inkl. CD!

Sichere Softwareentwicklung: Der Wandel muss her S.109 S&S

Deutschland € 6,50 Österreich € 7,00 Schweiz sFr 13,40

entwickler magazin

entwickler magazin

www.entwickler-magazin.de

Januar/Februar 1.2011

CD-INHALT:

JetBRAINS Develop with pleasure! | Hudson Labs | ORACLE | The Apache Software Foundation

- **JetBrains RubyMine:** Ruby/Rails-Entwicklungsumgebung mit neuen Funktionen (z. B. überarbeiteter Ruby Code Formatter uvm.)
- **Hudson 1.385:** Ein Tool für Continuous Integration von Softwareprojekten und zur Überwachung externer Abläufe.
- **Apache CouchDB 1.0.1:** CouchDB ist eine dokumentenorientierte Datenbank und über ein RESTful JSON/HTTP-API ansprechbar.
- **MySQL 5.1.53:** Aktuellste Version der 5er-Entwicklungslinie der Open-Source-Datenbank.
- **Ruby on Rails 3 von Michael Voigt und Stefan Tennigkeit:** Buchauszug – entwickler.press
- **Video von der MobileTech Conference 2010:** Augmented Reality – Von Science Fiction zum Massenmedium

Bonus

Zusätzlich finden Sie auf unserer aktuellen Magazin-CD wieder ausgewählte Tools sowie natürlich alle Quellcodes zu den Artikeln im Heft!

▸▸▸ Alle Infos auf Seite 3

Weitere Artikel

4 194156 606509 01

Datenträger enthält Info- und Lehrprogramme gemäß § 14 JuSchG

Continuous Integration mit Hudson

Social Media API
Soziale Netzwerke 2.0

Router, Rack, Middleware und I18N
Rails 3 – Volle Fahrt voraus

CouchDB
Daten per map/reduce abfragen

...oder wie Abfragen in CouchDB erstellt werden

CouchDB – map/reduce

Relax! CouchDB ist ein prominenter Vertreter im Reigen der dokumentbasierten Datenbanken. Vor wenigen Monaten hat die Version 1.0.0 das Licht der Open-Source-Welt erblickt und aktuell liegt die Datenbank in Version 1.0.1 vor. In diesem Artikel geht es darum, die prinzipielle Technik vorzustellen, die es ermöglicht, Daten aus einer CouchDB zu lesen.

von Andreas Wenk

In der Ausgabe 5.2010 des PHP Magazins [1] habe ich einen ersten Artikel über die CouchDB geschrieben: „Entspannt auf der Couch lümmeln – Ein PHP-Wrapper für die CouchDB“. Dort wurde ein einfacher Wrapper für den Zugriff auf die CouchDB erstellt und gezeigt, wie über das RESTful-API neue Datenbanken erstellt und einzelnen Dokumente geschrieben, gelesen, aktualisiert und gelöscht werden können. Wenn Sie sich das erste Mal mit einer dokumentbasierten Datenbank wie der CouchDB beschäftigen, empfehle ich Ihnen, den Artikel vorab zu lesen, um „entspannt“ in die zweite Runde einsteigen zu können. Im vorliegenden Artikel geht es darum, wie man prinzipiell Daten per *map/reduce* von einer CouchDB-Datenbank abfragt.

Futon

CouchDB bietet von Haus aus ein webbasiertes Administrationstool namens *Futon*. Wenn CouchDB standardmäßig installiert wurde, erreicht man Futon über folgenden URL: *http://127.0.0.1:5984/_utils/* (**Abb.1**).

