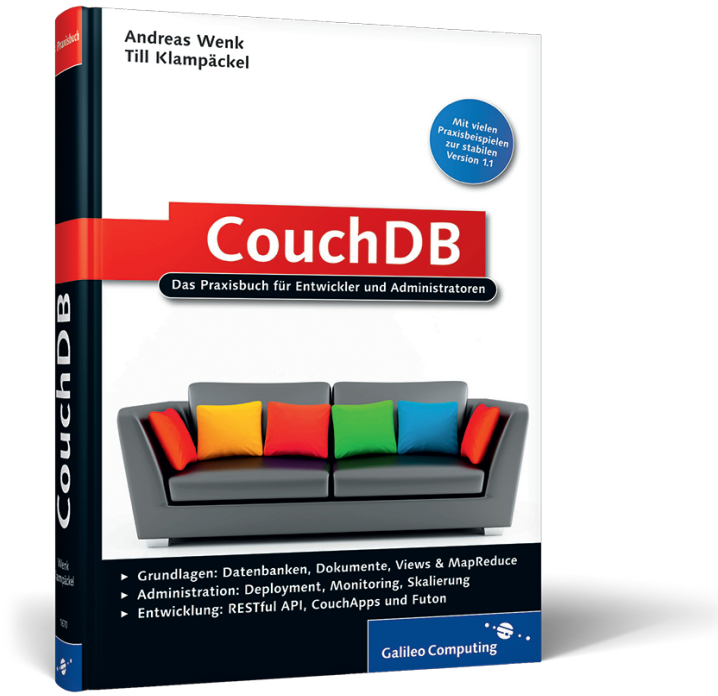
Andreas Wenk, Till Klampäckel

CouchDB

Das Praxisbuch für Entwickler und Administratoren



Galileo Press

Auf einen Blick

1 Einführung 25

2 Die Grundlagen 55

3 Praxis 1 – das Kassenbuch (zentraler CouchDB-Server) 139

4 Praxis 2 – das Kassenbuch als CouchApp 161

5 CouchDB-Administration 199

6 Bestehende Anwendungen und Libraries 259

Inhalt

Geleitwort von Jan Lehnardt 13
Geleitwort des Fachgutachters 15
Vorwort 17

1 Einführung 25

1.1 Was ist CouchDB? 25
1.1.1 Was bedeutet »dokumentbasiert«? 28
1.1.2 Was ist »RESTful«? 31
1.1.3 Was ist »MVCC«? 34
1.1.4 Was ist »ACID«? 35
1.1.5 Was ist »JSON«? 36
1.1.6 Was ist »MapReduce«? 37
1.1.7 Was ist ein »B+Tree-Index«? 40
1.1.8 Resümee 42
1.2 Das CouchDB-Open-Source-Projekt 42
1.3 Die CouchDB-Community 43
1.3.1 Der Erfinder: Damien Katz 44
1.4 Aktueller Stand und Aussichten 47
1.5 NoSQL 47
1.5.1 Der Begriff 47
1.5.2 Das CAP-Theorem 48
1.5.3 CAP im Detail 50
1.5.4 CAP und CouchDB 50
1.5.5 Wieso CAP? 51
1.5.6 Consistent Hashing und Partition Tolerance 52
1.5.7 Der Tellerrand 53

2 Die Grundlagen 55

2.1 Man spricht HTTP – CouchDBs RESTful API 56
2.1.1 Der erste Request – auf unterschiedliche Weise 56
2.1.2 Datenbanken erstellen und löschen 59
2.1.3 Dokumente erstellen, aktualisieren und löschen 59
2.2 Futon – das CouchDB-Web-Interface 59
2.2.1 Der Aufbau von Futon 60

2.2.2 Administratoren erstellen 63
2.2.3 Datenbank-Operationen in Futon 64
2.3 Datenbanken 68
2.3.1 Interna 69
2.3.2 Arbeiten mit der Datenbank 70
2.4 Dokumente 76
2.4.1 Grundlegendes zu Dokumenten 76
2.4.2 PUT /db/id – Dokument erstellen 78
2.4.3 POST db – Dokument erstellen 79
2.4.4 PUT /db/id -d JSON – Dokument erweitern 79
2.4.5 PUT /db/id -d JSON – Dokument aktualisieren 80
2.4.6 PUT /db/id/attachment – Dokument-Attachment 80
2.4.7 DELETE /db/id/attachment – Dokument-Attachment löschen 82
2.4.8 GET /db/id/attachment – Dokument-Attachment lesen 82
2.4.9 GET /db/_all_docs – alle Dokumente anzeigen 84
2.4.10 GET /db/id – ein Dokument anzeigen 86
2.4.11 DELETE /db/id – ein Dokument löschen 87
2.4.12 HEAD /db/id – Info über ein Dokument 87
2.4.13 COPY /db/id – ein Dokument kopieren 88
2.4.14 Zusammenfassung 89
2.5 Views 89
2.5.1 MapReduce und CouchDB 89
2.5.2 ETags 91
2.5.3 JavaScript 92
2.5.4 CommonJS-Unterstützung in Views 92
2.5.5 Parameter 94
2.5.6 Eingebaute Reduce-Funktionen 95
2.5.7 Temporary View 96
2.5.8 Fehleranalyse 97
2.5.9 Validieren und prüfen 97
2.5.10 Views schreiben in andere Sprachen 99
2.6 _show-Funktionen 101
2.6.1 Das _design-Dokument 101
2.6.2 Dokument für das Kassenbuch 104
2.6.3 Daten als HTML ausgeben 104
2.7 _list-Funktionen 107
2.7.1 Bereitgestellte Funktionen 108
2.7.2 Eine HTML-Liste erstellen 108

2.7.3 Zusammenfassung ... 111
2.8 URL-Rewriting ... 112
2.8.1 RewriteEngine On ... 112
2.8.2 Das ist alles? ... 115
2.9 Virtual Hosts ... 115
2.9.1 Webserver als Beispiel ... 116
2.9.2 Virtual Hosts in CouchDB ... 117
2.9.3 Rewriting und Virtual Hosts in Hochform ... 118
2.9.4 Alle Schritte zur einfachen URL ... 119
2.10 Replication ... 120
2.10.1 Continuous Replication ... 122
2.10.2 Konfliktmanagement ... 122
2.10.3 Die Datenbank »_replicator« ... 123
2.11 Sicherheit ... 125
2.11.1 Administration ... 125
2.11.2 Basic-Auth ... 126
2.11.3 Zugriffsrechte vergeben ... 128
2.11.4 Über Cookies anmelden ... 131
2.11.5 OAuth nutzen ... 133
2.11.6 Temporary Views ... 135
2.11.7 SSL und CouchDB ... 136
2.12 Proxy ... 137
2.12.1 Beispiele ... 138

3 Praxis 1 – das Kassenbuch (zentraler CouchDB-Server) ... 139

3.1 Die Kassenbuch-Applikation – kurz vorgestellt ... 140
3.1.1 Use Cases ... 140
3.1.2 Architektur der Applikation ... 141
3.1.3 Das Layout ... 145
3.1.4 Die Applikation in CouchDB speichern ... 146
3.1.5 »kassenbuch.js« näher betrachtet ... 150
3.2 Lesen – GET ... 155
3.2.1 Erzeugen der Liste mit Buchungen ... 155
3.2.2 Weitere GET-Requests ... 156
3.3 Speichern – PUT ... 158
3.4 Löschen – DELETE ... 159
3.5 Zusammenfassung ... 160

4 Praxis 2 – das Kassenbuch als CouchApp 161

4.1 Entwicklungsumgebung 161
4.1.1 Installation 162
4.1.2 couchapp-Befehle im Überblick 165
4.2 Projekt Kassenbuch 166
4.2.1 Im Detail 167
4.2.2 Hallo Welt 168
4.3 Kassenbuch – Einträge speichern und lesen 169
4.3.1 Strukturübersicht der CouchApp im _design-Dokument 170
4.3.2 Verwendete Bibliotheken 171
4.3.3 index.html 172
4.3.4 Neue Einträge speichern 174
4.3.5 Speichern 176
4.3.6 Ausprobieren 178
4.3.7 Lesen 179
4.4 Kassenbuch – Monatsansicht 182
4.4.1 Die Basis 183
4.4.2 Filter 186
4.5 Kassenbuch – Einträge löschen 190
4.5.1 Einen Button zum Löschen 190
4.5.2 DELETE 192
4.6 Kassenbuch – die Kür! 194
4.6.1 Was bleibt zu tun? 195
4.6.2 Deployment 196
4.6.3 Weitere Ideen 196

5 CouchDB-Administration 199

5.1 Installation 200
5.1.1 iOS 200
5.1.2 Android 200
5.1.3 Ubuntu 201
5.1.4 Debian Linux 206
5.1.5 Mac OS X 207
5.1.6 Couchbase-Server 216
5.2 Konfiguration 217
5.2.1 Programmatisch 222
5.2.2 Konfigurationsdateien 223
5.2.3 Point and Click 225

5.3 Deployment 225
5.3.1 Datenbankoptimierung 226
5.3.2 Statistiken und Monitoring 234
5.4 Skalierung 235
5.4.1 Caching 236
5.4.2 Sharding 239
5.4.3 Sharding a Shard 256
5.4.4 Bitte skalieren Sie weiter! 258

6 Bestehende Anwendungen und Libraries 259

6.1 PHPillow 259
6.1.1 Geschichte 260
6.1.2 Installation 260
6.1.3 Verwendung 261
6.1.4 Views 263
6.1.5 Zusammenfassung 265
6.2 Couch Potato 265
6.2.1 Projekt: Fotohosting mit Rails/CouchDB 266
6.2.2 Theorie 266
6.2.3 Los geht's 267
6.3 CouchDB-Tools 274
6.3.1 Installation 274
6.3.2 couchdb-dump 276
6.3.3 couchdb-load 277
6.3.4 couchdb-replicate 277
6.3.5 Zusammenfassung 277
6.4 jquery.couch.js 278
6.4.1 Überblick 279
6.4.2 Struktur des Plugins 280
6.4.3 Globale Methoden für den Cluster 283
6.4.4 Methoden pro Datenbank 288
6.5 Ubuntu One 296
6.5.1 Warum CouchDB? 297
6.5.2 CouchDB-Integration 297
6.5.3 Hacken 298
6.5.4 Zusammenfassung 299

Index 301

RESTful ist kein Buzz Word, sondern der gute Ton des Internets. CouchDB ist RESTful durch und durch. Die einfache, aber zugleich mächtige CouchDB API ist der Weg, mit der Datenbank zu sprechen. Diese Vokabeln und die weiteren Features von CouchDB lernen Sie in diesem Kapitel kennen.

2 Die Grundlagen

GUI, CLI, API

Jede Software will bedient werden. Dabei kann die »Bedienung« auf unterschiedliche Weise erfolgen. Zum Beispiel über ein *Graphical User Interface (GUI)* mit der Maus und der Tastatur als Eingabemedium. Oder aber über ein *Command Line Interface* – kurz: CLI –, in dem meist nur die Tastatur als Eingabemedium dient. In beiden Fällen werden Befehle abgesetzt, die das Programm anweisen, Operationen auszuführen. Die Befehle wiederum werden durch das *Application Interface – API* von den Entwicklern der Software festgelegt und dem Benutzer der Software zur Verfügung gestellt. Ob der Befehl und die darauf ausgeführte Operation nun durch einen Klick auf einen Button oder per schriftlicher Anweisung auf der Befehlseingabe gestartet wird, ist letzten Endes egal.

HTTP

Wenn der Gedanke weitergeführt wird, stellt sich im Anschluss die Frage, über welche Schnittstelle bzw. welche »Sprache« die Befehle an die Software transportiert werden. Im Fall von CouchDB ist das HTTP – die Sprache des Web. Dabei ist HTTP ein Protokoll und die Abkürzung für *Hypertext Transfer Protocol*.

Und zu guter Letzt darf auch nicht vergessen werden, dass jede Software konfiguriert werden kann, werden muss, werden sollte.

All diese angesprochenen Themen gelten natürlich auch für CouchDB. Nachdem Sie im vorigen Kapitel einen guten Überblick über einige dieser Themen gewonnen haben, gehen wir in diesem Kapitel auf diese (und weitere) Themen en détail ein. Sie werden dann mit dem nötigen Wissen gewappnet sein, um sich in den darauf folgenden Kapiteln in die Praxis stürzen zu können.

2.1 Man spricht HTTP – CouchDBs RESTful API

CouchDB ist aus dem Web für das Web. Man kann es nicht oft genug sagen. Daher spielt HTTP eine zentrale Rolle.

HTTP 1.1, RFC 2616

HTTP ist ein Protokoll, das schon seit der Geburtsstunde des Internets da ist und momentan in der Version 1.1 vorliegt. Die Bedeutung der Zahl 2616 sollte Ihnen (neben der Bedeutung der Zahl 42) als Nutzer des World Wide Web, und damit dieses Protokolls, nicht fremd sein. Genauer gesagt ermutigen wir Sie, die Spezifikation des HTTP-1.1-Protokolls, festgehalten in RFC 2616, zu verinnerlichen. Sie finden das Dokument z.B. unter *http://www.faqs.org/rfcs/rfc2616.html*.

2.1.1 Der erste Request – auf unterschiedliche Weise

RESTful

Im vorigen Abschnitt 1.1.2, »Was ist RESTful?«, haben wir den Begriff RESTful bereits erklärt. Schreiten wir jetzt zur Tat, und sprechen wir mit CouchDB. Aber Moment, wie tun wir das? Die einfachste Möglichkeit ist natürlich das Eingeben einer URL in der Adresszeile Ihres Lieblings-Browsers. Sehen wir uns das kurz an.

