hugh, andy
post:hugh:jexzyh	hugh	arrogance has to be earned	2007/03/15 12:30:17	andy

Wir wollen alle Blogeinträge nach ihrer Bewertung sortiert erhalten. Dazu erstellen wir eine View unter dem Namen `by_likes` mit einer Map-Funktion.

```
function(doc) {
  if (doc.type === "post")
    emit(doc.likes.length, doc.content);
}
```

Diese Funktion nimmt ein Dokument entgegen und überprüft mit `doc.type === "post"`, ob das Dokument tatsächlich ein Blogeintrag ist. Danach wird mit der `emit`-Funktion eine Zeile mit der Anzahl der „Likes“ als Schlüssel und mit dem Textinhalt als Wert in die View eingefügt.

```
http://127.0.0.1:5984/blog/_design/posts/_view/by_likes
```

```
{"total_rows":5,"offset":0,"rows":[
  {"id":"post:hugh:i79opo","key":0,"value":"humanity is overrated"},
  {"id":"post:hugh:jexzyh","key":1,"value":"arrogance has to be earned"},
  {"id":"post:bill:apppu4","key":2,"value":"get your own women!"},
  {"id":"post:hugh:i8045l","key":2,"value":"reality is almost always wrong"},
  {"id":"post:bill:7j6w0y","key":3,"value":"back off man, I'm a scientist"}
]}
```

Das Ergebnis einer View ist ein JSON-Objekt. Im Attribut `rows` ist eine Liste von ID-Schlüssel-Wert-Tripeln, wo die Einträge nach ihrem Schlüssel aufsteigend sortiert sind.

Abfrage einer View kann mit bestimmten Parametern spezifiziert werden. `descending=true` ändert die Sortierreihenfolge von aufsteigen zu absteigend. Mit `key` erhält man nur die Zeilen mit dem angegebenen Schlüssel. Mit `startkey` und `endkey` bzw. `startkey_docid` und `endkey_docid` werden nur die Zeilen angezeigt, deren Schlüssel bzw. ID sich zwischen diesen zwei Werten befindet. `limit` grenzt die Anzahl ausgegebener Zeilen ein und `skip` überspringt erstgehende Zeilen. Mit `limit` und `skip` können große Tupelmengen, die eine View liefert, blockweise angezeigt und geblättert werden. Alle Parameter sind mit einander kombinierbar. [44]

Für ein Beispiel nehmen wir die View `by_likes` und erweitern die Abfrage mit den Parametern `descending=true` und `endkey=2`.

```
http://127.0.0.1:5984/blog/_design/posts/_view/by_likes?descending=true&endkey=2
```

```
{"total_rows":5,"offset":0,"rows":[
  {"id":"post:bill:7j6w0y","key":3,"value":"back off man, I'm a scientist"},
  {"id":"post:hugh:i8045l","key":2,"value":"reality is almost always wrong"},
  {"id":"post:bill:apppu4","key":2,"value":"get your own women!"}
]}
```

Was wir erhalten, sind die bestbewerteten Blogeinträge. Die Einträge, die weniger als zwei Likes haben, werden in der View nicht mehr angezeigt. Die Sortierreihenfolge ist so eingestellt, dass Beiträge mit besserer Bewertung weiter oben stehen.

Parameter der `emit`-Funktion sind nicht auf elementare Werte eingeschränkt. Auch Objekte und Arrays können für das Erstellen der Views genutzt werden. Als zweites Beispiel sortieren wir die Blogeinträge nach dem Autor und dem Titel. Dazu übergeben wir der `emit`-Funktion ein Array mit den Werten der `author`- und `postedAt`-Attribute.

```
function(doc) {
  if (doc.type === "post")
    emit([doc.author, doc.postedAt], null);
}
```

Als Ergebnis erhalten wir alle Blogeinträge. Die Sortierung erfolgt in aufsteigender Reihenfolge nach dem Autornamen und, beim gleichen Autornamen, nach dem Erstelldatum.

```
http://127.0.0.1:5984/blog/_design/posts/_view/by_author
```


```
{"total_rows":5,"offset":0,"rows":[
  {"id":"post:bill:apppu4","key":["bill","1990/07/13 08:54:04"],"value":null},
  {"id":"post:hugh:jexzyh","key":["bill","2007/03/15 12:30:17"],"value":null},
  {"id":"post:hugh:i79opo","key":["hugh","2004/11/16 11:20:12"],"value":null},
  {"id":"post:hugh:i8045l","key":["hugh","2004/11/30 17:51:21"],"value":null},
  {"id":"post:bill:7j6w0y","key":["karl","1984/06/08 01:02:58"],"value":null}
]}
```


`emit` kann innerhalb einer Map-Funktion auch mehrmals aufgerufen werden. Wir wollen eine Sicht auf alle vergebenen Likes angezeigt bekommen. Dazu wird die `emit`-Funktion für jeden Wert im `likes`-Array aufgerufen und sie als Schlüssel übernimmt.

```
function(doc) {
  if (doc.type === "post")
    for (var i = 0; i < doc.likes.length; i++)
      emit(doc.likes[i], doc.content);
}
```