Mit Futon kann eine CouchDB-Instanz sehr einfach administriert werden, weshalb ich Ihnen das Tool ans Herz lege, um die Beispiele in diesem Artikel leicht verfolgen zu können. Nach einem kurzen „Zurechtfinden“ sollte es kein Problem sein, eine Datenbank, ein paar Dokumente und Views zu erstellen. Natürlich können auch weitere Features wie den Replicator und weiterführende Konfigurationseinstellungen genutzt werden. Für alle Mac-User hier noch der Hinweis auf die wohl einfachste Methode, CouchDB zu installieren: CouchDBX [2]. Jan Lehnardt von CouchOne [3] hat CouchDB in eine Mac App verpackt, um sie auf dem üblichen ein-

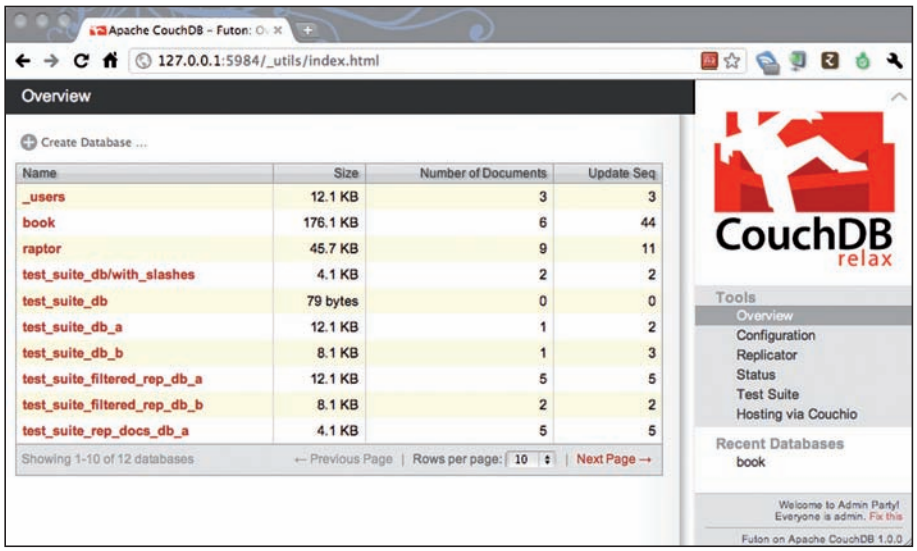


Abb. 1: CouchDB und Futon

fachen Weg installieren zu können. Sehr schön ist übrigens die Log-Ausgabe in CouchDBX.

curl

Ein weiterer Weg, mit der CouchDB zu sprechen, ist die Nutzung des Kommandozeilenprogramms *curl*. Wenn man ein unixartiges System wie Linux oder Mac OS nutzt, steht einem das Programm in der Regel von Haus aus zur Verfügung. Listing 1 zeigt in aller Kürze ein paar Beispiele, wie man mit curl und unter Nutzung der RESTful-API-Operationen an einer CouchDB-Datenbank arbeitet.

Im weiteren Verlauf des Artikels werden Futon und auch curl für die Beispiele genutzt. Für weitere Informationen ist die Lektüre des CouchDB Wiki [4] zu empfehlen – insbesondere das HTTP Database API, HTTP Document API und HTTP View API. Und natürlich wie immer die Manpage von curl.

Listing 1

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

$ curl -X GET http://127.0.0.1:5984/dinos/
{"db_name":"dinos","doc_count":4,"doc_del_count":0,"update_seq":10, \
"purge_seq":0, \
"compact_running":false,"disk_size":41049,"instance_start_time": \
"1289333879524332", \
"disk_format_version":5}

$ curl -X GET http://127.0.0.1:5984/dinos/_all_docs
{"total_rows":4,"offset":0,"rows":[ \
{"id":"_design/dinos","key":"_design/dinos","value": \
{"rev":"6-6de27b36c12f4507c81d3f1413c704fc"}}, \
{"id":"dcefca6f45c49859f34d5e91c200123c", \
"key":"dcefca6f45c49859f34d5e91c200123c", \
"value":{"rev":"1-7733f837cbc3e44a6ae92f5f102b243e"}}
]}

$ curl -X PUT http://127.0.0.1:5984/dinos/test-doc/ -d '{"name" : "test"}'
{"ok":true,"id":"test-doc","rev":"1-49ce25e3db701c8cb613c1fd18d99619"}