Davon ausgehend, dass Sie CouchDB standardmäßig lokal auf Port 5984 laufen lassen, erhalten Sie nach Eingabe der URL *http://127.0.0.1:5984* folgendes, in Abbildung 2.1 gezeigtes Ergebnis:

Abbildung 2.1 Welcome CouchDB

Sie sehen hier den einfachsten HTTP-Request an die Datenbank und die Standardantwort von CouchDB als JSON-Objekt. Allerdings können Sie auf diese Art und Weise nicht die Header des HTTP-Requests sehen.

Firefox, Chrome

Eine Möglichkeit, um diese Informationen sehen zu können, besteht darin, die Developer-Tools Ihres Browsers anzuwerfen. Google Chrome und Apple Safari bieten diese von Haus aus an, und im Mozilla Firefox installieren Sie einfach die Erweiterung FireBug, die Sie unter

http://getfirebug.com herunterladen können. Das Ergebnis in Chrome sieht aus wie in Abbildung 2.2.

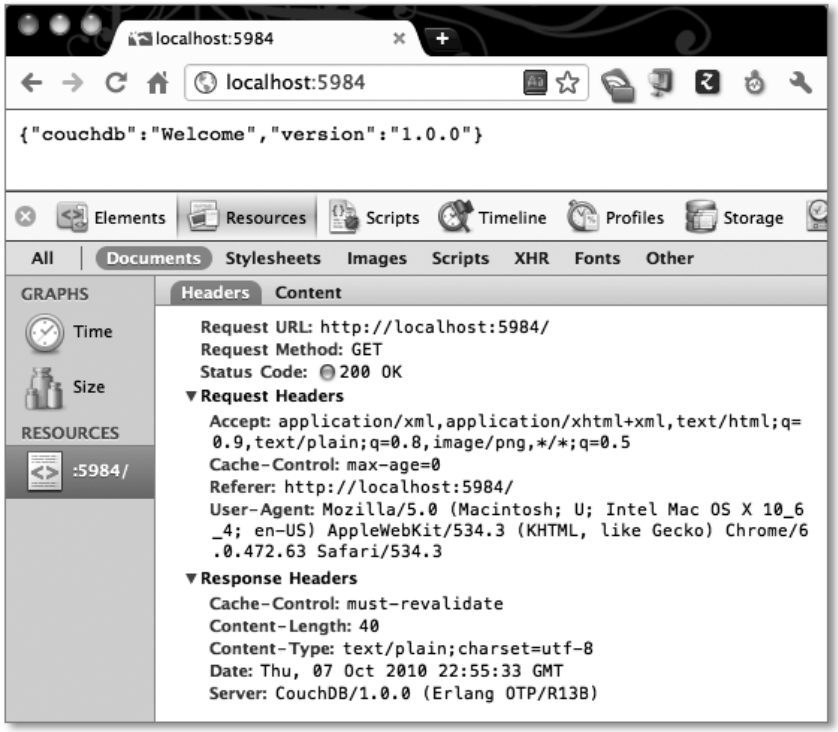


Abbildung 2.2 Welcome CouchDB mit HTTP-Header

Schließlich gibt es noch eine weitere Möglichkeit, die HTTP-Header- und Body-Informationen anzusehen – die Verwendung des Programms *cURL* auf der Kommandozeile (CLI). Die Nutzung von cURL ist ganz einfach und liefert folgendes Ergebnis:

cURL

```
$  curl -i -X GET http://127.0.0.1:5984
HTTP/1.1 200 OK
Server: CouchDB/1.0.0 (Erlang OTP/R13B)
Date: Thu, 07 Oct 2010 23:14:28 GMT
Content-Type: text/plain;charset=utf-8
Content-Length: 40
Cache-Control: must-revalidate

{"couchdb":"Welcome","version":"1.0.0"}
```

Die Ausführung des Programms cURL erwartet als Parameter mindestens eine URL. Wenn Sie durch die Option `-X` keine HTTP-Methode (`POST`, `PUT`,

HTTP-Methoden

`GET` usw.) angeben, wird standardmäßig ein `GET`-Request abgesetzt. Die Option `-i` gibt an, dass wir als Ausgabe die Header erhalten möchten. Um alle *request*- und *response*-Informationen zu erhalten, können Sie übrigens die Option `-v` (verbose, redselig) nutzen. Für weitere Optionen lesen Sie bitte die cURL-Manpage, die Sie mit `man curl` auf der Kommandozeile aufrufen.

Fehlermeldungen

Ein großer Vorteil bei der Nutzung von cURL ist die Ausgabe von Meldungen – im Besonderen Fehlermeldungen. Alle Meldungen werden als Antwort im JSON-Format von CouchDB zurückgegeben. Wenn alles gut geht, erhalten Sie meist eine Meldung der Art:

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

In der CouchDB-Version 1.1 wurden die Fehlermeldungen weiter verbessert. Nehmen wir z.B. an, dass die Konfigurationsdatei nicht dem Systembenutzer gehört, der CouchDB startet, und wir versuchen, etwas an dieser Datei zu ändern, erhalten wir eine ziemlich aussagekräftige Meldung:

```
$ cd /usr/local/etc/couchdb
$ chown root:root local.ini
$ curl -X PUT http://127.0.0.1:5984/_config/ \
  couchdb/delayed_commits -d '"false"'
{"error":"file_permission_error","reason":" \
  /usr/local/etc/couchdb/local.ini"}
```

JSON-String im Response-Body

Wie bereits erwähnt, ist der Request aus Abbildung 2.2 bzw. aus dem Quellcode auf Seite 57 der einfachste überhaupt. CouchDB antwortet mit der HTTP-Version und dem HTTP-Statuscode 200, was bedeutet, dass der Request erfolgreich war. Danach folgen weitere Informationen wie die Version des CouchDB-Servers, die Art des Inhaltes und das Encoding der Antwort. Schließlich sendet CouchDB auch einen JSON-String als HTTP-Response-Body zurück. Danach ist die Kommunikation beendet. Sie haben soeben das erste Mal mit der CouchDB gesprochen.

Ausgabe der Beispiele

Wir haben uns dazu entschieden, in den folgenden Abschnitten die Kommandozeile und das Programm cURL für alle Beispiele zu nutzen. Dabei werden wir nicht die gesamten Header ausgeben und deshalb beim `curl`-Kommando nicht immer die Option `-i` angeben.

Weil der HTTP-Response-Code allerdings in vielen Fällen sehr wichtig ist, werden wir diesen an geeigneter Stelle zusätzlich ausgeben.

Natürlich steht es Ihnen frei, die Beispiele auch auf anderem Wege als per cURL, z.B. wie in der Einführung beschrieben, nachzuvollziehen.

2.1.2 Datenbanken erstellen und löschen

Compaction

In Abschnitt 2.3, »Datenbanken«, werden wir ausführlich auf das Thema Datenbanken eingehen. Prinzipiell stehen die HTTP-Methoden `GET`, `PUT` und `DELETE` für das Aufrufen, Erstellen und Löschen von Datenbanken zur Verfügung. Außerdem haben Sie die Möglichkeit, mit der HTTP-Methode `POST` Konfigurationseinstellungen bzw. interne Operationen für eine einzelne Datenbank vorzunehmen. Zum Beispiel können Sie »Compaction« wie auf Seite 74 beschreiben so starten:

```
POST /db/_compact
```

Im weiteren Verlauf des Kapitels werden wir die wichtigsten Methoden für den Umgang mit Datenbanken zeigen.

2.1.3 Dokumente erstellen, aktualisieren und löschen

In Abschnitt 2.4, »Dokumente«, lernen Sie die RESTful API von CouchDB im Umgang mit Dokumenten kennen. Auch hier stehen Ihnen die HTTP-Methoden `GET`, `PUT`, `POST` und `DELETE` zur Verfügung. Außerdem können Sie die CouchDB-eigene Methode `COPY` nutzen, um einzelne oder mehrere Dokumente zu kopieren.

2.2 Futon – das CouchDB-Web-Interface

Administration

Im ersten Abschnitt dieses Kapitels haben Sie die RESTful API von CouchDB kennengelernt. Wie für viele Datenbanksysteme gibt es auch für CouchDB ein grafisches Administrations-Tool, das Ihnen das Leben bei der Arbeit mit CouchDB erleichtert. Dieses Tool heißt »Futon«. Es gibt zwei gravierende Unterschiede zu Administrations-Tools anderer Datenbanken.

Zum einen bekommen Sie Futon geschenkt – sprich das Tool ist nach der Installation sofort verfügbar. Ohne extra Download und Installation. Sie können es allerdings auch nicht »nicht haben wollen«. Aber das wollen Sie ja auch nicht. Tippen Sie nach der Installation von CouchDB auf dem Standardport 5984 in einen Browser Ihrer Wahl einfach *http://127.0.0.1:5984/_utils/* ein, und Sie erreichen Futon.

Ein weiterer Unterschied ist die Tatsache, dass Futon eine Web-Applikation ist. Einen Browser finden Sie in der Regel auf jedem PC (sollte das auf Ihrem Rechner nicht der Fall sein, stellt sich doch die Frage, ob Sie gerade das richtige Buch in Händen halten). Und dann steht Ihnen auch Futon zur Verfügung. Im Vergleich zu dieser Leichtigkeit sei hier die MySQL-Datenbank erwähnt. Mitgeliefert wird das MySQL-Client-Programm, das allerdings nur über die Kommandozeile genutzt werden kann. Angeboten wird als GUI der MySQL-Query-Browser. Die Nutzung bedarf allerdings einer extra Installation. Oder es steht phpMyAdmin als Webapplikation zur Verfügung – muss aber auch erst installiert werden.

Futon – out of the box

Einigen wir uns doch einfach darauf, dass »out of the box« wesentlich angenehmer und komfortabler ist.

2.2.1 Der Aufbau von Futon

Starten wir mit den Grundlagen und dem Aufbau von Futon.

iriscouch

In Abbildung 2.3 sehen Sie Futon im Zustand direkt nach der Installation. Rechts unten sehen Sie die Zeile Futon on Iris CouchDB 1.1.0. Also steht während des Schreibens dieser Zeilen die Version 1.1.0 von CouchDB auf den iriscouch-Servern zur Verfügung.

[+]

In diesem Zusammenhang beachten Sie bitte, dass Sie für die Nutzung einer CouchDB-Instanz bei iriscouch immer einen Admin anlegen müssen. Sonst können Sie z.B. den Bereich Configuration weder einsehen noch editieren.

Abbildung 2.3 Futon

Wenn Sie CouchDB lokal auf Ihrem System installieren, lesen Sie im Futon-Webinterface anstatt Futon on Iris CouchDB 1.1.0 die Zeile Futon on Apache CouchDB 1.1.0 und sehen anstatt dem Iris Couch Logo natürlich auch das Apache CouchDB Logo. [«]

Eine CouchDB bei iriscouch.com

In Abbildung 2.3 lesen Sie in der Adresszeile des Browsers die URL *http://couchbuch.iriscouch.com/* und nicht *http://127.0.0.1:5984*.

Der Grund hierfür ist ziemlich pragmatisch. Für Anschauungszwecke und Beispiele haben wir unter anderem bei *http://www.iriscouch.com* eine CouchDB-Instanz angelegt. Somit haben Sie als Leser die Möglichkeit, auf einfache und schnelle Weise in die Datenbanken, die wir hier erstellen und besprechen, hineinzusehen. Und noch besser – Sie können sich bei *http://www.iriscouch.com/* selbst eine Instanz erstellen, und zwar kostenlos: *http://www.iriscouch.com/service*!

Ein alternativer CouchDB-Hosting-Anbieter ist *http://www.cloudant.com*. Auch hier gibt es kostenlose Modelle, mit denen Sie Ihre Datenbanken hosten können. Zum Zeitpunkt der Entstehung dieser Zeilen gibt es das Produkt Cloudant 1.3.20, das auf CouchDB 1.0.2 basiert.

Tools

Auf der rechten Seite sehen Sie die Hauptnavigation. Diese ist unterteilt in zwei Bereiche: Tools und Recent Databases. Über den Punkt Tools gelangen Sie zu den wichtigsten Bereichen von Futon. Standardmäßig ist Overview ausgewählt, worunter Sie alle Datenbanken aufgelistet sehen. Zu diesem Punkt kommen wir in Kürze.

Der Punkt Configuration beinhaltet sämtliche Konfigurationseinstellungen der CouchDB. Sie können den Wert der einzelnen Optionen ändern, indem Sie das entsprechende Feld in der Spalte Value per Doppelklick öffnen. Auf Einzelheiten der Konfiguration werden wir in Abschnitt 5.2, »Konfiguration«, näher eingehen.

Replicator

Der nächste Punkt ist Replicator. Hier können Sie Datenbanken replizieren. Einzelheiten dazu finden Sie in Abschnitt siehe Abschnitt 2.10, »Replication«.

Der folgende Punkt Status dient als Übersicht über laufende Tasks wie der Fortschritt einer Replication. Mit dem Einstellen des »Poll interval« bestimmen Sie, in welchem Sekundenabstand die Anzeige aktualisiert wird.

Test Suite

Und zu guter Letzt finden Sie den Punkt Test Suite. Die dort aufgeführten Tests dienen dazu, das System und die Installation der CouchDB zu prüfen. Wenn hier Fehler auftauchen, sollten Sie die Installation überprüfen. Die Test Suite auf einer CouchDB-Cloud wie *iriscouch.com* auszuführen, macht eher weniger Sinn, da Sie keinerlei Eingriffsmöglichkeiten haben, wenn es zu Fehlern kommt.

Admin Party

Beachten Sie auch, dass vor dem Ausführen der Tests alle Admin Accounts gelöscht werden müssen, um CouchDB in den »Admin Party« – jeder darf alles machen – Zustand zu versetzen – Sie werden von Futon darauf hingewiesen und können entscheiden, ob Sie dies tun wollen.

couch.js

Ein nettes Feature der Test Suite ist die Möglichkeit, sich den Quellcode der Tests anzusehen. Dazu klicken Sie doppelt auf den Namen des Tests in der Spalte Name. Neben dem eigentlichen Testcode können Sie sehr viel über den Zugriff auf die CouchDB per JavaScript lernen. Hauptsächlich ist dabei die Datei *couch.js* von Interesse.