Obwohl die Likes nicht in externen Dokumenten gehalten werden, sondern im `post`-Dokument gekapselt sind, so kann das View-Ergebnis sie eine selbstständige „Tabelle“ mit dem JOIN auf die Blogeinträge nachbilden, wo die ID als Fremdschlüssel gesehen werden kann

```
http://127.0.0.1:5984/blog/_design/posts/_view/likes
```


```
{"total_rows":8,"offset":0,"rows":[
  {"id":"post:bill:7j6w0y","key":"andy","value":"back off man, I'm a scientist"},
  {"id":"post:bill:apppu4","key":"andy","value":"get your own women!"},
  {"id":"post:hugh:i8045l","key":"andy","value":"reality is almost always wrong"},
  {"id":"post:hugh:jexzyh","key":"andy","value":"arrogance has to be earned"},
  {"id":"post:bill:7j6w0y","key":"hugh","value":"back off man, I'm a scientist"},
  {"id":"post:bill:apppu4","key":"hugh","value":"get your own women!"},
  {"id":"post:bill:7j6w0y","key":"kate","value":"back off man, I'm a scientist"},
  {"id":"post:hugh:i8045l","key":"kate","value":"reality is almost always wrong"}
]}
```


Die Reduce-Funktion kommt zum Einsatz, wenn das Ergebnis noch zusammengefasst werden soll. Wie verwenden die View `likes` aus dem vorhergehenden Beispiel und fügen eine Reduce-Funktion hinzu. Es soll ausgerechnet werden, wie viel Blogbeiträge jeder Benutzer mag. Bei der Reduce-Funktion reicht es, wenn sie nur die Anzahl der angenommenen Werte zurückgibt.

```
function(keys, values) {
  return values.length;
}
```

Allerding liefert die View die Gesamtzahl aller Zeilen der View `likes`. Um das gewünschte Ergebnis zu erreichen, müssen die Schlüssel mit dem Parameter `group=true` gruppiert werden. Die Reduce-Funktion wird dann jeweils für jede Schlüssel-Gruppe aufgerufen.

```
http://127.0.0.1:5984/blog/_design/posts/_view/likes?group=true
```


```
{"rows":[
  {"key":"andy","value":4},
  {"key":"hugh","value":2},
  {"key":"kate","value":2}
]}
```

2.6.2 Validierungsfunktionen

Die Dokumente können dynamisch durch JavaScript-Funktionen sowohl für die Sicherheit als auch für Schemaeinhaltung der Daten validiert werden. Wenn ein Dokument alle Validierungskriterien passiert, wird das Update erlaubt. Wenn die Validierung fehlschlägt, wird das Update abgebrochen und der Benutzer erhält eine Fehlermeldung als Antwort. Validierungsfunktionen sind optional, wenn man keine definiert, erfolgt auch keine Überprüfung der Daten. Sie werden in einem Design-Dokument unter dem Feld "`validate_doc_update`" gespeichert. In einem Design-Dokument kann nur eine Validierungsfunktion enthalten sein. Die Funktion wird für jedes Dokument, das erstellt oder aktualisiert wird, ausgeführt und erhält drei Parameter: das eingehende Dokument, das aktuelle Dokument und die Information über den Benutzer, der beim Ausführen des Requests diese Funktion ausgelöst hat. Ist der Wert des aktuellen Dokuments `null`, dann heißt es, dass das Dokument neuerstellt wird. Das Update eines Dokuments wird abgebrochen, wenn die Validierung einen Fehler mit `throw` wirft. [45]

Um das Prinzip der Validierungsfunktionen zu verstehen werden wir auf die Eigenschaften der fünf Beispielblogeinträge eingehen. Um den Code einfach zu halten, werden wir das Attribut `likes` nicht weiter betrachten.

Es soll sichergestellt werden, dass die Felder `author`, `content` und `postedAt` stets vorhanden sind. Hat das eingehende neue Dokument diese Felder nicht, wird ein Fehler geworfen.

```
if (!newDoc.author) throw({forbidden:"'author' required"});
if (!newDoc.content) throw({forbidden:"'content' required"});
if (!newDoc.postedAt) throw({forbidden:"'postedAt' required"});
if (!newDoc.likes) throw({forbidden:"'likes' required"});
```

Es wird geprüft ob `postedAt` vom JavaScript-Objekt `Date` erkannt wird und es ein gültiges Format "`YYYY/MM/DD hh/mm/ss`" aufweist.

```
if (isNaN(Date.parse(newDoc.postedAt)) ||
    !newDoc.postedAt.match(/^\d{4}\/\d{2}\/\d{2} \d{2}:\d{2}:\d{2}$/))
  throw({forbidden:"'postedAt' invalid"});
```