$ curl -X GET http://127.0.0.1:5984/dinos/test-doc
{"_id":"test-doc","_rev":"1-49ce25e3db701c8cb613c1fd18d99619","name":"test"}

$ curl -X DELETE http://127.0.0.1:5984/dinos/test-doc? \
rev=1-49ce25e3db701c8cb613c1fd18d99619
{"ok":true,"id":"test-doc","rev":"2-277a0d434467120336c479306bd814aa"}
```

Daten in Futon eingeben

Im nächsten Abschnitt werden Sie *_design*-Dokumente kennen lernen. Um dann tatsächlich mit der CouchDB und in Futon arbeiten zu können, bietet es sich an, ein paar Beispieldaten anzulegen. Dafür startet man CouchDB, öffnet Futon unter *http://127.0.0.1:5984/_utils/* und erstellt eine Datenbank (*Create Database*) mit einem Namen der eigenen Wahl – sagen wir *dinos*. Jetzt sollen mindestens drei Dokumente mit folgenden Inhalten (keys): name, age, aliment, size, weight erstellt werden. Dazu klickt man auf *New Document*. So geschehen, erstellt CouchDB den ersten Eintrag Namens *_id* und schlägt eine generierte ID als *Value* vor (**Abb. 2**).

Die ID kann getrost per Klick auf den grünen Pfeil übernommen werden. Danach fügt man die einzelnen Felder hinzu, indem *Add Field* ausgewählt und jeweils ein *Field* (key) und ein *Value* eingegeben werden. Wenn Sie meinem Vorschlag folgen, sieht das erste Dokument in Futon ähnlich wie in **Abbildung 3** aus.

Abbildung 3 zeigt übrigens bereits die gespeicherte Version – zu erkennen an dem Feld *_rev*. Wenn man auf den Reiter Source klickt, sieht man die JSON-Repräsentation des Dokuments (Kasten „Was ist JSON?“). Nach einem Klick auf *Save Document* ist das Doku-

Was is JSON?

JSON ist eine einfache Datenstruktur für den Austausch von Daten. Einfach bedeutet hier vornehmlich „einfach zu lesen für den Menschen“. Entwickelt wurde JSON von Douglas Crockford, einem der JavaScript Gurus schlechthin in der IT-Welt. Einer der größten Vorteile bei der Nutzung von JSON ist der native Zugriff auf diese Datenstruktur durch JavaScript. CouchDB speichert alle Daten (außer natürlich binäre Attachements) im JSON-Format. Wenn man mit JSON noch nicht vertraut sein sollte, hilft *http://www.json.org/json-de.html*. Prinzipiell sollte man wissen, wie die grundlegende Struktur aussieht (Listing 2).
Das Objekt beinhaltet kommaseparierte Key-Value-Paare. Als *value* kann entweder ein einzelner Wert oder ein weiteres Objekt oder ein Array (Liste) auftreten.

ment in der Datenbank gespeichert. Nun sollen noch weitere Dokumente eingefügt und gespeichert werden. Danach kann es losgehen mit den *Views*.

views und _design-Dokumente