Abbildung 2.4 Futon – »Test Suite«

Custom Test

Außerdem dient dem fortgeschrittenen Anwender der Quellcode des Tests als Grundlage für das Schreiben eigener Tests. Um das zu tun, klicken Sie in der horizontalen Navigation über der Tabelle mit den Tests auf Custom Test. Es wird ein neues Fenster geöffnet (siehe Abbildung 2.5) in dem Ihnen bereits ein »Testgrundgerüst« zur Verfügung gestellt

wird. Erweitern Sie diesen Code, und probieren Sie ihn auch gleich aus, indem Sie den Button Run klicken.

Abbildung 2.5 Futon – »Custom Test«

Dies waren die Navigationspunkte unterhalb von Tools. Darunter finden Sie einen weiteren Navigationspunkt, Recent Databases. Dieser Punkt beinhaltet alle zuletzt betrachteten Datenbanken in Futon.

2.2.2 Administratoren erstellen

local.ini

Rechts unten in Futon wird die aktuelle Version von CouchDB angezeigt. Aktuell ist das Futon on Apache CouchDB 1.1.0. Darüber haben Sie die Möglichkeit, sich als Administrator einzuloggen oder aber neue Administratoren zu erstellen. Beachten Sie dabei, dass Sie natürlich nur als Administrator weitere Administratoren anlegen können. Da stellt sich die Frage, wie das bei einer jungfräulichen CouchDB-Installation gehen soll. Die Antwort ist einfach: Sie müssen in der Datei *local.ini* manuell den ersten Administrator eintragen.

Seit der Version 1.1 ist es möglich, den Namen eines bestehenden Administrators zu ändern. Ein kleines, aber häufig gefordertes Feature.

Für weitere Informationen zu Administratoren lesen Sie bitte Abschnitt 2.11.1, »Administration«.

2.2.3 Datenbank-Operationen in Futon

Sehen wir uns in diesem Abschnitt an, welche Datenbank-Operationen in Futon möglich sind. Wechseln Sie dazu zuallererst in den Bereich Overview.

Create Database

Die grundlegendste Operation ist sicherlich das Erstellen einer neuen Datenbank. Klicken Sie dazu auf den Punkt Create Database ..., und geben Sie in der Lightbox einen Namen ein (beachten Sie dabei die Konventionen für einen gültigen Datenbanknamen). Nennen wir die Datenbank *kassenbuch*. In Abbildung 2.6 sehen Sie die entsprechende Ansicht, nachdem Sie die Datenbank erstellt haben.

Abbildung 2.6 Futon – neue Datenbank

Über die horizontale Navigation erreichen Sie alle relevanten Operationen, um mit der Datenbank arbeiten zu können:

- Dokumente erstellen – New Document
- Admins und Readers erstellen – Security...
- Compaction – Compact & Cleanup
- die Datenbank löschen – Delete Database
- zu einem Dokument auf Grund seiner _id springen – Jump to:
- Views aufrufen – View:
- Stale Views aufrufen; d.h., dass das Ergebnis eines Views nicht aktualisiert wird, sondern ein vorher generierter View angezeigt wird.

Dokumente erstellen

New Document

Unsere Kassenbuch-Applikation braucht Dokumente. Futon hat eine aus unserer Sicht relativ komfortable Möglichkeit, neue Dokumente zu erstellen, wie Sie in Abbildung 2.7 sehen. Mit einem Klick auf den Navigationspunkt New Document öffnen Sie ein Formular, um ein Dokument

zu erstellen. Da in CouchDB jedes Dokument eine _id benötigt, müssen Sie als Erstes im Feld _id einen Wert eingeben oder den von CouchDB vorgeschlagenen MD5-Hash übernehmen.

Abbildung 2.7 Futon – neues Dokument

JSON-Ansicht

Neue Felder fügen Sie einfach über den Navigationspunkt ADD FIELD hinzu, wobei Sie zuerst den Key anlegen und dann per Doppelklick in der Spalte VALUE einen Wert eintragen. Neben jedem Formularfeld erscheint ein grünes Häkchen, um die Eingabe zu bestätigen, oder ein rotes Kreuz, um die Eingabe zu verwerfen. Sind Sie mit der Eingabe aller Keys und Values fertig, speichern Sie das Dokument mit dem Navigationspunkt SAVE DOCUMENT. In Abbildung 2.8 sehen Sie ein fertiges Dokument. Einmal in der »Futon-Ansicht« und einmal in der »JSON-Ansicht«. Beachten Sie, dass auch die Revision des Dokumentes im Feld mit dem Key `_rev` eingefügt wurde.

Abbildung 2.8 Futon – Dokumentansicht

Das Bearbeiten eines Dokumentes verhält sich wie erwartet. Sie wählen in der Ansicht Overview • kassenbuch das entsprechende Dokument aus und ändern die Werte für den Key (Field) oder Value, fügen Felder hinzu oder löschen welche.

Schnell mehrere Dokumente erstellen

Die JSON-Darstellung eines Dokumentes ist eine gute Möglichkeit, schnell mehrere gleichartige Dokumente anzulegen.

Öffnen Sie dazu ein bereits erstelltes Dokument in der Source-Ansicht, und kopieren Sie alles nach den Einträgen für `_id` und `_rev`. Erstellen Sie dann ein neues Dokument, geben Sie eine `_id` ein, oder bestätigen Sie die vorgeschlagene, und fügen Sie dann in der Source-Ansicht die Inhalte aus der Zwischenablage ein. Danach müssen Sie nur noch die Values anpassen, und schon ist ein neues Dokument erstellt.

Attachment

Über die Navigation haben Sie schließlich noch die Möglichkeit, dem Dokument über Upload Attachemnt ... ein Attachment anzufügen, und Sie können das Dokument auch über Delete Document ... löschen.

Views erstellen

Temporary View

Views in Futon zu erstellen ist nicht minder komfortabel wie das Erstellen von Dokumenten. Am einfachsten erledigen Sie diese Aufgabe, indem Sie einen *Temporary view* erstellen und diesen dann unter einem bestimmten Namen speichern. Wählen Sie im Dropdown View: also den Punkt Temporary view aus. Auf der linken Seite geben Sie den Quellcode für die Map Function ein. Auf der rechten Seite die optionale Reduce Function. Natürlich wollen Sie nach Eingabe des Quellcodes das Ergebnis sehen – und klicken deshalb auf Run. Als Ergebnis erhalten Sie eine Liste aus Key-Value-Werten.

Grouping

Die Standardansicht der Ergebnisliste ist immer ohne die Ausführung der Reduce-Funktion. Soll diese ausgeführt werden, markieren Sie die Checkbox reduce in der Spalte Value. Beim näheren Betrachten der Map-Funktion in Abbildung 2.9 werden Sie sehen, dass der Funktion `emit` als erster Parameter ein Array mit mehreren Werten übergeben wird. Als zweiten Parameter übergeben wir den Wert des Feldes Betrag aus allen Dokumenten. Dies dient der Gruppierung, denn in der Spalte Key der Ergebnisliste können Sie für das Ausführen der Reduce-Funktion das Grouping angeben, was bedeutet, dass die Beträge nach typ (level 1), typ und datum (level 2) oder typ, datum und mwst gruppiert und letztlich addiert werden. Um das Beispiel nachvollziehen zu können, legen Sie folgende Dokumente an:

```
{
   "_id": "id_1", "betrag": 43.2,
   "beschreibung": "Schreibmaterial",
   "datum": "2011-01-24", "mwst": 0.07, "typ": "ausgabe"
}

{
   "_id": "id_2", "betrag": 1680,
   "beschreibung": "Rechnung 1234",
   "datum": "2011-01-21", "mwst": 0.19, "typ": "einnahme"
}

{
   "_id": "id_3", "betrag": 55.86,
   "beschreibung": "Tanken",
   "datum": "2011-01-18", "mwst": 0.19, "typ": "ausgabe"
}

{
   "_id": "id_4", "betrag": 14,
   "beschreibung": "Bürobedarf",
   "datum": "2011-01-23", "mwst": 0.07, "typ": "ausgabe"
}

{
   "_id": "id_5", "betrag": 12.5,
   "beschreibung": "Bürobedarf",
   "datum": "2011-01-23", "mwst": 0.07,  "typ": "ausgabe"
}
```

Und hier noch die JSON-Repräsentation der MapReduce-Funktionen:

```
{
   "_id": "_design/buchhaltung",
   "_rev": "1-c220db9411023372b2b8796d830a930a",
   "language": "javascript",
   "views": {
       "auswertung": {
           "map": "function(doc){emit([doc.typ,doc.mwst, \
               doc.datum], doc.betrag);}",
           "reduce": "_sum"
       }
  }
}
```

In Futon sieht das Ergebnis aus wie in Abbildung 2.9. Nehmen Sie sich etwas Zeit, und spielen Sie mit den MapReduce-Funktionen. Es lohnt sich!

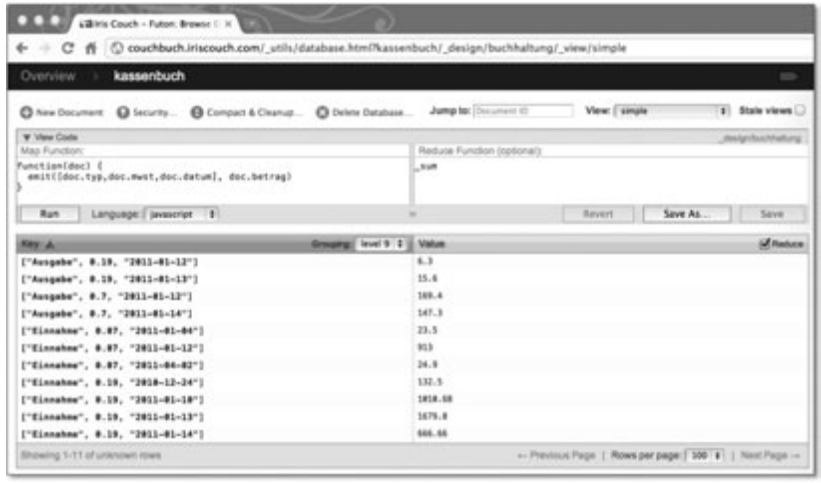


Abbildung 2.9 Futon – MapReduce

Mit diesem Beispiel wollen wir den Abschnitt über Futon abschließen. Sie haben gesehen, wie Sie die wichtigsten Datenbank-Operationen für eine CouchDB über Futon ausführen können. Ob Sie nun lieber die Kommandozeile oder Futon für das direkte »Sprechen« mit der CouchDB nutzen, liegt ganz bei Ihnen. Sie werden feststellen, dass unterschiedliche Aufgaben mit dem einen oder anderen Tool besser erledigt werden können. Im nächsten Abschnitt beschäftigen wir uns etwas genauer mit Datenbanken. Los geht's!

2.3 Datenbanken

Datenbank-Kernfunktionen

In den beiden vorangegangenen Abschnitten haben wir Ihnen die Grundlagen vermittelt, wie Sie mit einer CouchDB sprechen können. Zum einen mit Hilfe der Kommandozeile und zum anderen durch die Nutzung des Web-Interfaces »Futon«. In den jetzt folgenden Abschnitten beschäftigen wir uns eingehend mit den Kernfunktionen, die CouchDB bereitstellt. Namentlich sind das:

- Datenbanken
- Dokumente

- Views
- Show- und List-Funktionen
- Replication
- URL-Rewriting
- Virtual Hosts
- Sicherheitsaspekte

In diesem ersten Abschnitt »Datenbanken« sehen wir uns alles Wissenswerte zu CouchDB-Datenbanken an.

2.3.1 Interna

Datei .couch

Eine CouchDB-Datenbank ist letztlich nichts anderes als eine auf der Festplatte eines Rechners abgelegte binäre Datei. Diese Datei hat die Endung *.couch*. Je nachdem, mit welchem Betriebssystem Sie arbeiten und wie Sie CouchDB installiert haben (siehe Abschnitt 5.1) werden diese Dateien an unterschiedlichen Stellen abgelegt. Bei der Installation auf Ubuntu oder Mac OS X liegen diese Dateien unter */usr/local/var/lib/couchdb*.

Wenn Sie in dieses Verzeichnis sehen und auch die versteckten Ordner und Dateien anzeigen, finden Sie für jede Datenbank auch einen Ordner mit den Design-Dokumenten darin – natürlich sofern Design-Dokumente erstellt wurden. Der Ordner hat einen Namen nach der Konvention ».datenbankname_design«. Dieser enthält für alle Views (siehe Abschnitt 2.5) eine binäre Datei, in der die Key-Value-Paare aus dem Ergebnis des jeweiligen Views vorliegen. Diese Paare sind das Ergebnis der emit-Funktion aus einer Map-Funktion. Außerdem beinhaltet diese Datei auch Werte als Ergebnis einer Reduce-Funktion (siehe Abschnitt 2.5.1).

Datenbank umbenennen

Ok – die Datenbank ist also in einer Datei vorhanden. Diese Datei kann ja nun auch kopiert werden. Ja – in der Tat können Sie diese Datei einfach irgendwo anders hinkopieren – z.B. auf einen anderen Rechner, um diese CouchDB-Instanz dann dort zu nutzen. Oder Sie können diese Datei auch umbenennen. Dabei sollten Sie darauf achten, auch den Design-Dokument-Ordner umzubenennen. Oder Sie müssen alle Views einmal ausführen, denn bei diesem Vorgang wird ein neuer Design-Dokument-Ordner mit den View-Dateien erstellt.