Um zu verhindern, dass der Wert eines Feldes sich ändert, muss der Wert des aktuellen Dokuments mit dem Wert des eingehenden Dokuments verglichen werden. Außerdem muss man überprüfen, ob das aktuelle Dokument überhaupt existiert. Bei den Beispieldokumenten ist es sinnvoll den Autornamen und das Erstelldatum unveränderbar zu machen, vor allem, weil diese Felder für die Zusammensetzung der ID genutzt werden.

```
if (oldDoc) {
  if (oldDoc.author !== newDoc.author))
    throw({forbidden:"'author' can not be changed"});
  if (oldDoc.postedAt !== newDoc.postedAt))
    throw({forbidden:"'postedAt' can not be changed"});
}
```

Nachdem das Ändern der Felder `author` und `poster` gesperrt wurde, kann das Format der ID geprüft werden.

```
if (!oldDoc) {
  var postedAt36 = (Date.parse(newDoc.postedAt) / 1000).toString(36);
  if (newDoc._id !== ("post:" + newDoc.author + ":" + postedAt36))
    throw({forbidden:"'_id' invalid"});
}
```

`Date.parse` ermitteln die Millisekunden seit dem 01.01.1970. Mit der Funktion `toString` kann eine Zahl zu einer gewählten Basis (maximal 36) als Zeichenkette repräsentiert werden.

Einige Validierungskriterien müssen benutzerspezifisch implementiert werden. Dazu wird der dritte Parameter der Validierungsfunktion benötigt; `userCtx` ist ein Objekt mit den Attributen `name` und `roles`. So kann die Validierungsfunktion aus dem Beispiel ergänzt werden, um das Erstellen und das Bearbeiten der Blogeinträge durch die Benutzer, die keine Administratoren oder Blogger sind, verhindern.

Wenn der Name des Benutzers nicht mit dem Namen der Autors übereinstimmt und der Inhalt der Blogeintrag geändert wurde, wird das Request abgewiesen.

```
if (oldDoc)
  if (userCtx.name !== newDoc.author)
    if (newDoc.content !== oldDoc.content)
      throw({forbidden:"not authorized to change 'content'"});
```

Die Benutzer dürfen nur ihre eigenen Beiträge löschen. Des Weiteren wird es auch den Serveradministratoren erlaubt die Löschoperation durchzuführen. Die Serveradministratoren haben immer die Rolle "`_admin`".

```
if ((newDoc._deleted === true) && (oldDoc.type === "post")) {
  if ((userCtx.roles.indexOf("_admin") === -1) && (userCtx.name !== oldDoc.author))
    throw({forbidden:"not authorized to delete post"});
  return;
}
```

Die Blogeinträge zu schreiben wird nur den Benutzern, die die Rolle "`poster`" haben, erlaubt. Die Erstelloperation wird auch dann abgewiesen, wenn der Autorname nicht mit dem Namen des Benutzers übereinstimmt.

```
if (!oldDoc) {
  if (userCtx.roles.indexOf("poster") === -1)
    throw({forbidden:"not authorized to create post"});
  if (userCtx.name !== newDoc.author)
    throw({forbidden:"'author' invalid"});
}
```

Alle diesen Kriterien werden in der Validierungsfunktion in einer sinnvollen Reihenfolge zusammengefasst. Die Funktion kann ziemlich schnell ziemlich lang werden. Deswegen empfiehlt sich für jeden Dokumenttyp ein eigenes Design-Dokument anzulegen, wo in der Validierungsfunktion nur die Dokumente des jeweiligen Typs überprüfen.

```
function validate_doc_update(newDoc, oldDoc, userCtx) {
  if ((newDoc._deleted === true) && (oldDoc.type === "post")) {
    if ((userCtx.roles.indexOf("_admin") === -1) &&
        (userCtx.name !== oldDoc.author))
      throw({forbidden:"not authorized to delete post"});
    return;
  }
  if (newDoc.type !== oldDoc.type) throw ({forbidden:"'type' can not be changed"});
  if (newDoc.type !== "post") return;
  if (!newDoc.author) throw ({forbidden:"'author' required"});
  if (!newDoc.content) throw ({forbidden:"'content' required"});
  if (!newDoc.postedAt) throw ({forbidden:"'postedAt' required"});
  if (isNaN(Date.parse(newDoc.postedAt)) ||
      !newDoc.postedAt.match(/^\d{4}\/\d{2}\/\d{2} \d{2}:\d{2}:\d{2}$/))
    throw ({forbidden:"'postedAt' invalid"});
  if (!oldDoc) {
    if (userCtx.roles.indexOf("poster") === -1)
      throw({forbidden:"not authorized to create post"});
    if (userCtx.name !== newDoc.author)
      throw({forbidden:"'author' invalid"});
    var postedAt36 = (Date.parse(newDoc.postedAt) / 1000).toString(36);
    if (newDoc._id !== ("post:" + newDoc.author + ":" + postedAt36))
      throw({forbidden:"'_id' invalid"});
  } else {
    if (oldDoc.author !== newDoc.author))
      throw({forbidden:"'author' can not be changed"});
    if (oldDoc.postedAt !== newDoc.postedAt))
      throw({forbidden:"'postedAt' can not be changed"});
    if (userCtx.name !== newDoc.author)
      if (newDoc.content !== oldDoc.content)
        throw({forbidden:"not authorized to change 'content'"});
  }
}
```