Wenn man sich mit datenbankgestützten Anwendungen beschäftig, wird man mit allergrößter Wahrscheinlichkeit auch mit SQL-Datenbanken wie PostgreSQL, MySQL oder Oracle zu tun haben. Und wahrscheinlich ist einem dann auch geläufig, was eine *View* ist. Kurze Gedächtnisstütze: Eine View ist eine fest definierte Abfrage, die als „Sicht" auf eine oder mehrere Tabellen in der Datenbank dient und durch eine virtuelle Tabelle repräsentiert wird. In RDBMS (relationalen Datenbankmanagement-Systemen) hat die Nutzung von Views verschiedene Vorteile:

- Verstecken von komplexen Datenstrukturen
- Sicherheitsaspekte: Benutzer haben das Recht, die View zu sehen, nicht aber das Recht, die Tabellen, auf die die View zugreift, anzusehen oder zu verändern

Klar, dass eine View in SQL geschrieben ist. In CouchDB sind Views ebenfalls Sichten auf Daten. Dabei gibt es ein paar Unterschiede. Zum einen werden Views in besonderen Dokumenten gespeichert: in *_design*-Dokumenten. Dann ist die Sprache, in der eine View erstellt wird – sozusagen das Pendant zu SQL – JavaScript. Und zu guter Letzt folgen alle Views *Map/Reduce* (Kasten „Was ist Map/Reduce?"). Aber eins nach dem anderen.

Das denkbar einfachste Dokument in CouchDB hat die JSON-Repräsentation wie in Listing 3. Zu beachten ist, dass die *_id* immer eindeutig sein muss. Außerdem erhält jedes Dokument eine Revision. Ein *_design*-Dokument folgt dem gleichen Prinzip, hat aber ein paar eindeutige Merkmale (Listing 4).

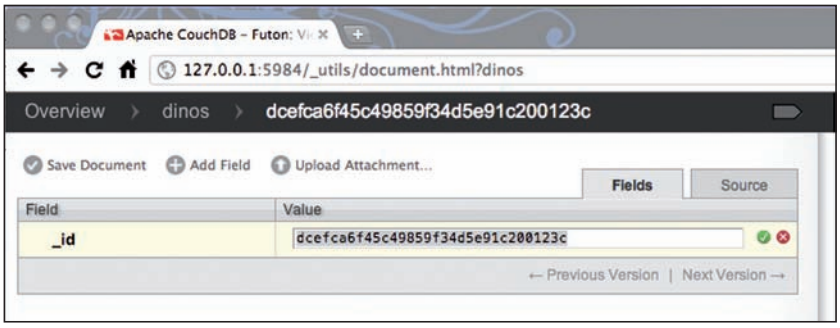


Abb. 2: CouchDB erstellt den Eintrag namens _id

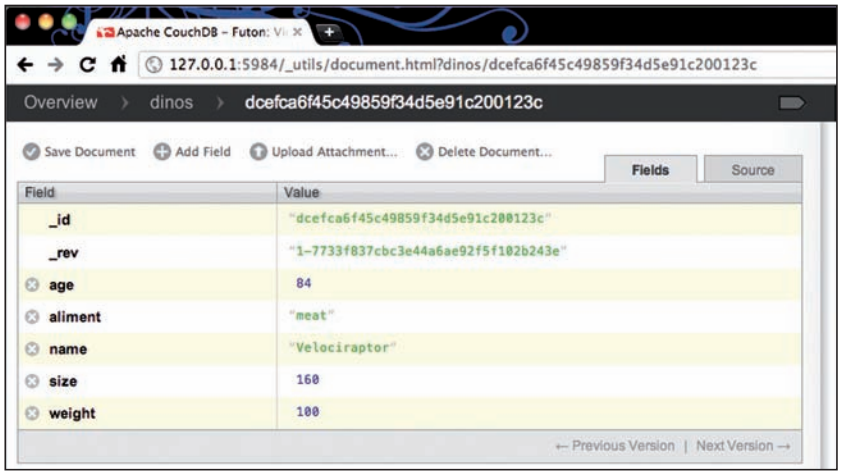


Abb. 3: Das erste Dokument in Futon

In Listing 4 ist unschwer zu erkennen, dass es sich wieder um eine relativ einfache JSON-Struktur handelt. Dass es sich um ein *_design*-Dokument handelt, lässt sich schnell über die *_id* identifizieren. Die *_id* beginnt immer mit *_design/*, gefolgt vom Name des *_design*-Dokuments, wobei der Name eine von CouchDB generierte ID sein kann oder ein selbst vergebener (innerhalb der Datenbank) eindeutiger Name.

Was ist Map/Reduce?

In CouchDB wird auf die Dokumente bzw. deren Daten per *Map / Reduce* zugegriffen. Dabei wird zuerst eine *Map*-Funktion ausgeführt, um ein Set von Daten aus der Datenbank zu erhalten. Das geschieht per JavaScript. Einer solchen Funktion sollte sinnvollerweise immer das *doc*-Objekt übergeben werden, um auf einzelne Elemente bzw. Felder des Dokuments zugreifen zu können (doc.schubbidu, doc.fallera). In einem weiteren Schritt können mit einer *Reduce*-Funktion Aggregationen aus dem Ergebnis Set der *Map*-Funktion erstellt werden. Im einfachsten Fall sind das Summen per *sum()*. Die *Reduce*-Funktion sollte immer nur einen Wert zurückliefern.

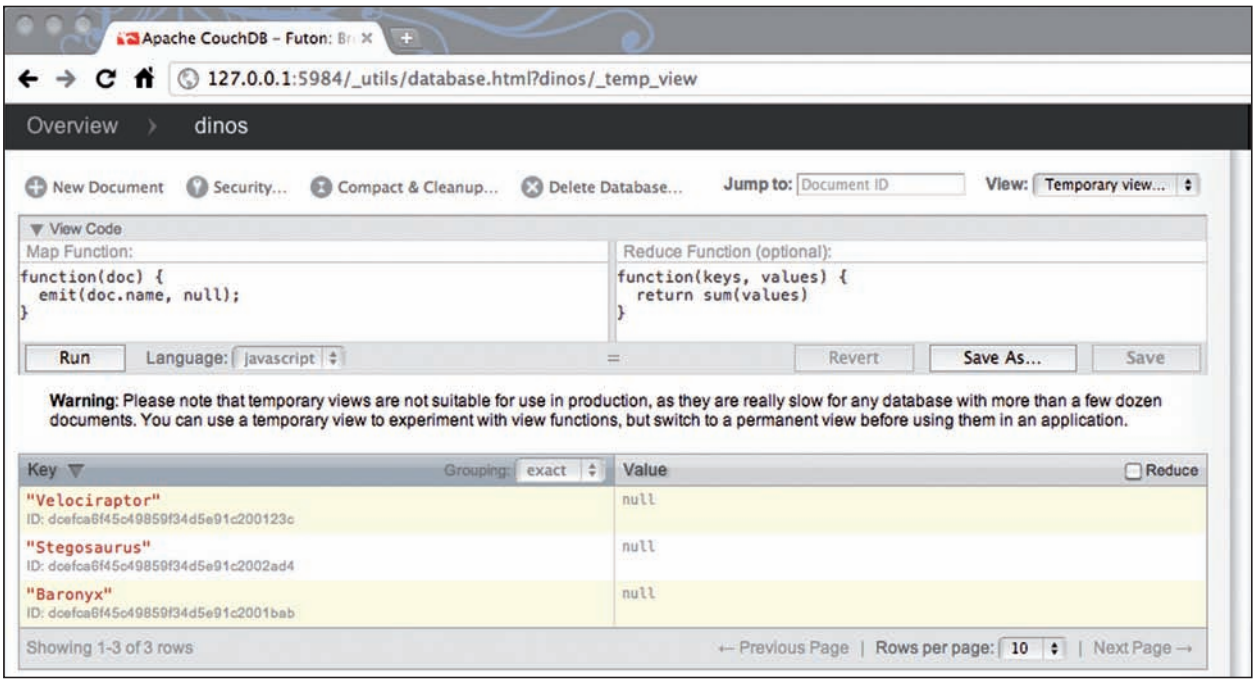
Listing 2