2.3.2 Arbeiten mit der Datenbank

HTTP PUT

Was wäre der CouchDB-Server ohne Datenbanken? Genau – nutzlos! Also erstellen wir eine Datenbank – nennen wir sie »kassenbuch«. Laut Spezifikation werden Ressourcen auf einem Server per HTTP-Methode `PUT` erstellt. Daraus ergibt sich folgender Request:

```
$  curl -X PUT http://127.0.0.1:5984/kassenbuch
HTTP/1.1 201 Created

{"ok":true}
```

War der Request erfolgreich, erhalten wir als Antwort zum einen den HTTP-Status-Code 201, der so viel wie »Ressource erstellt« bedeutet, und zum anderen ein Key-Value-JSON-Objekt. Aber was passiert nun, wenn wir versuchen, eine weitere Datenbank mit demselben Namen zu erstellen? Probieren Sie es aus:

```
$  curl -X PUT http://127.0.0.1:5984/kassenbuch
HTTP/1.1 412 Precondition Failed

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

Ziemlich eindeutig. CouchDB sendet einen entsprechenden HTTP-Code und eine entsprechende Meldung.

Konventionen für die Vergabe von Datenbanknamen

Sie dürfen für den Namen einer Datenbank folgende Zeichen verwenden: a–z (nur Kleinschreibweise), 0–9, _, $, (,), +, -, /. Das erste Zeichen muss ein Buchstabe sein, und alle Sonderzeichen müssen URL-tauglich codiert werden. Eine Übersicht der Unicode-Darstellung einiger Sonderzeichen finden Sie in der Tabelle 2.1.

Zeichen	Unicode
$	%24
(	%28
)	%29
+	%2B
/	%2F

Tabelle 2.1 Unicode-Darstellung von Sonderzeichen in URL

So gesehen wäre es angebracht, einmal festzustellen, welche Datenbanken jetzt im CouchDB-Server vorhanden sind. Dafür stellt CouchDB einen speziellen Request bzw. eine spezielle URL zur Verfügung:

```
$ curl -X GET http://127.0.0.1:5984/_all_dbs
["_users","kassenbuch"]
```

Als Antwort erhalten wir eine sortierte Liste aller vorhandenen Datenbanken. Schließlich entscheiden wir uns aber, dass wir die *kassenbuch* Datenbank nicht benötigen, und löschen diese wieder. Um Ressourcen zu löschen, stellt das HTTP-Protokoll die Methode `DELETE` zur Verfügung. Also:

```
$ curl -X DELETE http://127.0.0.1:5984/kassenbuch
HTTP/1.1 200 OK

{"ok":true}
```

Funktionen der HTTP-Methoden

An diesen ersten Beispielen sehen Sie wunderbar, wie die RESTful API und die Anwendung der HTTP-Methoden im Allgemeinen funktionieren: Die URL verweist auf eine Ressource – was mit dieser geschehen soll, wird durch die HTTP-Methode bestimmt. An dieser Stelle ist wichtig zu verstehen, dass die von der CouchDB-API bereitgestellten URLs nicht mit jeder HTTP-Methode aufgerufen werden können. Sie erhalten einen Fehler, wenn Sie versuchen, *http://127.0.0.1:5984/_all_dbs* mit der HTTP-Methode `DELETE` aufzurufen.

```
$ curl -X DELETE http://127.0.0.1:5984/_all_dbs
HTTP/1.1 405 Method Not Allowed
Allow: GET,HEAD

{"error":"method_not_allowed","reason":"Only GET, \
HEAD allowed"}
```

Diese »Antwort« ist relativ einfach nachzuvollziehen, denn der Request »lösche alle vorhandenen Datenbanken« ist schlichtweg nicht erlaubt. Es gibt nun noch weitere nützliche Methoden, um Informationen über eine einzelne Datenbank zu erhalten.

Allgemeine Info zu einer Datenbank

Statistische Übersicht

Rufen Sie einfach nur die Datenbank ohne weitere Angaben auf, erhalten Sie eine statistische Übersicht. Eine solche Ausgabe könnte folgendermaßen aussehen:

```
$ curl -X GET http://127.0.0.1:5984/kassenbuch
{
  "db_name":"kassenbuch",
  "doc_count":0,
  "doc_del_count":0,
  "update_seq":0,
  "purge_seq":0,
  "compact_running":false,
  "disk_size":79,
  "instance_start_time":"1296313943608507",
  "disk_format_version":5
}
```

Als Antwort erhalten Sie ein JSON-Objekt mit einigen Informationen über die Datenbank. So zum Beispiel den Namen der Datenbank, die Anzahl der in der Datenbank vorhandenen Dokumente, wie viel Speicherplatz die Datenbank auf der Festplatte in Anspruch nimmt oder aber auch den Zeitpunkt, zu dem die Datenbank-Instanz erstellt wurde (Unix Timestamp).

_changes

Überwachung

Eine weitere nützliche Informationsquelle, besonders für das Logging bzw. Überwachungszwecke, ist der Aufruf der URI `_changes`. Die dort zu findende Funktion bietet Ihnen die Möglichkeit, Änderungen an der CouchDB-Instanz zu überwachen. Dabei können Sie der Funktion über den Request mehrere unterschiedliche Parameter übergeben. Beachten Sie dabei, dass die URL auf Grund des Encodings der Sonderzeichen in Anführungszeichen geschrieben werden muss! Beispielsweise:

```
$ curl -X GET "http://127.0.0.1:5984/kassenbuch/ \
_changes?feed=continuous&heartbeat=2000"
{"seq":1,"id":"eintrag\_1",  \
"changes":[{"rev":"1-864cb8fcc1b8c2358e6574346adfc08e"}]}
{"seq":2,"id":"ceb41f8b237b9790f9144d41f300083c", \
"changes":[{"rev":"1-967a00dff5e02add41819138abb3284d"}]}
```

tail -f

Sie sehen an der Ausgabe, dass die Datenbank *kassenbuch* bislang nur über ein Dokument verfügt. Wenn Sie nun ein neues Dokument erstellen, werden Sie dies in der obigen Ausgabe sehen. Das Gleiche gilt für das Aktualisieren eines Dokumentes. Diese Variante ähnelt einem `tail -f` unter Unix-artigen Systemen. Es bedeutet letztlich, dass die Ausgabe fortlaufend bei Änderungen aktualisiert wird und somit eingesehen werden kann.

In der nachfolgenden Tabelle finden Sie weitere hilfreiche Parameter, mit denen Sie die Ausgabe beim Aufruf von `_changes` steuern können.

Option	Beschreibung
since=n	n ist eine Zahl und stellt die Sequence-Nummer dar, nach der Änderungen angezeigt werden.
style=all_docs	Es werden detailliertere Angaben zu Revisions und Conflicts pro Ausgabezeile gemacht.
limit=n	nur n Ergebniszeilen ausgeben
feed=longpolling	Im Gegensatz zu *feed=continuous* wird beim Start eine HTTP-Verbindung geöffnet und, wenn eine Änderung auftritt, wieder geschlossen.
heartbeat=n	Alle n Millisekunden wird ein »newline-Zeichen« von CouchDB gesendet – damit kann überprüft werden, ob die Verbindung noch existiert.

Tabelle 2.2 URL-Parameter für die Steuerung von _changes

_changes-API

Die `_changes`-API lässt sich hervorragend nutzen, um in einer Applikation automatisch Prozesse anzustoßen, sobald es bestimmte Änderungen in der Datenbank gibt. Ein sehr gutes Beispiel dafür ist die Technologie hinter dem Projekt »The World's First Digital Foosball« – zu finden unter *http://digitalfoosball.com/*. Dort wird die Anzeige der gefallenen Tore durch die Überwachung des `_changes`-feeds umgesetzt.

Die Ausgabe von »_changes« filtern

Filter

Sie haben gesehen, dass der `_changes`-Feed ein sehr nützliches Tool ist, um eine CouchDB-Datenbank zu überwachen. Durch die Angabe unterschiedlicher Optionen bzw. Parameter in der URL ist bereits eine gute Filterung möglich. Damit aber nicht genug. Es besteht die Möglichkeit im `_design`-Dokument Filter mit einem beliebigen Namen zu setzen und diese dann per URL aufzurufen.

Nehmen wir an, Sie möchten nur Änderungen an Dokumenten überwachen, wenn der Betrag über 1.000 ist oder geändert wird. Im `_design`-Dokument erstellen wir einen Eintrag `filters` und fügen eine Funktion ein, die überwacht, ob die gegebene Bedingung erfüllt ist. Ist dem so, wird `true`, andernfalls `false` zurückgegeben.

```
{
   "_id": "_design/buchhaltung",
   "_rev": "261-03125704727a25aeececb9c02962c74d",
   "language": "javascript",
   "filters": {
```

```
        "big_amount": "function(doc, req) \
        { if(doc.betrag > 1000) \
        { return true; }
          else \
        { return false; }}"
    }
}
```

Der Aufruf dieses Filters geschieht dann folgendermaßen:

```
$ curl -X GET "http://127.0.0.1:5984/kassenbuch/\
  _changes?feed=continuous& \
  filter=buchhaltung/big_amount"
{"seq":6,"id":"b0006","changes":\
  [{"rev":"1-c0cc402fd08cd613863743153d639210"}]}
{"seq":432,"id":"b0002","changes":\
  [{"rev":"3-46ce3a69d46977a25f47813d70276e51"}]}
```

Das Ergebnis soll hier nur als Beispiel dienen. Im Dokument mit der ID b0006 war der Betrag bereits über 1.000, und im Dokument mit der ID b0002 wurde der Betrag auf über 1.000 erhöht.

_compact

Compaction

Von Zeit zu Zeit wird es notwendig, die Datenbankdatei (*dbname.couch*) aufzuräumen. Bei Aktualisierungen entstehen Bereiche in der Datei, die nicht weiter benötigt werden. Außerdem benötigen alle Versionen der Dokumente (_rev) der Datenbank physikalischen Speicher. Klar, dass bei vielen und großen Dokumenten im Zusammenhang mit vielen Aktualisierungen auch schnell viel Speicherplatz benötigt wird. Um nun aufräumen zu können, gibt es den »Compaction«-Mechanismus. Compaction schmeißt alte Dokumentversionen weg und entfernt nicht mehr benötigte Bereiche in der Datenbank-Datei. Danach schrumpft diese Datei erheblich. Allerdings können Sie dann auch nicht mehr auf ältere Versionen zugreifen.

Ein Beispiel:

```
$ curl -X POST http://127.0.0.1:5984/kassenbuch/_compact \
-H "Content-Type: application/json"
HTTP/1.1 202 Accepted

{"ok":true}
```

content-type-Header

Wenn Sie jetzt im Anschluss einfach den URI der Datenbank aufrufen, um die Informationen anzusehen, werden Sie einige Unterschiede erkennen. Übrigens müssen Sie den `content-type`-Header wie bei jedem

POST-Request mitsenden – obwohl Sie gar kein JSON-Objekt mitschicken. Beachten Sie auch, dass Views gesondert mit Compaction behandelt werden müssen.

Weitere Einzelheiten zu Datenbankoperationen entnehmen Sie bitte der API-Referenz.

_replicate

Replication

Dem Thema Replication haben wir ab Seite 120 einen eigenen Abschnitt gegönnt. Dieses Verfahren gehört allerdings auch zu den grundlegenden Datenbankoperationen, weshalb wir es an dieser Stelle kurz ansprechen möchten.

Die Replication ist ein Mechanismus, um auf einfache Weise eine Datenbank innerhalb eines CouchDB-Clusters oder auf einen anderen Host zu replizieren (identisch kopieren bzw. synchronisieren). Ein möglicher HTTP-Request sieht so aus:

```
$ curl -X GET http://127.0.0.1:5984/_all_dbs
["_users","kassenbuch"]

$ curl -iX POST http://127.0.0.1:5984/_replicate \
-H "content-type:application/json" \
-d '{"source": "kassenbuch", "target": "kassenbuch_copy", \
"create_target":true}'
HTTP/1.1 200 OK

{"ok":true, \
"session_id":"f75eb944bac70f40e77953f484afb64c", \
"source_last_seq":36, \
"history":[{ \
  "session_id":"f75eb944bac70f40e77953f484afb64c", \
  "start_time":"Thu, 14 Apr 2011 20:36:12 GMT", \
  "end_time":"Thu, 14 Apr 2011 20:36:12 GMT", \
  "start_last_seq":0, \
  "end_last_seq":36,
  "recorded_seq":36, \
  "missing_checked":0, \
  "missing_found":14,
  "docs_read":14,
  "docs_written":14,
  "doc_write_failures":0
  }]
}
```

_replicate Das JSON-Objekt, das wir an den URI `_replicate` übergeben, beinhaltet zum einen den Namen der zu replizierende Datenbank im Key »source«. Zum anderen im Key »target« die Zieldatenbank und außerdem den Key »create_target« mit dem Value »true«, um anzugeben, dass die Zieldatenbank vor dem Replizieren erstellt werden soll.

2.4 Dokumente

In diesem Abschnitt befassen wir uns mit CouchDB-Dokumenten. Der Umgang mit Dokumenten ist essenziell für Ihre Arbeit mit CouchDB. Deshalb gilt es, diesem Abschnitt besonderes Augenmerk zu schenken.

2.4.1 Grundlegendes zu Dokumenten

CouchDB-Dokumente werden in der Datenbankdatei *dbname.couch* abgelegt. Diese Datei haben Sie im vorigen Abschnitt bereits kennengelernt. Und wie generell alles in CouchDB liegen die Inhalte von Dokumenten als JSON-Objekte vor.

RAM Die Größe von Dokumenten ist theoretisch nur durch die Größe des zur Verfügung stehenden Festplattenspeichers begrenzt. Praktisch ist aber eher die Größe des zur Verfügung stehenden RAM (Arbeitsspeicher) ausschlaggebend. Jedes Dokument muss von CouchDB zu einem JSON-Object serialisiert und wieder deserialisiert werden. Dies geschieht komplett im RAM, was bedeutet, dass bei einem sehr großen Dokument das RAM komplett für diese Operation benötigt wird. Wenn das Dokument zu groß ist, »swapt« der Rechner. Das bedeutet, zusätzlicher Speicher wird auf der langsamen Festplatte belegt. Im Endeffekt bedeutet dies, dass der Rechner extrem langsam wird. Ein befreundeter Sysadmin hat es in diesem Fall auf den Punkt gebracht:

Viel RAM ist gut – noch mehr RAM ist besser!

Sollten Sie also mit sehr großen Dokumenten arbeiten, muss der/müssen die Rechner entsprechend von der Hardware dimensioniert sein.

Große Textdaten als Attachment

Es gibt auch die Möglichkeit, sehr große Datenmengen in Textform als Attachment zu speichern. Dabei geben Sie den Header `content-type: application/json` beim Einfügen in das Dokument an. Allerdings haben Sie keine Möglichkeit, diese Daten zu indizieren, sprich per View abzufragen, denn CouchDB reicht Attachments – egal in welcher Form – nicht an die View Engine weiter.

_id: Dokument-Name

Dokument-ID

Jedes CouchDB-Dokument muss eine eindeutige ID bzw. einen eindeutigen Namen haben. Sie haben bereits gesehen, dass Sie CouchDB anweisen können, eine ID zu erstellen (`POST`) oder aber selbst eine ID wählen können. Dabei sollten Sie beachten, dass Sie keinen Namen wählen, der mit einem »_« (Unterstrich) beginnt, da diese Namen von CouchDB selbst reserviert sind (z.B. _design). Außerdem sollten Sie keine Sonderzeichen nutzen. Als Faustregel für die Namensgebung gilt (regulärer Ausdruck in JavaScript):