2.7 Inkrementelle Replikation

CouchDB ist ein peerbasiertes verteiltes Datenbanksystem. Eine beliebige Anzahl von CouchDB-Hosts (Server und Offline-Clients) kann unabhängige „Replikationskopien“ von der gleichen Datenbank haben, in denen die Anwendungen die volle Datenbankinteraktivität besitzen.

Die Replikation ist ein inkrementelles Prozess unter der Einbeziehung von zwei Datenbanken (Quelle und Ziel). Die Replikation überträgt nur die letzte Version jedes Dokuments; alle früheren Versionen, die auf der Quelldatenbank liegen, werden nicht in die Zieldatenbank kopiert. Der Prozess wird durch das Senden einer `POST`-Anfrage an die `_replicate`-URL gestartet. Im Körper der Anfrage muss sich ein JSON-Objekt befinden, wo die Quelle und das Ziel angegeben sind.

```
curl -X POST http://127.0.0.1:5984/_replicate
-d '{"source":"http://example.org/db","target":"http://127.0.0.1:5984/db"}'
```

Wenn man `"continuous":true` zum Replikationstrigger-Objekt hinzufügt, wird CouchDB nach dem Replizieren aller fehlenden Dokumente von der Quelle zum Ziel nicht aufhören; das System lauscht nach den Änderungen und repliziert alle neuen Dokumente automatisch.

```
{
  "source":"http://example.org/db",
  "target":"http://127.0.0.1:5984/db",
  "continuous":true
}
```

Allerding erinnert sich CouchDB nicht über die fortlaufenden Replikationen nach dem Serverneustart. Man ist verpflichtet sie nochmals auszulösen, wenn man den Server wieder hochfährt. Mit dem Attribut `"cancel":true` unterbricht man die laufende Replikation, was aber mit dem Wurf des Fehlers 500 (Herunterfahren) die Anwendung zum Sturz bringt. [46]

Es können auch partielle Repliken erstellt und gepflegt werden. Die Replikation kann durch eine JavaScript-Funktion gefiltert werden, so dass nur bestimmte Dokumente oder solche, die bestimmte Kriterien erfüllen, repliziert werden. Dies kann den Benutzern erlauben, Teilmengen einer großen gemeinsamen Datenbank-Anwendung offline für den eigenen Gebrauch zu verwenden. Die Filter-Funktionen erwarten zwei Parameter, das zu replizierende Dokument und die Replikationsanfrage, und geben `true` zurück, falls dieses Dokument repliziert werden soll.

```
function(doc, req) {
  if (req.key === "value")
    if (doc.type === "type")
      return true;
  return false;
}
```

Die Filter-Funktionen existieren in Design-Dokumenten unter dem Feld `filters`.

```
{
  "_id":"_design/doc",
  "filters":{
    "filter":"function(doc, req) {...}"
  }
}
```

Um den Filter zu aktivieren, muss bei einer Replikation im JSON-Objekt noch ein zusätzliches Attribut `filter` mit dem Pfad zur diesen Filter-Funktion, die sich in der Quelldatenbank befindet, übergeben werden. Unter dem Feld `"query_params"` können zusätzliche Argumente, die diese Funktion als zweiten Parameter erhält, angegeben werden. [46]

```
{
  "source":"http://example.org/db",
  "target":"http://127.0.0.1:5984/db",
  "filter":"doc/filter",
  "query_params":{"key":"value"}
}
```

CouchDB erlaubt einer beliebigen Anzahl von widersprüchlichen Dokumenten gleichzeitig in der Datenbank zu existieren, wobei jede Datenbankinstanz deterministisch entscheidet, welche Dokumente die Konfliktverursacher sind. Nur das Gewinner-Dokument kann in den Views angezeigt werden, während die Verlierer-Dokumente immer noch in der Datenbank vorhanden sind, bis sie manuell oder während einer Datenbank-Reinigung gelöscht werden. Weil die Konflikt-Dokumente trotzdem gewöhnliche Dokumente sind, werden sie mit den anderen Dokumenten repliziert und unterliegen den gleichen Sicherheits- und Validierungsregeln. Da alle automatisierten Agenten, die diesen Gewinner auswählen, nach den gleichen Verfahren arbeiten, sind die Daten in allen Datenbankreplikationskopien irgendwann konsistent. Das Verfahren erklärt diejenige Revision zum Gewinner, deren Liste der Versionsgeschichten die längste ist. Wenn sie gleich lang sind, werden die `_rev`-Werte in die ASCII-Sortierreihenfolge gebracht und die höchste gewinnt. In einem Beispiel; "2-b91bb807b4685080c6a651115ff558f5" schlägt "2-5bc3c6319edf62d4c624277fdd0ae191". [47]