```
{"tools" : ["knife", "spoon", "fork"],
 "kitchen" : 1,
 "cook" : "Bocuse",
 "guests" : {
   "Belmondo" : 2,
   "Reno" : 4
 }
}
```

Listing 3

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

Abb. 4: Einen View erstellen



Der nächste Eintrag hat den Key *_rev*. Dies stellt die Revision des Dokuments dar. Diese Revision ist der 32-stellige MD5 Hash des Dokuments, angeführt von einer Nummer, die angibt, um welche laufende Nummer des Dokuments es sich handelt.

Weiter geht es mit dem key *views*. Hier spielt die Musik. Wie der Key schon vermuten lässt, werden hier die einzelnen Views dieses *_design*-Dokuments gespeichert. Im obigen Beispiel gibt es nur eine View namens *all_dinos*. Innerhalb dieser View ist dann eine *map*- und eine *reduce*-Funktion abgelegt. Dabei ist zu beachten, dass die *reduce* Funktion optional ist. Wenn aber eine *reduce*-Funktion erstellt wurde – auch wenn sie kein Ergebnis liefert – wird sie standardmäßig ausgeführt. Im Fall, dass man einfach das gesamte Ergebnis der *map*-Funktion erhalten will, lässt man die *reduce*-Funktion weg und setzt folgenden URL ab: *curl -X GET http://127.0.0.1:5984/dinos/_design/dinos/_view/all_dinos*

Listing 4