```
var valid_id_name = /^[^_\W][a-zA-Z0-9-_]+$/.test(name)
```

UUID

Wenn Sie CouchDB anweisen, eine ID zu wählen, wird sich einer UUID (*Universally Unique IDentifier*, *http://tools.ietf.org/html/rfc4122*) bedient. Der URI sieht so aus:

```
$ curl -X GET http://127.0.0.1:5984/_uuids
HTTP/1.1 200 OK

"uuids":["6e5df481c02aa88e9bcd92741b0174c4"]
```

Wenn Sie an die obige URL *?count=4* anhängen, erhalten Sie anstatt einer, vier UUIDs. **[«]**

_rev: Dokument-Versionen

ETag

Jedes Dokument hat eine Version. Diese wird im Key »_rev« gespeichert. Die Version ist die MD5-Repräsentation des Dokumentes. Außerdem wird zu Anfang eine fortlaufende Nummer, gefolgt von einem »-«-Zeichen, angegeben. Dies ist sehr hilfreich, um schnell erkennen zu können, um die wie vielte Version des Dokumentes es sich handelt. Die Version wird bei vielen Operationen im *ETag-Header* gespeichert. Ein Beispiel können Sie in Abschnitt 2.4.12 sehen.

Wenn Sie ein Dokument aktualisieren (`UPDATE`) oder löschen möchten (`DELETE`) müssen Sie immer die Version angeben. Wie das geht, sehen Sie in den Beispielen ab dem folgenden Abschnitt 2.4.2.

Weitere interne Keys in einem Dokument

CouchDB-interne Keys

Jedes Dokument kann noch weitere CouchDB-interne Keys enthalten. Hier eine tabellarische Übersicht:

key	Beschreibung
`_attachments`	Speicherort für alle Attachments
`_deleted`	True, wenn das Dokument als gelöscht markiert ist. Wird durch `_compact` entfernt.
`_revisions`	Beinhaltet alle Revisionen eines Dokumentes und kann per Parameter `?revs=true` ausgegeben werden.
`_rev_infos`	Zeigt mehr Details als `_revisions`.
`_conflicts`	Info zu Konflikten
`_deleted_conflicts`	Info zu behobenen Konflikten

Tabelle 2.3 Dokument-Keys

2.4.2 PUT /db/id – Dokument erstellen

Das denkbar einfachste Dokument, das Sie in CouchDB erstellen können, sieht folgendermaßen aus:

```
{ "_id":"7b97ac5dd9873c63349453b66c003d81",
  "_rev": "1-967a00dff5e02add41819138abb3284d" }
```

Dies ist wiederum ein JSON-Objekt und enthält nur zwei Keys: `_id` und `_rev`. Halten wir also fest, dass jedes Dokument mindestens diese beiden Keys beinhalten muss. Dieses Dokument wurde mit folgenden Requests erstellt:

```
$ curl -X GET http://127.0.0.1:5984/_uuids
{"uuids":["7fd1e3606a017d9c94a3d9ef43000061"]}
$ curl -X PUT http://127.0.0.1:5984/kassenbuch/ \
7fd1e3606a017d9c94a3d9ef43000061 -d {}
HTTP/1.1 201 Created

{"ok":true,"id":"7fd1e3606a017d9c94a3d9ef43000061", \
"rev":"1-967a00dff5e02add41819138abb3284d"}
```

Alternativ können Sie wie folgt eine eigene ID angeben:

```
curl -X PUT http://127.0.0.1:5984/kassenbuch/eintrag_1 -d '{}'
{"ok":true,"id":"eintrag_1", \
"rev":"1-967a00dff5e02add41819138abb3284d"}
```

Automatische Versionsnummer

Wir haben mit den Requests folgenden Job ausgeführt: Zuerst haben wir eine eindeutige `_id` erstellt und danach auf Grundlage dieser `_id` mit der HTTP-Methode `PUT` ein Dokument in der Datenbank *kassenbuch* erstellt. Durch die Ausführung des `PUT`-Request wurde von CouchDB automatisch

eine Versionsnummer erstellt, die im Key `_rev` innerhalb des Dokuments gespeichert ist.

Es gibt noch eine zweite Möglichkeit, ein Dokument zu erstellen. Das sehen Sie im nächsten Abschnitt.

Hinweis zu den verwendeten IDs

Zugegebenermaßen sehen die in den Beispielen verwendeten IDs hässlich aus. Allerdings sieht CouchDB von Haus aus diese IDs vor, weshalb wir sie in unseren Beispielen nutzen. Es steht Ihnen auf der anderen Seite völlig frei, selbst IDs zu wählen, und das sollten Sie auch tun. Sie müssen nur darauf achten, dass Ihre IDs zum einen »unique« sind und zum anderen den Namenskonventionen entsprechen.

2.4.3 POST db – Dokument erstellen

Wie oben bereits erwähnt, bietet CouchDB neben der Möglichkeit, mit der HTTP-Request-Methode `PUT` ein Dokument zu erstellen auch die Option, die HTTP-Request-Methode `POST` zu nutzen. Der Unterschied besteht darin, dass Sie bei der Nutzung von `POST` keine »_id« vorgeben müssen. Das erledigt CouchDB für Sie. Allerdings müssen Sie den HTTP-Header `content-type: application/json` mitsenden. Der Request, mit dem ein leeres Dokument erstellt wird, sieht dann folgendermaßen aus:

```
$ curl -X POST http://127.0.0.1:5984/kassenbuch/ \
-H "Content-Type: application/json" -d '{}'
HTTP/1.1 201 Created

{"ok":true,"id":"7fd1e3606a017d9c94a3d9ef430004ce", \
"rev":"1-967a00dff5e02add41819138abb3284d"}
```

Ob Sie `PUT` oder `POST` nutzen, bleibt letztlich Ihnen überlassen. Wenn Sie die `_id` für ein Dokument nicht selbst erstellen wollen oder müssen, ist die Nutzung von `POST` wahrscheinlich einfacher. Wenn Sie die `_id` selbst erstellen, wie z.B. `eintrag_1`, `eintrag_2`, sollten Sie auf `PUT` zurückgreifen. Beachten Sie aber auf alle Fälle, dass Dokumente mit älteren Revisionen bei der Nutzung von »Compaction« (siehe Abschnitt 2.3.2, »Arbeiten mit der Datenbank«) verloren gehen.

Compaction

2.4.4 PUT /db/id -d JSON – Dokument erweitern

Zugegebenermaßen ist ein solches Dokument eher unspannend. Deshalb fügen wir diesem einen Key »name« hinzu. Um ein Dokument aktualisieren zu können, müssen Sie neben der `_id` auch die Version des Doku-

MVCC

mentes, das Sie ändern wollen und durch `_rev` definiert ist, angeben. Der Hintergund ist die Nutzung der MVCC-Architektur, wie bereits besprochen (siehe Abschnitt 1.1.3 , »Was ist MVCC?«). Der Request sieht dann so aus:

```
$ curl -X PUT http://127.0.0.1:5984/kassenbuch/ \
7fd1e3606a017d9c94a3d9ef43000061 \
-d '{"_rev": "1-967a00dff5e02add41819138abb3284d",
"typ": "Ausgabe"}'
HTTP/1.1 201 Created

{"ok":true,"id":"7fd1e3606a017d9c94a3d9ef43000061", \
"rev":"2-53f21467344e4cb88384fc9e2e189049"}
```

2.4.5 PUT /db/id -d JSON – Dokument aktualisieren

Das Dokument zu aktualisieren ist im Prinzip genau das Gleiche wie dieses zu erweitern. Sie müssen nur darauf achten, dass Sie die neueste Version übergeben. Im vorhergehenden Beispiel möchten wir jetzt aus der »Einnahme« eine »Ausgabe« machen. Der Request sieht so aus:

```
$ curl -X PUT http://127.0.0.1:5984/kassenbuch/ \
7fd1e3606a017d9c94a3d9ef43000061 \
-d '{"_rev": "2-53f21467344e4cb88384fc9e2e189049",
"typ": "Einnahme"}'
HTTP/1.1 201 Created

{"ok":true,"id":"7fd1e3606a017d9c94a3d9ef43000061", \
"rev":"3-b2e8c24d7f00dee35c659d88a3652d57"}
```

2.4.6 PUT /db/id/attachment – Dokument-Attachment

Jedes Dokument kann sogenannte *Attachments* aufnehmen. Der Key dafür ist »_attachment«. Ein Attachment kann im Prinzip jegliche Art bzw. jeglichen Content-Type haben.

»Inline-Attachments« vs. »External Attachments«

Es gibt auch die Möglichkeit, sogenannte *Inline Attachments* zu generieren. Dabei muss der Inhalt »Base64-encoded« werden. Das bedeutet, dass Sie die Daten vorher bearbeiten müssen. Wir sehen hier eher wenige Anwendungsfälle und besprechen diese Art Attachments deshalb nicht. Weitere Informationen dazu erhalten Sie im Wiki unter *http://wiki.apache.org/couchdb/HTTP_Document_API#Attachments*.

Attachment

Gehen wir davon aus, dass wir einen kleinen Text als Textdatei vorliegen haben und diesen als Attachment speichern möchten. Hier der Text:

Heute kann es regnen,
stürmen oder schneien,
denn du strahlst ja selber
wie der Sonnenschein.
Heut ist dein Geburtstag,
darum feiern wir,
alle deine Freunde
freuen sich mit dir.

UTF-8

Diese Zeilen legen wir in einer Datei namens *geburtstag.txt* ab. Im URI geben Sie dann den Typ im entsprechenden Header `content-type` an. Hier ist wichtig zu beachten, dass Sie im Content-Type das Character-Set mit angeben (`content-type: text/plain;charset=utf-8`), denn wir wollen ja, dass das »ü« in »stürmen« auch als »ü« und nicht als »Ã¼« bei der Ausgabe erscheint. Das passiert, wenn versucht wird, ein nicht richtig encodiertes UTF-8-Zeichen als UTF-8 auszugeben.

Um das Attachment im Dokument zu speichern, gehen Sie folgendermaßen vor:

```
$ curl -iX PUT http://127.0.0.1:5984/kassenbuch/ \
d1d98e6fc0a07b967c6dd82c25003e70/ \
geburtstag.txt?rev=4-2a7e6056491908d36f7b75ad75f87863 \
--data-binary @geburtstag.txt \
-H "content-type: text/plain;charset=utf-8" \
HTTP/1.1 201 Created
Server: CouchDB/1.1.0 (Erlang OTP/R14B03)
Location: http://127.0.0.1:5984/kassenbuch/ \
d1d98e6fc0a07b967c6dd82c25003e70/geburtstag.txt
Etag: "5-d5171a67e9e27aeba9beac32149a86a9"
Date: Mon, 07 Feb 2011 22:53:25 GMT
Content-Type: text/plain;charset=utf-8
Content-Length: 95
Cache-Control: must-revalidate

{"ok":true,"id":"d1d98e6fc0a07b967c6dd82c25003e70", \
"rev":"5-d5171a67e9e27aeba9beac32149a86a9"}
```

Sie können ja mal sehen, ob das auch wirklich geklappt hat, und geben die URL *http://127.0.0.1:5984/kassenbuch/d1d98e6fc0a07b967c6dd82c25003e70/geburtstag.txt* im Browser ein. Was Sie sehen, ist der Text aus der Datei *geburtstag.txt*.

Drei Varianten für den Datentyp in cURL

Wir haben im Beispiel die Option `--data-binary` genutzt. Denn wir wollen die Daten, »so wie sie sind«, an das Dokument anhängen. Sie können auch einfach `-d` nutzen, dann werden allerdings Zeilenumbrüche nicht erkannt.

Oder Sie können auch `-data-urlencode` nutzen. Der Inhalt würde URL-tauglich encodiert werden: »Heute%20kann%20es%20regnen%2C%0Ast%C3%BCrmen%20oder%20schneien ...«.

Wenn Sie z.B. ein Bild hochladen wollen, würden Sie also immer `-data-binary` nutzen, damit die Daten auf keinen Fall »bearbeitet« werden. Die Daten gehen im HTTP-Request immer in den `body`.

Ein Attachment zu aktualisieren ist dasselbe wie es neu zu erstellen.

2.4.7 DELETE /db/id/attachment – Dokument-Attachment löschen

Das Löschen eines Attachments erfolgt wieder nach dem schon bekannten Muster. Sie müssen die Version und den Namen des Attachments angeben:

```
$ curl -iX DELETE http://127.0.0.1:5984/kassenbuch/ \
d1d98e6fc0a07b967c6dd82c25003e70/ \
geburtstag.txt?rev=10-1fe78a8ede50d32b2968cfa9931f3b33 \
-H "Content-Type: appliction/json"
HTTP/1.1 200 OK
Server: CouchDB/1.1.0 (Erlang OTP/R14B03)
Date: Mon, 07 Feb 2011 23:20:32 GMT
Content-Type: text/plain;charset=utf-8
Content-Length: 96
Cache-Control: must-revalidate

{"ok":true,"id":"d1d98e6fc0a07b967c6dd82c25003e70", \
"rev":"11-d0c4cb266b9ff56b8b54c57844cba7ae"}
```

2.4.8 GET /db/id/attachment – Dokument-Attachment lesen

Wie bereits in den vorhergehenden Abschnitten angemerkt, können Sie ein Attachment mit einer `GET`-Abfrage auch wieder aus CouchDB *abholen*. Ein weiteres nettes Feature im Zusammenhang damit sind Range-Queries!

Range-Queries

Range-Queries – zu Deutsch: Bereichsabfragen – werden im Internet zum Beispiel beim Streamen von Videos und Musik eingesetzt. Auch das Anhalten und Fortsetzen von Downloads nutzt diese Eigenschaft aus.

Seit der Version 1.1.0 von CouchDB unterstützen Attachments diese Funktionalität.

Praktisch werden diese Abfragen mit Hilfe des Headers (Kopfzeile) `Range` umgesetzt. Über diesen Header teilt der Client dem Server in der Anfrage mit, welchen Teil (in Byte) einer Datei er in der Antwort lesen möchte.

Um ein Range-Query benutzen zu können, müssen Sie beim Speichern/Anlegen von Attachments immer den Header `Content-Type: application/ octet-stream` angeben. **[×]**

Beispiel

Der Einfachheit sei es geschuldet, dass wir in diesem Beispiel keine Binärdatei, sondern eine Textdatei verwenden. **[«]**

Zuerst erstellen wir eine Datei mit dem Namen *beispiel.txt*, dann erstellen wir ein neues Dokument (mit der `_id range-beispiel`) und laden den Anhang hoch:

```
$ echo "Sie lesen das CouchDB-Buch" > ./beispiel.txt
$ curl -X PUT \
http://127.0.0.1:5984/kassenbuch/range-beispiel
\ -d ' "beispiel":true'
{
  "beispiel":true,
  "id":"range-beispiel",
  "rev":"1-d82955fa0df6d6fe32ec056028848519"
}
$ curl -X PUT \
http://127.0.0.1:5984/kassenbuch/range-beispiel/beispiel.txt \
?rev="1-d82955fa0df6d6fe32ec056028848519" \
--data-binary @beispiel.txt
```

Danach führen wir einen kurzen Test aus, um zu sehen, ob die Range-Query auch wirklich unterstützt wird:

```
$ curl -I \
http://127.0.0.1:5984/kassenbuch/range-beispiel/beispiel.txt
HTTP/1.1 200 OK
Server: CouchDB/1.2.0a-3636047-git (Erlang OTP/R14B02)
ETag: "2-cde2cf75d35ec86175c976769ed9600b"
Date: Sun, 03 Jul 2011 17:51:40 GMT
Content-Type: application/octet-stream
Content-MD5: klYr4W3Y459RhQoOAxOnFA==
Content-Length: 27
Cache-Control: must-revalidate
Accept-Ranges: bytes
```

Herzlichen Glückwunsch, es ging alles gut! Sollte Ihnen beim Anlegen der Datei ein Fehler unterlaufen sein – zum Beispiel haben Sie den falschen `Content-Type` angegeben –, würde CouchDB mit `Accept-Ranges: none` antworten.

Last but not least – die Range-Query:

```
$ curl -H 'Range: bytes=14-27' \
http://127.0.0.1:5984/kassenbuch/range-beispiel/beispiel.txt
CouchDB-Buch
```

Da jeder Buchstabe in unserer Textdatei einem Byte entspricht, haben wir die ersten 14 Zeichen (`Sie lesen das`) weggelassen und nur die Antwort `CouchDB-Buch` erhalten.

[+] Die gesamte Anzahl der Bytes (in unserem Beispiel 27) erhalten Sie immer über den Header `Content-Length`.

2.4.9 GET /db/_all_docs – alle Dokumente anzeigen

Wir haben jetzt zwei Dokumente in unserer CouchDB-Datenbank *kassenbuch*. Das ist zugegebenermaßen relativ übersichtlich. Zumal die Dokumente auch nur einen konkreten Eintrag haben – nämlich den Key »typ«. Für unsere Anschauungszwecke reicht das aber völlig aus.

Im nächsten Schritt möchten wir gerne einmal alle Dokumente aus der *kassenbuch*-CouchDB als Übersicht sehen. Hier kein Problem – aber Vorsicht, wenn Sie viele Millionen Dokumente haben, kann das Anzeigen einige Zeit dauern. Ein Zugriff auf die Dokumente würde in solch einem Fall eher über Views laufen. Aber jetzt zum Request:

```
$ curl -iX GET http://127.0.0.1:5984/kassenbuch/_all_docs
HTTP/1.1 200 OK

{"total_rows":2,"offset":0,"rows":[
{"id":"7fd1e3606a017d9c94a3d9ef43000061", \
"key":"7fd1e3606a017d9c94a3d9ef43000061", \
"value":{"rev":"3-b2e8c24d7f00dee35c659d88a3652d57"}},
{"id":"7fd1e3606a017d9c94a3d9ef430004ce", \
"key":"7fd1e3606a017d9c94a3d9ef430004ce", \
"value":{"rev":"1-967a00dff5e02add41819138abb3284d"}}
]}
```

Als Ergebnis erhalten wir – wie nicht anders zu erwarten – ein JSON-Objekt mit allen relevanten Informationen zu jedem Dokument. Namentlich der ID und der Version. Diese Daten sehen Sie in der Liste ([{}, {}]) »rows«.

descending=true

CouchDB bietet unterschiedliche Möglichkeiten, um die Ausgabeliste zu manipulieren. Der am einfachsten einzusetzende Parameter ist auf jeden Fall `descending=true`:

```
$ curl -iX GET http://127.0.0.1:5984/kassenbuch/ \
  _all_docs?descending=true
HTTP/1.1 200 OK

{"total_rows":13,"offset":0,"rows":[
{"id":"b0012","key":"b0012","value": \
  {"rev":"1-213d17f1128a459670a5101a20d46cf5"}},
{"id":"b0011","key":"b0011","value": \
  {"rev":"1-213d17f1128a459670a5101a20d46cf5"}},
{"id":"b0010","key":"b0010","value": \
  {"rev":"1-3815d88459ba5aa5016115363d86b214"}},
...
{"id":"_design/buchhaltung", \
  "key":"_design/buchhaltung", \
  "value":{"rev":"35-100405fb6a5c51000a5bfdc6b85f9b87"}}
]}
```

CouchDB bietet auch die Möglichkeit, mehrere bestimmte Dokumente auszugeben. Diese werden in einem `POST`-Request als Keys in einer Liste (Array) übergeben:

```
$ curl -iX POST http://127.0.0.1:5984/kassenbuch/\
  _all_docs -d '{"keys":["b0001","b0002"]}'
HTTP/1.1 200 OK

{"total_rows":13,"offset":0,"rows":[
{"id":"b0001","key":"b0001","value": \
```

```
  {"rev":"3-d0a6522f7b8e07a9a49b332895362892"}},
{"id":"b0002","key":"b0002","value": \
  {"rev":"1-467e80017e61b506e2e856588c9946a4"}}
]}
```

Schließlich besteht auch die Möglichkeit, die Inhalte der Dokumente mit auszugeben:

```
$ curl -iX GET http://127.0.0.1:5984/kassenbuch/ \
  _all_docs?include_docs=true
...
```

Natürlich können Sie nun diverse Kombinationen aus diesen Parametern erstellen. Probieren Sie es doch mal aus.

2.4.10 GET /db/id – ein Dokument anzeigen

Für das Anzeigen eines einzelnen Dokumentes gibt es zwei Möglichkeiten – mit und ohne Version. Sie nutzen dafür die HTTP-`GET`-Methode und geben auf jeden Fall die ID des Dokumentes an:

```
$ curl -X GET http://127.0.0.1:5984/kassenbuch/ \
7fd1e3606a017d9c94a3d9ef43000061
HTTP/1.1 200 OK

{"_id":"7fd1e3606a017d9c94a3d9ef43000061", \
"_rev":"3-b2e8c24d7f00dee35c659d88a3652d57", \
"type":"einnahme"}
```

Dokument-Versionen

In diesem Fall sehen Sie das aktuellste Dokument, weil CouchDB ohne Angabe einer Version immer das aktuellste zurückgibt. Im nächsten Beispiel wollen wir, aber eine ganz bestimmte Dokumentversion sehen. Sagen wir, die zweite. Dazu geben Sie dem Request die Version als Parameter mit:

```
$ curl -X GET http://127.0.0.1:5984/kassenbuch/ \
7fd1e3606a017d9c94a3d9ef43000061 \
?rev=2-53f21467344e4cb88384fc9e2e189049
HTTP/1.1 200 OK

{"_id":"7fd1e3606a017d9c94a3d9ef43000061", \
"_rev":"2-53f21467344e4cb88384fc9e2e189049", \
"type":"ausgabe"}
```

rev=true

Wenn Sie alle vorhandenen Versionen des Dokumentes sehen möchten, geben Sie an den URI den Parameter `rev=true` mit:

```
$ curl -X GET http://127.0.0.1:5984/kassenbuch/ \
7fd1e3606a017d9c94a3d9ef43000061?revs=true
{"_id":"7fd1e3606a017d9c94a3d9ef43000061", \
"_rev":"4-a370d69068650c5bf48312ce2f9a0188", \
"type":"einnahme", "_revisions":{"start":4,\
"ids":["a370d69068650c5bf48312ce2f9a0188", \
"b2e8c24d7f00dee35c659d88a3652d57", \
"53f21467344e4cb88384fc9e2e189049", \
"967a00dff5e02add41819138abb3284d"]}}
```

Wie schon weiter oben beschrieben, können Sie auch den Parameter `revs_info=true` nutzen, um noch detailliertere Revisions-Informationen zu erhalten.

revs_info=true

2.4.11 DELETE /db/id – ein Dokument löschen

Wenn Dokumente erstellt werden können, muss es auch eine Möglichkeit geben, diese wieder zu löschen. HTTP bietet für das Löschen einer Ressource die Methode `DELETE`. Um ein Dokument löschen zu können, müssen Sie neben der ID auch die Version angeben:

DELETE

```
$ curl -iX DELETE http://127.0.0.1:5984/kassenbuch/ \
7fd1e3606a017d9c94a3d9ef430004ce \
?rev=1-967a00dff5e02add41819138abb3284d
HTTP/1.1 200 OK

{"ok":true,"id":"7fd1e3606a017d9c94a3d9ef430004ce", \
"rev":"2-eec205a9d413992850a6e32678485900"}
```

2.4.12 HEAD /db/id – Info über ein Dokument

HTTP bietet die Methode `HEAD`, um Informationen zu einer Ressource anzuzeigen. CouchDB unterstützt auch diese Methode. Sie kann hilfreich sein, um Informationen zu einem Dokument zu erhalten.

```
$ curl -X GET --head http://127.0.0.1:5984/kassenbuch/ \
7fd1e3606a017d9c94a3d9ef43000061
HTTP/1.1 200 OK
Server: CouchDB/1.1.0 (Erlang OTP/R14B03)
Etag: "4-a370d69068650c5bf48312ce2f9a0188"
Date: Thu, 03 Feb 2011 23:31:56 GMT
Content-Type: text/plain;charset=utf-8
Content-Length: 105
Cache-Control: must-revalidate
```

curl -head

Leider ist das Programm cURL an dieser Stelle etwas inkonsistent. Der Aufruf `curl -X HEAD` funktioniert so leider nicht. Deshalb die etwas andere Schreibweise wie im Beispiel. Der Parameter `-head` kann für `GET`, `PUT` und `POST` angegeben werden, um eben nur die Header der Response und nicht den Body zu erhalten.

2.4.13 COPY /db/id – ein Dokument kopieren

CouchDB bietet eine eigene HTTP-Methode namens `COPY`. Wie der Name schon sagt, können Sie mit dieser Methode Dokumente kopieren. Zum einen können Sie ein bestehendes Dokument in der aktuellsten Version zu einem neuen Dokument kopieren:

```
$ curl -X COPY http://127.0.0.1:5984/kassenbuch/ \
7fd1e3606a017d9c94a3d9ef43000061 \
-H "Destination:2f9d4949a95949ace47ef053b600072eX"
HTTP/1.1 201 Created

{"id":"2f9d4949a95949ace47ef053b600072eX", \
"rev":"1-f5f678e8784e16196c3490a0d7fd3c33"}
```

COPY

Alternativ haben Sie im Fall von `COPY` die Möglichkeit, auch von einer bestimmten Dokumentversion eine Kopie zu erstellen. Dafür geben Sie den entsprechenden Versions-Hash als Parameter an den URI:

```
$ curl -X COPY http://127.0.0.1:5984/kassenbuch/ \
7fd1e3606a017d9c94a3d9ef43000061?rev= \
3-53f21467344e4cb88384fc9e2e189049
...
```

Und schließlich haben Sie die Möglichkeit, eine Kopie eines Dokumentes in ein bereits bestehendes einzufügen. Dazu geben Sie die Version des Zieldokuments an:

```
$ curl -X COPY http://127.0.0.1:5984/kassenbuch/ \
7fd1e3606a017d9c94a3d9ef43000061 \
-H "Destination:2f9d4949a95949ace47ef053b600072eX \
?rev=1-f5f678e8784e16196c3490a0d7fd3c33"
HTTP/1.1 201 Created