Konflikte können auch manuell gelöst werden. Konflikt-Dokumente werden mit dem speziellen Attribut `_conflicts` markiert. Um dieses Attribut sehen zu können, muss bei der Abfrage eines Dokuments der Parameter `conflicts=true` weitergereicht werden.

```
curl $HOST/db/doc?conflicts=true
```

Im Attribut `_conflicts` ist ein Array, wo die Versionsnummern der Verlierer-Dokumente angegeben sind.

```
{"_id":"doc","_rev":"2-123bgg24","_conflicts":["2-34g5j3h6","2-5bc3c631"]}
```

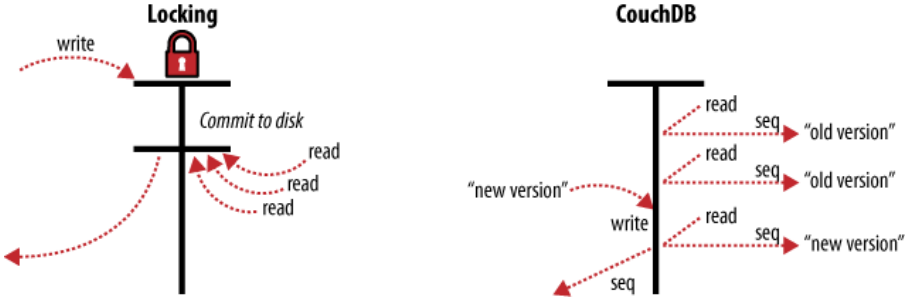
Mit diesen Versionsnummern kann man den Inhalt der Verlierer-Dokumente betrachten.

```
curl -X GET http://127.0.0.1:5984/db/doc?rev=2-456
```

Die einzige Möglichkeit ein anderes Dokument als Gewinner-Dokument zu wählen, ist das aktuelle Dokument mit dem gewünschten Inhalt zu überschreiben. Beim erneuten Replizieren wird das Konflikt-Dokument verlorengehen. [47,48]

2.8 B-Tree-Datenstruktur und MVCC

Um die Parallelität zu verwalten, verwenden die traditionellen relationalen Datenbanksysteme den sogenannten Locking-Mechanismus, der einem Benutzer den Zugriff auf die Daten versperrt, während ein anderer Benutzer die Aktualisierungen dieser Daten durchführt. Dies verhindert, dass mehrere Benutzer die Änderungen am gleichen Datensatz zum selben Zeitpunkt vornehmen. In CouchDB gibt es kein Locking, stattdessen setzt es auf eine Methode genannt Multiversion Concurrency Control (MVCC), wo jedem Benutzer ein Speicherauszug der letzten Version der Datenbank bereitgestellt wird. Dies bedeutet, dass die Änderungen von den anderen Nutzern nicht zu sehen sind, bis die Transaktion beendet ist. Die Abfragen verlaufen parallel, was die volle Leistung des Systems ausschöpft und so die Geschwindigkeit der Antwortzeiten maximiert. Will eine Reihe von Leseanfragen auf das gleiche Dokument zugreifen, werden sie zu keinem Zeitpunkt ausgesperrt. Die erste Anfrage liest ein Dokument. Während diese Anfrage sich in der Bearbeitung befindet, verändert das zweite Request dieses Dokument. Nachdem das Bearbeiten des Dokuments abgeschlossen wurde, werden die weiteren Anfragen auf das neue Dokument zugreifen, wobei die erste Anfrage die vorherige Version erhält. [49]



Die meisten modernen Datenbanken haben damit begonnen von Locking-Mechanismen zu MVCC zu wechseln, einschließlich Oracle (seit V7), MySQL (wenn es zusammen mit InnoDB verwendet wird) und Microsoft SQL Server 2005 [6].