```
{
  "_id": "_design/dinos",
  "_rev": "1-285870a61b252bcb3a971a5750d36d93",
  "views": {
     "all_dinos": {
        "map": "function(doc) {
    emit(doc.name, null)
   }",
        "reduce": "function(keys, values) {
    return sum(values)
   }"
     }
  }
}
```

Wenn das obige Beispiel ausgeführt werden soll, aber die *reduce*-Funktion nicht, ergänzt man zum obigen URL den Parameter *reduce=false*:

```
curl -X GET http://127.0.0.1:5984/dinos/_design/dinos/_view/all_dinos? \
reduce=false
```

Ich komme hierauf später noch einmal zurück. Für den Moment soll das als Erklärung ausreichen.

Eine View erstellen

Bevor es weitergeht, sollte man wissen, wie in Futon eine View erstellt wird. Voraussetzung dafür sind grundlegende JavaScript-Kenntnisse. Am einfachsten erstellt man ein *_design*-Dokument, indem in der Übersicht der Datenbank rechts im Drop-Down-Menü der Eintrag *Temporary View* ausgewählt wird. In dieser Ansicht befindet sich auf der linken Seite ein Eingabefeld für die *Map*-Funktion und auf der rechten das Eingabefeld für die *Reduce*-Funktion. Hier gibt man im jeweiligen Feld die JavaScript-Funktion aus dem obigen JSON-Objekt ein. Danach führt man die View aus, indem man auf *Run* klickt. Das Ergebnis wird je nach den eingegebenen Daten und je nach Anzahl der Dokumente aussehen wie in **Abbildung 4**. Dieses *_design*-Dokument sollte nun per *Save As …* gespeichert werden. Zu einem späteren Zeitpunkt kann das Dokument wieder über das View-Drop-Down-Menü ausgewählt, bearbeitet und ggf. unter einem neuen Namen gespeichert werden.

Map-Funktionen

Nun soll die *Map*-Funktion etwas näher betrachtet werden. Der Quellcode dafür ist in Listing 5 zu finden.

Das ist eine ziemlich simple JavaScript-Funktion, hat aber bereits einige CouchDB-spezifische Eigenschaften. Zuerst sollte jeder *Map*-Funktion immer ein Dokument übergeben werden. Gewöhnlich wird der Funktion ein

Parameter Namens *doc* übergeben. Da es sich um ein JavaScript-Objekt handelt (im JSON-Format bzw. kurz ein JSON-Objekt), greift man auf die einzelnen Objekteigenschaften in der Notation *object.name* zu. Also in unseren Dino-Dokumenten z. B. *doc.name*, *doc.age* usw. Um Daten als Ergebnis zu erhalten, wird immer die von CouchDB bereitgestellte Funktion *emit(key, value)* genutzt. In der Funktion aus obigem Beispiel wird als *key doc.name* übergeben und als *value* null. Das Ergebnis soll also alle vorhandenen Dokumente abfragen und nur den Namen des Dinos ausgeben. Das Ergebnis sieht man in **Abbildung 4**.

Listing 5

```
function(doc) {
  emit(doc.name, null)
}
```

Listing 6

```
function(doc) {
  if(doc.name) {
    emit(doc.name, null)
  }
}
```

Listing 7