{"id":"2f9d4949a95949ace47ef053b600072eX", \
"rev":"2-0590a99b8b6024a96891270751b1453f"}
```

Anhand des zurückgegebenen JSON-Objektes sehen Sie, dass eine neue Version des Zieldokumentes erstellt wurde.

Mehrere Dokumente gleichzeitig anlegen (_bulk_docs)

_bulk_docs

In bestimmten Situationen möchten Sie bestimmt mehrere Dokumente gleichzeitig anlegen. Dafür bietet CouchDB die `_bulk_docs`-Methode. Der Aufruf sieht folgendermaßen aus:

```
$ curl -X POST http://127.0.0.1:5984/kassenbuch/_bulk_docs \
  -H "content-type:application/json" \
  -d '{"docs":[{"_id":"b00b1","typ":"Ausgabe"}, \
  {"_id":"b00b2","typ":"Einnahme"}]}'
[{"id":"b00b1","rev":"1-ec700b25c2cfed526dcd03660fba8de7"}, \
{"id":"b00b2","rev":"1-7f20f0af782221eb13ff603152d552d1"}]
```

2.4.14 Zusammenfassung

In diesem Abschnitt haben Sie die wichtigsten Teile der Dokument-API von CouchDB kennengelernt. Sie sind nun in der Lage, Dokumente zu erstellen, zu aktualisieren, zu kopieren und zu löschen. Im nächsten Abschnitt, »Views«, lernen Sie alles über Abfragen in CouchDB mit MapReduce.

2.5 Views

In CouchDB werden Dokumente über ihren Key bzw. die ID gelesen.

Für komplexere Abfragen – zum Beispiel um mehrere Dokumente eines bestimmten Typs aus der Datenbank zu lesen oder Daten auszuwerten – bietet CouchDB sogenannte *Views*.

View

Die Views werden in Design-Dokumenten gespeichert und implementieren das so genannte *MapReduce*-Pattern.

MapReduce

Im Gegensatz zu relationalen Datenbanken wird der Index eines Views in CouchDB erst beim Lesen des Views aktualisiert, nicht beim Schreibvorgang in die Datenbank. Das hat den Vorteil, dass CouchDB beim Schreiben von Daten schneller ist – allerdings auf Kosten der Geschwindigkeit beim Lesen.

[«]

2.5.1 MapReduce und CouchDB

Während der Map-Phase werden grundsätzlich alle Dokumente in der Datenbank verarbeitet. Dabei ist die Reihenfolge, in der die Dokumente verarbeitet werden, nicht relevant. Relevant ist, dass die Daten verarbeitet werden.

Map

Eine einfache `map`-Funktion sieht in CouchDB wie folgt aus:

```
"_id": "_design/Beispiel",
"views": {
    "janzEinfach": {
        "map": "function (doc) {
            emit(doc._id, null);
        }"
    }
}
```

Der View wird wie folgt aufgerufen:

```
$ curl http://127.0.0.1:5984/DB/_design/ \
Beispiel/_view/janzEinfach
```

Im Allgemeinen wird bei MapReduce das Ergebnis der Map-Phase an die Reduce-Phase weitergegeben. Bei CouchDB ist dieser Schritt optional – die Reduce-Phase ist nicht zwingend notwendig.

Reduce Wenn die Ergebnisse der Map-Phase vorliegen und weiterverarbeitet werden sollen, wird in der Reduce-Phase eine Funktion auf die (Zwischen-)Ergebnisse ausgeführt.

[zB] Ein praktisches Beispiel für den Einsatz von Reduce ist das Erstellen einer Liste von verwendeten Schlagworten und ihrer Anzahl:

Die Datenbank enthält folgende Dokumente:

```
{
   "_id": "b58955b31849c35272c83b353e772fab",
   "_rev": "1-161aa57e2cffbbec2a062c946ff3a677",
   "type": "Schlagwort",
   "name": "andy"
}
{
   "_id": "da96aee30e0257a64c5afcfdb883ea2a",
   "_rev": "2-b5d608363bee2170606ffd234291706c",
   "name": "sebastian",
   "type": "Schlagwort"
}
{
   "_id": "be9163b78677a2349e41ceda9b38b183",
   "_rev": "4-778c3dc27f85857bc5fa5494b63eb479",
   "type": "Schlagwort",
   "name": "till"
}
{
```

```
   "_id": "75009b0e3183ad0fba385e284de4fa19",
   "_rev": "1-644563aa4d2787e578d05aef8b2cabcf",
   "type": "Schlagwort",
   "name": "sebastian"
}
```

Das Design-Dokument mit `map`- und `reduce`-Funktionen:

```
"_id": "_design/Schlagwort",
"views": {
    "Zaehler": {
        "map": "function (doc) {
            if (doc.type == 'Schlagwort') {
                emit(doc.name, 1);
            }
        }",
        "reduce": "function(keys, values, rereduce) {
            return sum(values);
        }"
    }
}
```

Der `curl`-Aufruf:

```
$ curl http://127.0.0.1:5984/DB/_design/ \
Schlagwort/_view/Zaehler?group=true
{"rows":[
{"key":"andy","value":1},
{"key":"sebastian","value":2},
{"key":"till","value":1}
]}
```

`group=true` sorgt dafür, dass während der Reduce-Phase gleiche Keys in den Ergebnissen reduziert werden. **[«]**

2.5.2 ETags

ETag[1]-Header werden im HTTP-Protokoll zu 90 % für das Cachen von Ressourcen verwendet. Hier eine gute Beschreibung aus Wikipedia[2]:

> *Bei der ersten Anfrage einer Ressource sendet der Server einen für diese Ressource spezifischen ETag-Wert im ETag-Header-Feld, der vom Client zusammen mit der Ressource lokal gespeichert wird. Bei einer erneuten Anfrage derselben Ressource sendet der Client in dem Header-Feld If-None-*

1 *http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.19*
2 *http://de.wikipedia.org/wiki/HTTP_ETag*

Match den zuvor gespeicherten ETag-Wert mit. Auf der Server-Seite wird nun der gesendete ETag-Wert mit dem aktuellen verglichen und bei Übereinstimmung mit dem Statuscode 304 beantwortet. Die Daten der Ressource werden in diesem Fall nicht mitgeschickt und der Client verwendet die lokal gespeicherten Daten.

View-Index

In CouchDB wird für alle Views in einem `_design`-Dokument bei jeder Änderung eines View-Index für alle Views ein neues ETag generiert. Dabei ist es egal, ob sich am jeweiligen View-Index etwas geändert hat – zum Beispiel durch die Änderung der Map- oder Reduce-Funktion. Dieses Verhalten wurde in der Version 1.1 geändert. Jetzt wird pro Anfrage eines Views ein ETag generiert, und dieses wird nur geändert, wenn sich an diesem einzelnen View-Index etwas ändert. Das hat zur Folge, dass häufiger der HTTP-Status-Code `304 Not Modified` als Antwort (Response) ausgegeben wird.

2.5.3 JavaScript

Spidermonkey

Von Haus aus werden Views in CouchDB in JavaScript geschrieben. Unter der Haube setzt CouchDB zum Ausführen dieser Funktionen das Mozilla-Projekt *Spidermonkey*[3] ein. Spidermonkey ist eine in C geschriebene JavaScript-Engine und wird unter anderem im Firefox Webbrowser verwendet.

2.5.4 CommonJS-Unterstützung in Views

Seit der CouchDB-Version 1.1 wird in Views die Nutzung von CommonJS[4]-Modulen unterstützt. CommonJS ist der Ansatz, eine einheitliche API für die Programmierung von JavaScript-gestützten Applikationen zu schaffen – egal auf welcher Plattform wie z.B. server-side, client-side oder aber auch auf dem Desktop. Auf der Community-Website finden Sie unter *http://www.commonjs.org/impl/* eine Liste mit vielen Projekten und Applikationen, die den CommonJS-Standard implementieren.

Die Nutzung eines CommonJS-Moduls erfolgt im View und unterliegt bestimmten Voraussetzungen. Hier zuerst der Ausschnitt eines Views:

3 *http://www.mozilla.org/js/spidermonkey/*
4 *http://www.commonjs.org/*

```
{
   "_id": "_design/buchhaltung",
   "_rev": "275-a3f8495cc032a09fb1dfdbaa26da0703",
   "views": {
       "lib": {
           "calculate": {
               "mwst19": "exports.mwst = function(betrag) \
                 { return [betrag, betrag * 0.19]; }",
               "mwst7": "exports.mwst = function(betrag) \
                 { return [betrag, betrag * 0.07]; }"
           }
       },
       "betrag_und_mwst19": {
           "map": "function(doc) { emit(doc.typ, require( \
             'views/lib/calculate/mwst19') \
             .mwst(doc.betrag))}"
       }
   }
}
```

CommonJS-Module

Der Geltungsbereich für CommonJS erstreckt sich auf das gesamte Design-Dokument. Deshalb besteht auch die Möglichkeit, CommonJS-Module innerhalb von List- und Show-Funktionen einzubinden. Würde nun in Views von irgendwoher ein CommonJS-Modul eingebunden werden, würde das bedeuten, den Index des Views bei jeder Änderung neu zu erstellen. Um dies zu vermeiden, haben sich die Entwickler dazu entschieden, dass CommonJS-Module für Views nur innerhalb der Views eingebunden werden dürfen. Um es genau zu sagen, müssen die Module unter `views.lib.modul_name` abgelegt werden.

Im obigen Beispiel gibt es unter `views.lib` das Modul `calculate` mit den Modulen `mwst19` und `mwst7`. Diese Module exportieren jeweils eine Methode `mwst`, und diese erwartet als Parameter einen Betrag. Bei der Ausführung des Views werden dann als Value der Gesamtbetrag und die berechnete Mehrwertsteuer zurückgegeben.

Key	Value
Ausgabe ID: b0003	[6.3, 1.197]
Ausgabe ID: b0004	[84.2, 15.998000000000001]
Ausgabe ID: b0007	[15.6, 2.964]
Ausgabe ID: b0009	[127.8, 24.282]
Ausgabe ID: b0010	[19.5, 3.705]

Tabelle 2.4 Ergebnis CommonJS-View

Key	Value
Ausgabe ID: b1111	[85.2, 16.188000000000002]
Einnahme ID: byyyyy	[23.6, 4.484]

Tabelle 2.4 Ergebnis CommonJS-View (Forts.)

2.5.5 Parameter

Bei Abfragen an den View werden verschiedene Parameter unterstützt. Diese Parameter werden zum Beispiel benutzt, um das Ergebnis des Views einzuschränken, zu sortieren oder gruppieren. Die folgenden Abschnitte haben wir nach der Methode unterteilt, wie sie auf den View zugreifen – also zum Beispiel `GET`, `HEAD` und `POST`.

Abfragen über die Methoden GET und HEAD

`GET`- und `HEAD`-Requests unterstützen folgende Parameter:

Parameter	Wert	Standardwert	Beschreibung
`key`	Key-Value	–	JSON-Datenstruktur
`startkey`	Key-Value	–	JSON-Datenstruktur
`startkey_docid`	Dokument-ID	–	**Von** – Bis: Dokument-ID
`endkey`	Key-Value	–	JSON-Datenstruktur
`endkey_docid`	Dokument-ID	–	Von – **Bis**: Dokument-ID
`limit`	Anzahl	–	Anzahl der Dokumente, die zurückgeliefert werden soll
`stale`	ok	–	`stale=ok/update_after`: ok hat zur Folge, dass der View vor der Rückgabe nicht aktualisiert wird, und `update_after` hat zur Folge, dass der View nach der Rückgabe aktualisiert wird.
`descending`	true / false	false	Ausgabe umdrehen
`skip`	Anzahl	0	Anzahl der Dokumente, die übergangen werden sollen
`group`	true	false	Automatische Reduce-Phase, um gleiche Werte aus einem Map zu filtern

Tabelle 2.5 View-Parameter

Parameter	Wert	Standardwert	Beschreibung
`group_level`	Anzahl	–	Gibt an, auf welcher Ebene das Ergebnis von Map reduziert wird
`reduce`	true / false	true	Nutze die reduce-Funktion des Views. Standardmäßig `true`, wenn eine reduce-Funktion existiert, sonst `false`
`include_docs`	true / false	false	Integriert den Inhalt aller Dokumente in der Antwort des View, auch wenn nur der Key durch die `emit`-Funktion geliefert wurde
`inclusive_end`	true / false	true	Bestimmt, ob der `endkey` im Ergebnis integriert ist.

Tabelle 2.5 View-Parameter (Forts.)

POST-Requests

```
keys=[key1, key2, key3]
```

CouchDB verhält sich in *99.999999999* % aller Fälle RESTful – d.h., dass Lesezugriffe immer über `GET`-Requests realisiert werden. Ausnahme ist der `keys`-Parameter.

Grund ist, dass Requests über `GET` eine maximale Länge von 255 Byte haben dürfen. Dieser Wert könnte durch die Angabe mehrerer Schlüssel überschritten werden.

2.5.6 Eingebaute Reduce-Funktionen

Reduce-Phase

Seit der CouchDB-Version 0.11.0 sind die drei folgenden Funktionen für die Reduce-Phase eingebaut: `_sum`, `_count` und `_stats`.

Im Gegensatz zu Funktionen in JavaScript werden diese Funktionen innerhalb von CouchDB ausgeführt, da sie in Erlang implementiert wurden.

Abhängig von den Daten (Anzahl, generelle Größe sowie Lese- und Schreibzugriffe) in der Datenbank kann der Einsatz einer in Erlang geschriebenen Funktion deutliche Geschwindigkeitsvorteile gegenüber einer Funktion in JavaScript bieten. Genauere Daten zu den Geschwin-

digkeitsvorteilen erheben Sie am besten, indem Sie eine entsprechende Benchmark für Ihr Use Case durchführen.

Beispiel in JavaScript:

```
"reduce": "function(keys, values, rereduce) {
    return sum(values);
}"
```

Das Äquivalent `_sum`:

```
"reduce": "_sum"
```

2.5.7 Temporary View

Alle bisher genannten Funktionen stehen auch als sogenannte *Temporary Views* zur Verfügung. Temporary Views sind nicht persistent, da sie nur während des Requests existieren:

- **Schritt 1:**
  View wird erstellt.
- **Schritt 2:**
  View wird ausgeführt: Die Daten werden mit MapReduce analysiert.
- **Schritt 3:**
  Ergebnis wird an den Client geliefert.
- **Schritt 4:**
  View wird gelöscht.

Wie sich aus der Schrittfolge ableiten lässt, handelt es sich dabei um weniger performante Operationen.

Temporary Views bei der Entwicklung

Der Vorteil gegenüber normalen Views ist, dass durch das Entfallen der Pflege von Design-Dokumenten Temporary Views vor allem während der Entwicklung lohnen.

[!] Temporary Views in CouchDB sind kein adäquater Ersatz für adhoc-Abfragen. Im produktiven Einsatz sollte auf Temporary Views verzichtet werden.

Hier sehen Sie ein Beispiel:

```
POST /datenbank/_temp_view  HTTP/1.0
Content-Length: 48
Date: Mon, 02 Dec 2010 00:00:00 +0200
Content-Type: application/json
```

```
{
    "map": "function(doc) {
        if (doc.hallo) {
            emit(null, doc.hallo);
        }
     }"
}
```

2.5.8 Fehleranalyse

Menschen machen Fehler – getreu dieser Devise kann es vorkommen, dass ein View eventuell nicht die Ergebnisse liefert, die erwartet werden.

- **JavaScript-Syntax:**
  Sollte es vorkommen, dass die von CouchDB gelieferten Fehlermeldungen wenig aufschlussreich sind, bietet es sich, an die Funktion(en) in eine Datei (*file.js*) zu speichern und die JavaScript-Syntax über folgenden Aufruf zu prüfen:

  ```
  $ js file.js
  ```

  Das Programm `js` wird automatisch bei der Installation von Spidermonkey mitinstalliert.

- **JSON-Syntax:**
  Die JSON-Struktur eines Dokuments kann ebenfalls mit Hilfe des `js`-Programms geprüft werden. Eine Alternative zu `js` ist die Website *http://www.jsonlint.com/*.

- **Log:**
  Angenommen, syntaktische Fehler können ausgeschlossen werden, so eignet sich die Funktion `log()` hervorragend dazu, in den View von außen reinzuschauen:

  ```
  {
      "map": "function(doc) {
          if (Bedingung) {
              log("Hallo Welt"); log(doc);
          }
      }"
  }
  ```

2.5.9 Validieren und prüfen

Seit CouchDB-Version 0.9.0 sind in Design-Dokumenten Funktionen zum Prüfen der Daten möglich.

validate_doc_update

Die Funktion wird in der Eigenschaft `validate_doc_update` des Design-Dokuments gespeichert und unterstützt drei Parameter:

- **newDocument:**
  Enthält die Daten, die hinzugefügt werden sollen.
- **currentDocument:**
  Ein optionaler Paramter – nur gesetzt, wenn bereits ein Dokument vorhanden ist.
- **userContext:**
  Der aktuelle angemeldete Benutzer und seine Rollen.

[»] Die Bezeichung der Parameter ist beliebig, nur die Reihenfolge im Kopf der Funktion ist wichtig.

```
{
    "_id": "_design/Beispiel",
    "_rev": "...",
    "views": ...,
    "validate_doc_update": "function (newDocument, \
     currentDocument, userContext) {
        ...
    }
}
```

Seperation of Concerns

Pro Design-Dokument ist es möglich, eine Funktion anzugeben. Die Funktion kann beliebig komplex sein. Es empfiehlt sich jedoch an dieser Stelle, auf SoC[5] zu achten, d.h. auf mehrere Design-Dokumente verteilte kurze Funktionen erhöhen auch in diesem Fall Lesbarkeit und Wartbarkeit des Quelltextes.

Das bedeutet »SoC«

Die Abkürzung »SoC« (»Separation of Concerns«) bedeutet, dass sich einzelne Funktionen in einem Programm so wenig wie möglich überschneiden sollen. Oft wird dies durch die Modularisierung der Funktionen innerhalb eines Programms erreicht.

[»] Bitte beachten Sie, dass beim Schreiben in die Datenbank – d.h. beim Anlegen und beim Bearbeiten jeglicher Dokumente – alle zum Prüfen angelegten Funktionen ausgeführt werden.

5 *http://en.wikipedia.org/wiki/Separation_of_concerns*

Im Detail könnte eine `validate_doc_update`-Funktion wie folgt aussehen:

```
function (newDocument, currentDocument, userContext) {
    if (!newDocument.email) {
        throw({forbidden:'Email required.'});
    }
}
```

Im Falle von Fehlern sollten diese semantisch korrekt behandelt werden.

- **Bitte einloggen:**
  `throw({unauthorized : message});`
- **Fehler aufgetreten:**
  `throw({forbidden : message});`

Wenn keine Exception aus der `validate_doc_update`-Funktion geworfen wird, geht CouchDB davon aus, dass die Eingaben richtig sind, und speichert das Dokument.

2.5.10 Views schreiben in andere Sprachen

View-Server

Wie bereits aus diesem Kapitel ersichtlich wurde, unterstützt CouchDB von Haus aus *vor allem* Views in JavaScript. Wem JavaScript nicht genügt, der hat die Möglichkeit, einen sogenannten *View-Server* in jeder beliebigen Sprache zu implementieren.

Was macht der View-Server?

Der View-Server in CouchDB kümmert sich um das Erstellen der Views in CouchDB. Egal ob die View in JavaScript geschrieben ist oder nicht, es handelt sich bei diesem Prozess immer um einen für CouchDB *externen* Prozess, mit dem CouchDB über ein textbasiertes Protokoll kommuniziert.

Andere View-Server

Zurzeit existieren neben dem View-Server in JavaScript weitere Server in Clojure, Coldfusion, Common Lisp, Erlang, Lisp, Perl, PHP, Python und Ruby. Eine aktuelle Liste findet sich im CouchDB-Wiki: *http://wiki.apache.org/couchdb/View_server#Implementations*

nodeoncouch

Node.js

CouchDB nutzt als JavaScript-Engine Mozillas SpiderMonkey. Diese Engine ist sehr gut und bewährt. Eine schnellere und neuere JavaScript-Engine ist hingegen V8 aus dem Hause Google. Wenn Sie nun die größere Perfor-

mance von V8 in CouchDB nutzen wollen, empfehlen wir Ihnen, einen Blick auf nodeoncouch zu werfen. Der Autor Meno Abels hat einen Weg gefunden, wie er der CouchDB beibringt, die V8-JavaScript-Engine von *Node.js* zu nutzen. Sie können das Git-Repository unter *https://github.com/mabels/nodeoncouch* forken oder clonen. Die Einbindung ist trivial.

View-Server einbinden

Als kleinen Bonus liefert CouchDB neben dem View-Server in JavaScript den View-Server in Erlang bereits mit. Um diesen oder einen anderen zu aktivieren, fügen Sie folgende Zeilen in der *local.ini* Ihrer CouchDB-Installation hinzu:

```
[native_query_servers]
erlang = couch_native_process, start_link, []
php = /home/till/viewserver.php
ruby = /home/andy/viewserver.rb
```

Danach ist ein Neustart von CouchDB erforderlich.

Einsatz

Ein View in Ruby würde nach Einbau des View-Servers so aussehen:

```
{
   "_id": "_design/kassenbuch",
   "_rev": "1-94bd8a0dbce5e2efd699d17acea1db0b",
   "language": "ruby",
   "views": {
     "benutzer": {
       "map": "proc { |doc| return doc \
         if doc['type'] == 'user' }"
     }
   }
}
```

Wichtig ist, dass `"language": "ruby"` angegeben wurde, damit der für Ruby konfigurierte View-Server auch verwendet wird.

[»] `"language": "javascript"` ist impliziert und muss nicht angegeben werden, wenn der View in JavaScript geschrieben wurde.

[!] Zusätzlich eingebundene View-Server werden von CouchDB nicht in einer Art Sandbox ausgeführt. Das heißt, je nachdem wie der View-Server implementiert ist, ist zum Beispiel über `_temp_view` auch das Ausführen von Befehlen auf dem Host-System (außerhalb von CouchDB) möglich.

Index

.couch-Dateien *69*
.couchapprc 167
_all_docs 84
_bulk_docs 89
_changes 72
_compact 74
_count 102
_design 67, 89
_id 77
_replicate 75
_replicator 123, 221
_rev 77
_show-Funktionen 101
_stats 102, 221
_sum 102
_view 66, 89
`_list`-Funktionen 107

A

ACID 35, 48
ACL 130
Admin-Party 126
Administrator 125
admins 222
AKID 49
Amazon Dynamo 48
Anderson, J. Chris 43
Androide 200
ASF 43
Attachments 80
attachments 217
Availability 49

B

B+Tree Index 42
B-Tree-Index 40
Backup 120, 274
BASE 48
Basic-Auth 126
Bell, Alan 42
Benutzerrollen 128
Bereichsabfrage 82
BigCouch 248
Konfiguration 254
Brewer, Eric 48
by_id_index 41
by_sequence_index 41

C

Caching 236
Cacti 225
Canonical 296
CAP-Theorem 48
Christopher Lenz 274
Cloudant 196, 248
cloudant.com 61
Cluster-Konstante 254
Codd, Edgar F. 28
CommonJS 92
Compaction 74
CONNECT 32
Consistency 49
Consistent Hashing 52, 242
Continuous Replication 277
Cookies 131
COPY 32
Couch Potato 265
Couch-Camp 18
couchapp 162
couchapp.conf 168
Couchbase-Server 216
CouchCache 237
CouchDB 1.1.0 58, 60, 82, 91, 92, 115, 118, 123, 136, 137, 217, 219, 221, 226
CouchDB, Definition von 25
CouchDB-Community 43
CouchDB-Hosting 61
CouchDB-Lounge 51, 52, 244
CouchDB-Tools 274
couchdb_httpd_auth 217
CouchDBX 208
COUNT 40
CREATE INDEX 40
Crockford, Douglas 36
cURL 20, 33

D

D-Bus 297
daemons 218
Database Compaction 226
database_dir 230
Datenbank löschen 71
Datenbankoptimierung 226
DB2 27
Debian 206
default.ini 223
delayed_commits 230
DELETE 32
Deployment 196, 225
desktopcouch 296
Dokumentbasiert 28
Dokumente 76
Domino 45
dumbproxy 244
dump.py 276

E

ElasticSearch 138
ETag 91, 236
evently 171
Eventual Consistency 50

F

Fielding, Roy 31
Foreign Key 28
FreeBSD 211
fsync 230
Futon 59, 225
Test Suite 62

G

Ganglia 225
GET 32
Gilbert, Seth 49
GIN 40
GiST 40
Google 37
Google BigTable 48
GROUP BY 40
group_level 103

H

HEAD 32
Homebrew 210
HTTP 31
HTTP 1.1 56
HTTP-Proxy 137
HTTP-Proxy-Server 237
httpd_db_handlers 219

I

IBM 45
Installation 200
iOS 200
iostat 227
Iris Associated 44
IrisCouch 196
iriscouch.com 61

J

JavaScript 92
jQuery 171
jquery.couch.js 278

K

Katz, Damien 42
Konfiguration 100, 217
Konfigurationsdateien 223

L

Lang, Alexander 265
Lehnardt, Jan 43
load.py 277
Loadbalancer 252
local.ini 117, 223
lode 51, 52
log 220
Logging 97, 220
logging 232
Lotus Notes 45
Lucene 138
Lynch, Nancy 49

M

Mac OS X 207
MacPorts 209
Magic Cookie 252
Map-Funktion 101
MapReduce 37, 68, 89, 90, 106, 181
Master-Master-Replication 120
Mobile Couchbase 201
Monitoring 225, 234
Munin 225
mustache.js 171
MVCC 34

N

Nagle-Algorithmus 219
Nginx 116, 252
nginx-lounge 246
node.js 99, 237, 238
non-blocking IO 238
Nordmann, Kore 259
NoSQL 47

O

O(log n) 41
OAuth 133, 298
OPTIONS 32
Oracle 27
ORDER BY 40
Oversharding 240

P

Paketmanager 201
Partition Tolerance 49–52
Partitionierung 50, 242
PHP 259
PHPillow 259
POST 32
PostgreSQL 27, 40
PUT 32

Q

query_server_config 220
query_servers 221
Quorum 254

R

Range 82
Range-Query 82
Ray Ozzie 44
RDBMS 27
Rebalancing 240, 244, 255
replicate.py 277
Replication 75, 221, 239, 277
RESTful 31
RESTful-API 25
Revision 77
RFC 2518 33
RFC 2616 32
RFC 4627 36
Rogers, Mikael 237
RPC 32
Ruby 100, 265

S

second write failed 35
SELECT 39
Shard-Key 241
Shard-Splitting 257
Sharding 52, 239, 240
Sicherheit 125, 298
single-point-of-failure 51
Slater, Noah 43
smartproxy 245
smartproxyd 246
Solr 138
SQL 39
SSL 136, 221
Statistik 234
stats 221
Streaming 82
Suche 138
Synaptic 201

T

Tablespace 228
TCP_NODELAY 219
Temporary View 96, 135, 233
Tools 274
TRACE 32

U

Ubuntu 201
Ubuntu One 296
URL-Rewriting 112
UUID 77, 231

V

validate_doc_update 97, 195
Varnish 239
View 89
View in Futon 66
View-Cleanup 229
View-Compaction 228
View-Parameter 94
View-Partitionierung 227
view_index_dir 217, 230
Virtual Host 115
Volltextsuche 138

W

WebDAV 33
Windows 212

X

XULRunner 203, 249

Z

Zabbix 225
ZenOSS 225
Zugriffsrechte 128