In CouchDB ist eine Datenbank ein B-Baum und jeder B-Baum ist in einer separaten Datei gespeichert. Innerhalb dieser Datei, gibt es „zusammenhängende“ Regionen, wo die Dokumente gelagert werden. Die B-Baum-Implementierung von CouchDB unterscheidet sich von der originalen B-Baum-Definition. Während alle wichtigen Eigenschaften beibehalten sind, ist der Implementierung mit dem Multi-Version-Concurrency-Control erweitert. Zur Beschleunigung des Zugriffs auf die Dokumente gibt es zwei B-Baum-Indizien. Der eine Index `by_id` verwendet die Dokumenten-ID, um nach einem Dokument zu suchen und zeigt auf die Liste von seinen Revisionen. Damit wird die Einzigartigkeit jeder Dokumenten-ID innerhalb einer Datenbank erzwungen. Der andere Index `by_seqnum` verwendet eine monoton wachsende Zahl (Versionsnummer), die beim Aktualisieren eines Dokuments generiert wird, und verfolgt den Zeiger auf eine Replikation. In einem B-Baum werden die Daten nur in den Blattknoten gehalten. Alle Dokumenten-Updates arbeiten mit dem append-only-Mechanismus; anstatt die existierenden Dokumente zu ändern, wird eine neue Kopie in den erstellten Bereich geschrieben. Somit wächst die Datenbankdatei beim Hinzufügen von neuen Daten nur an ihrem Ende. Jede Datenbankdatei hat einen einzigen Schreibprozess und mehrere Leseprozesse. Bei der Suche nach einem Dokument wird im Dateivorsatz die Position des Wurzelknotens des B-Baums ermittelt. Anschließend erfolgt eine Traversierung `by_id`-Index (aktuelles Dokument) oder `by_seqnum`-Index (bestimmte Version des Dokuments) innerhalb dieses Baums, um den Blattknoten und damit die endgültige Position des Dokuments zu finden. Wenn eine neue Version des Dokuments erstellt wird, wird im zuletzt angelegten Dateibereich geschaut, ob das Dokument hineinpasst – falls nicht, wird die Datei um den neuen freien Bereich erweitert. Bei der Aktualisierung bzw. bei der Erstellung eines Dokuments wird der `by_id`-Baum so modifiziert, dass er einen Knoten mit dem Zeiger auf die neue Position des Dokuments beinhaltet, und in den `by_seqnum`-Baum wird ein Knoten mit der neuen Versionsnummer hinzugefügt. Wenn ein Dokument gelöscht wird, wird der `by_seqnum`-Baum zwar genauso modifiziert, allerding wird der zur Dokumenten-ID gehörige Knoten im `by_id`-Baum gelöscht. Alle Zugriffe auf die B-Baum-Knoten, sowie ihre Updates, verlaufen in $O(\log N)$, wobei N die Anzahl der Knoten ist. [50-52]

Zusammengefasst, da CouchDB die vorhandenen Daten nicht überschreibt, wird das auf der Festplatte bereits Geschriebene nie berührt. Somit kann die Datenbankdatei, z.B. bei einem Stromausfall oder fehlerhafter Hardware, nicht korrumpiert werden. Beim jeden Commit, wird nur ein Teil des Indexes, das durch das geänderte Dokument für ungültig erklärt wurde, in der Wurzel des Baums überschrieben (durch den signierten Dateivorsatz wird die Zuverlässigkeit gewährleistet).

2.9 Eventual Consistency

Das CAP-Theorem besagt, dass es für ein System zum verteilten Rechnen unmöglich ist, gleichzeitig die drei Eigenschaften Konsistenz (Consistency), Verfügbarkeit (Avalability) und Partitionstoleranz (Partition Tolerance) zu garantieren.

- Konsistenz: alle Nutzer sehen zur selben Zeit dieselben Daten.
- Verfügbarkeit: alle gestellten Anfragen werden stets beantwortet.
- Partitionstoleranz: eine Datenbank kann über mehrere Server verteilt werden.

Beim Entwurf eines solchen Systems, welches die Datenbanken über mehrere Server partitioniert, müssen wichtige Entscheidungen bezüglich der Arbeitsweise getroffen werden. Man stellt sich einer Herausforderung, sobald die Frage der Konsistenz in den Vordergrund rückt. Aggregatfunktionen, Replikationen, Dokumentenauflistungen, Map/Reduce-Abfragen – sie alle sind potentiell langlaufende Prozesse, die von der Konsistenz der Daten abhängen. CouchDB zieht Verfügbarkeit und Partitionstoleranz der Konsistenz vor. Partitionstoleranz wird durch den Mechanismus der inkrementellen Replikation erreicht. Dadurch können zwei oder mehrere Datenbanken auch über geografisch weit verteilte Standorte synchronisiert werden. Die ständige Verfügbarkeit der Daten ist dank der B-Baum-Datenstruktur möglich – eine beliebige Anzahl von Leseoperationen kann zur selben Zeit, ohne dass aufeinander gewartet wird, erfolgen. Weil die Daten nicht sofort repliziert werden und die Abfrage der Dokumente nicht unbedingt aktuell ist, ist die Konsistenz deswegen nur „schlussendlich“. [53,54]