```
function(doc) {
  if(doc.aliment === 'meat') {
    emit(doc.name, null)
  }
}
```

Listing 8

```
function(doc) {
  if(doc.name && doc.aliment === 'meat') {
    emit(doc.name, null)
  }
}
```

Listing 9

map-Funktion:

```
function(doc) {
  emit("Total weight", doc.weight);
}
```

reduce-Funktion:

```
function(keys, values) {
  return sum(values)
}
```

Ergebnis ohne reduce:

```
$ curl -X GET http://127.0.0.1:5984/dinos/_design/dinos/_view/total_weight? \
reduce=false
{"total_rows":3,"offset":0,"rows":[
  {"id":"dcefca6f45c49859f34d5e91c200123c","key":"Total weight","value":100},
  {"id":"dcefca6f45c49859f34d5e91c2001bab","key":"Total weight","value":1500},
  {"id":"dcefca6f45c49859f34d5e91c2002ad4","key":"Total weight","value":2000}
]}
```

Ergebnis mit reduce:

```
$ curl -X GET http://127.0.0.1:5984/dinos/_design/dinos/_view/total_weight
{"rows":[
  {"key":null,"value":3600}
]}
```

Bitte unbedingt merken

Bleiben wir einen Augenblick bei diesem einfachen Beispiel. Sie haben gerade gelesen, dass alle Dokumente durchsucht werden, die ein Feld *name* beinhalten. Hierbei gibt es wichtige Anmerkungen zu machen. Eine berechtigte Frage ist: wie kann ich die Anzahl der zu durchsuchenden Dokumente einschränken? Das tut man, indem man prüft, ob das Feld vorhanden ist (Listing 6).

Darauf folgt die nächste Frage: Wenn das bei jedem Aufruf der View über alle Dokumente der Datenbank geschieht, ist das nicht extrem inperformant? Ja das wäre es. Glücklicherweise wird die View nur das erste Mal „computet“. Und bei diesem ersten Mal wird ein *B+-Tree*-Index auf die Ergebnismenge gesetzt, was bei allen weiteren Ausführungen der View einen sehr schnellen Zugriff auf die Daten ermöglicht (auch oder gerade bei sehr hoher Anzahl von Dokumenten im Millionenbereich). Natürlich muss der Index komplett neu erstellt werden, wenn man die View ändert. Werden Daten hinzugefügt, aktualisiert oder gelöscht, wird der Index aktualisiert.

Eine weitere Möglichkeit, um am Anfang der View anzugeben, dass nur bestimmte Dokumente einbezogen werden sollen, ist die Abfrage nach einem bestimmten Value eines Feldes. Listing 7 zeigt z. B. nur alle fleischfressenden Dinos. Und natürlich macht auch die Kombination aus beidem Sinn (Listing 8).

Reduce-Funktionen

Der zweite Teil einer Abfrage an CouchDB besteht meistens aus einer *Reduce*-Funktion, die dazu dient, die Ergebnismenge aus der *Map*-Funktion weiter zu verarbeiten. In den meisten Fällen werden Aggregationen durchgeführt. Und hier bekommt man von CouchDB wieder eine Funktion geschenkt: *sum()*. Wie der Name schon vermuten lässt, summiert diese Funktion Werte (Listing 9).

Gehen wir Step by Step vor. Die *Map* Funktion besteht nur aus dem Aufruf von *emit()*, wobei dieser Funktion als Key der String *Total weight* übergeben wird und als Value das Feld *weight* des aktuellen Dokuments. Es ist zu beachten, dass CouchDB das Ergebnis aus der *map*-Funktion automatisch aufsteigend sortiert anhand des Values (siehe Ergebnis ohne *reduce* → value).

Die *Reduce*-Funktion soll nun im zweiten Schritt eine Summe über die ihr übergebenen Werte bilden. Wenn man noch einmal einen Blick auf die *map*-Funktion wirft, sieht man, dass *emit()* als zweiten Parameter (Value) *doc.weight* entgegennimmt. Und genau darüber bildet die *Reduce*-Funktion die Summe, weshalb als Ergebnis von *reduce* als Value 3600 zurückgeben wird.

Nun will man natürlich nicht immer ein aggregiertes Ergebnis erhalten, sprich, eine *reduce*-Funktion ausführen. Und es ist auch nicht wirklich cool, immer den Parameter *?reduce=false* an den URL anzuhängen. Die einfache Lösung: keine *reduce*-Funktion erstellen. Dadurch wird CouchDB nur die *Map*-Funktion ausführen und das Ergebnis sortiert zurückgeben (Listing 10).

Anhand dieses recht trivialen Beispiels kann man das Zusammenspiel von *map* und *reduce* sehr gut sehen. Spielen Sie in Futon damit herum und probieren Sie unterschiedliche Varianten aus. Als alternatives Anschauungsbeispiel empfehle ich außerdem einen Blick auf die interaktive Demo [5].

Listing 10