2.10 ACID-Eigenschaften

Es gibt keine Transaktionen in CouchDB – die ACID-Eigenschaften werden von CouchDB im Sinne einer ein Dokument betreffenden Serveroperation gewährleistet. Die Operationen von CouchDB sind atomar. Während des Commits, sind die Felder und die Metadaten eines Dokuments im Puffer gepackt. Alle Dokument-Daten und die damit verbundenen Indexaktualisierungen werden synchron auf die Festplatte geschrieben und am Ende eine Datenbankdatei angehängt (append-only). Im Falle eines kritischen Serverabsturzes werden die unterbrochenen Updates beim Neustart einfach vergessen. Zusätzlich bleibt eine vorherige Kopie des Dateivorsatzes bestehen. Nur das Rollback des Dateivorsatzes ist notwendig, sodass die korrumpierten Dokumente nicht mehr zu erreichen sind. Im Falle der Abweisung einer Operation, z.B. durch eine Validierung, muss kein Rollback erfolgen, da die Daten auf die Festplatte gar nicht erst geschrieben wurden. Durch die Validierungsfunktionen kann ein einziges Dokument – ohne die Beeinträchtigung der anderen Dokumente – konsistent gehalten werden. Wenn die Validierungsfunktion umgeschrieben wurde, was zur Inkonsistenz der Daten geführt hat, können weitere Updates eventuell keinen konsistenten Datenzustand hinterlassen. Dies verletzt aber die Consistency-Eigenschaft nicht, da sie einen konsistenten Zustand vor der Transaktion voraussetzt. Alle Zugriffe auf die Dokumente werden voneinander isoliert. Die Leseoperationen verlaufen parallel – mittels MVCC kann eine beliebige Anzahl von Clients unterschiedliche und sogar dieselben Dokumente lesen, ohne von gleichzeitigen Aktualisierungen gesperrt oder unterbrochen zu werden. Für die Schreiboperationen ist ein einzelner Prozess zuständig, der sie nacheinander ausführt. Da stets nur ein Dokument beeinflusst werden kann, sind alle Operationen elementar und können nicht, wie Transaktionen, aus weiteren Teiloperationen bestehen. [12,55]

2.11 Kritik

Nachdem die wichtigsten Aspekte von CouchDB in vorhergehenden Abschnitten betrachtet wurden, ist das Auftauchen der Fragen bezüglich einiger Komponenten des Datenbanksystems wahrscheinlich. CouchDB muss zurzeit viele negative Kritiken erdulden, nachdem es zu Beginn noch für Begeisterung sorgte. [56]

Das Konzept der CouchDB-Views ist tatsächlich sehr elegant. Bedauerlicherweise kann man die Views nur aus den ursprünglichen Daten erstellen und nicht aus den anderen Views. Folglich kann man keine rekursiven Abfragen ausführen – deswegen können keine transitiven Beziehungen ausgedrückt werden. Man kann zwar so viele Map/Reduce-Funktionen definieren wie man will, allerding ist diese Lösung ineffizient. Es können keine zentralen JavaScript-Bibliotheken definiert werden, auf die innerhalb jeder Funktion jedes Design-Dokuments zugegriffen werden könnte. Bestimmte Codepassagen müssen oft wiederholt kopiert werden, was bei kleinen Änderungen große Bearbeitungszeit nach sich zieht.

Die Dokumente können miteinander auch nicht verlinkt werden. Mit Map-Funktionen ist man zwar in der Lage die Dokumente zu finden, die eine auf ein anderes Dokument beziehende ID aufweisen, doch stellt dies – sowie die eingebetteten Dokumente – nicht immer eine ausreichende Lösung dar.

Das Replikationssystem wird stark hochgelobt, doch die damit hängende Konfliktbehandlung ist eher unbefriedigend. Ein Konflikt ist kein Fehler, ohne explizite Suche nach diesen Konflikten wird die Arbeit mit den „Gewinnern“ fortgesetzt. Aus der Anwendersicht, fehlt es an der Transparenz – die benötigten Daten könnten vorhanden sein, aber werden nicht angezeigt.

Die Unterstützung der partiellen Replikation könnte ein nützlicher Start zum Aufbau verteilter Anwendungen sein. Somit eignet sich CouchDB sehr gut für die Mobile- und Desktopgeräte. Allerdings stellt Erlang, die Sprache, auf der CouchDB geschrieben ist, ein großes Problem dar. Erlang ist keine verbreitete Sprache – sie wird von wenigen solchen Geräten unterstützt.

Im Dezember 2011 erklärte Canonical, dass der Cloud-Dienst Ubuntu One zukünftig nicht mehr auf CouchDB aufsetzen wird. Die Begründung dafür waren die sich stetig verschlechternden Performance-Probleme. Bei der Anforderung von mehreren Millionen Benutzern soll CouchDB nicht ausreichend vertikal skalierbar sein und zu viele Rechnerressourcen verbrauchen. [57]

In CouchDB sind partielle Dokumenten-Updates nicht möglich. Wenn man ein Dokument aktualisiert, aktualisiert man es vollständig, es gibt nichts dazwischen. Dieses Prinzip macht zwar Sinn, so wie CouchDB funktioniert, aber den Benutzern bereitet es oft Probleme und Unbequemlichkeiten bei der Implementieren der auf CouchDB basierenden Anwendung. Es existieren einige sprachabhängige Tools, die das Update von einzelnen Attributen scheinbar erlauben, doch sie lesen lediglich das vollständige Dokument und laden die geänderte Version wieder hoch. Eine hart-codierte Lösung ist wünschenswert.

Für die Einsteiger ist es besonders hinderlich, dass es keine einheitliche Dokumentation über alle Besonderheiten von CouchDB gibt. Bestimmte Informationen muss man sich selbstständig zusammensuchen oder darauf achten, dass sie nicht veraltet sind. Bei technischen Fragen ist die einzige Möglichkeit sich an das Team per E-Mail zu wenden, weil nähere Erläuterungen zum diesen Thema kaum existieren.

CouchDB macht eine Menge Spaß. Es ist unkompliziert zu installieren und damit gleich loszulegen. Es kommt mit einer netten Web-UI, die die Arbeit mit CouchDB vereinfacht. Es benutzt REST und JSON, um eine einfache Schnittstelle bereitzustellen, welche von nahezu jeder Sprache unterstützt wird. Trotzdem gibt es genug Dinge, die an CouchDB noch verbessert oder gar überarbeitet werden sollen.

Referenzen

[1] http://nosql-database.org/
[2] http://en.wikipedia.org/wiki/Document-oriented_database
[3] http://foldoc.org/MUMPS
[4] http://en.wikipedia.org/wiki/Lotus_Notes
[5] http://wikis.gm.fh-koeln.de/wiki_db/Datenbanken/Web20
[6] http://www.ibm.com/developerworks/opensource/library/os-couchdb/index.html
[7] http://ruby.about.com/od/nosqldatabases/a/nosql1.htm
[8] http://www.golem.de/0910/70585-3.html
[9] http://www.mysql.de/why-mysql/scaleout/wikipedia.html
[10] http://wiki.apache.org/couchdb/Frequently_asked_questions
[11] http://couchdb.apache.org/
[12] http://couchdb.apache.org/docs/overview.html
[13] http://wiki.apache.org/couchdb/CouchDB_in_the_wild
[14] http://mail-archives.apache.org/mod_mbox/couchdb-dev/200910.mbox/%3C4AD53996.3090104@canonical.com%3E
[15] http://damienkatz.net/2005/02/new-couch.html
[16] http://couchdb.apache.org/downloads.html
[17] http://damienkatz.net/2008/01/new_gig.html
[18] http://damienkatz.net/2004/09/im-choosing-erlang.html
[19] http://horicky.blogspot.com/2008/06/exploring-erlang-with-mapreduce.html
[20] http://wiki.apache.org/couchdb/Frequently_asked_questions
[21] http://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xml
[22] http://damienkatz.net/2008/02/couchdb_accepte.html
[23] http://projects.apache.org/
[24] http://damienkatz.net/2012/01/the_future_of_couchdb.html
[25] http://blog.cloudant.com/the-future-of-couchdb/
[26] http://guide.couchdb.org/editions/1/en/why.html
[27] http://www.mail-archive.com/user@couchdb.apache.org/msg07478.html
[28] http://json.org/
[29] http://wiki.apache.org/couchdb/EntityRelationship
[30] http://wiki.apache.org/couchdb/HTTP_Document_API
[31] http://wiki.apache.org/couchdb/Performance
[32] http://wiki.apache.org/couchdb/Technical_Overview
[33] http://guide.couchdb.org/editions/1/en/api.html
[34] http://www.ibm.com/developerworks/webservices/library/ws-restful/
[35] http://curl.haxx.se/docs/manpage.html
[36] http://wiki.apache.org/couchdb/HTTP_database_API
[37] http://wiki.apache.org/couchdb/HttpGetUuids
[38] http://wiki.apache.org/couchdb/HTTP_Document_API
[39] http://guide.couchdb.org/editions/1/en/tour.html
[40] http://guide.couchdb.org/editions/1/en/security.html
[41] http://wiki.apache.org/couchdb/Security_Features_Overview
[42] http://guide.couchdb.org/editions/1/en/design.html
[43] http://iamseanmurphy.com/2008/09/08/couchdb-view-generation/
[44] http://wiki.apache.org/couchdb/HTTP_view_API
[45] http://guide.couchdb.org/editions/1/en/validation.html
[46] http://wiki.apache.org/couchdb/Replication
[47] http://guide.couchdb.org/editions/1/en/conflicts.html
[48] http://wiki.apache.org/couchdb/Replication_and_conflicts
[49] http://guide.couchdb.org/editions/1/en/consistency.html
[50] http://jchrisa.net/drl/_design/sofa/_list/post/post-page?startkey=%5B%22CouchDB-Implements-a-Fundamental-Algorithm%22%5D
[51] http://guide.couchdb.org/editions/1/en/btree.html
[52] http://horicky.blogspot.com/2008/10/couchdb-implementation.html
[53] http://de.wikipedia.org/wiki/CAP-Theorem
[54] http://guide.couchdb.org/editions/1/en/consistency.html
[55] http://www.mail-archive.com/user@couchdb.apache.org/msg07295.html
[56] http://planet.couchdb.org/
[57] http://www.linux-magazin.de/NEWS/Ubuntu-One-verabschiedet-sich-von-CouchDB