```
{
 "_id":"_design/dinos",
 "_rev":"8-6f7a930eeac43a9e53aa04b987aa8ff7",
 "language":"javascript",
 "views":{
  [...]
  "total_weight_no_reduce":{
    "map":"function(doc) {
      emit("Total weight", doc.weight);
    }"
  }
 }
}
```

Listing 11

map-Funktion:

```
function(doc) {
   emit("min age", doc.age);
   emit("min weight", doc.weight);
}
```

reduce-Funktion:

```
function(keys, values) {
  var result = null, i;
  for (i = 0; i < values.length; ++i) {
    result = Math.min(values[i], result || values[i]);
  }
  return result;
}
```

Ergebnis:

```
"min weight"  100
"min age"     70
```

Listing 12

map-Funktion:

```
function(doc) {
   emit("dino info", {name: doc.name, age: doc.age, weight: doc.weight});
}
```

Ergebnis:

```
"dino info"    {name: "Velociraptor", age: 84, weight: 100}
 ID: dcefca6f45
... (für alle Dokumente)
```

Map/Reduce @work

An dieser Stelle sollten die Grundprinzipien einer *Map*-Funktion klar sein. Bislang wurden sehr triviale und nicht weiter gewinnbringende Abfragen erstellt. Natürlich hat man mit den vorliegenden Dokumenten auch nicht die allergrößten Möglichkeiten, aber ein paar weitere Gedankenanstöße sollen hier noch präsentiert werden. Prinzipiell ist man nicht darauf beschränkt, nur eine *emit()*-Funktion in einer *map*-Funktion auszuführen. Das können beliebig viele sein. Es ist hilfreich, wenn man z. B. gleichzeitig das niedrigste Alter und das niedrigste Gewicht wissen möchte. Also – lassen Sie uns das herausfinden (Listing 11).

Zum Schluss noch ein einfacheres Beispiel. Um in der Ausgabe nur bestimmte Daten aus den Dokumenten zu erhalten, kann man *emit()* als zweiten Parameter (Values) ein Objekt übergeben (Listing 12). Das sind alles ziemlich einfache Beispiele. Allerdings zeigen sie auch, dass man mit *map/reduce* in CouchDB so gut wie alle Arten und Formen von Ergebnissen aus Abfragen erstellen kann.

Fazit

CouchDB ist cool! Wirklich! In diesem Artikel habe ich die grundlegenden Möglichkeiten von *Map/Reduce* in CouchDB gezeigt. Sie können damit sehr schnell und einfach Daten aus der CouchDB-Datenbank auslesen, aber auch sehr komplexe Abfragen realisieren. Performant wird das Ganze (nach der ersten Ausführung einer View) durch die Nutzung eines *B+-Tree*-Index bei den Views. Auch weitere Pluspunkte für CouchDB. Ich hoffe, Sie haben Geschmak an CouchDB gefunden. Nicht zuletzt auch nach der Lektüre des ersten Artikels über CouchDB im PHP Magazin [1]. Haben Sie einen Anwendungsfall für CouchDB? Schreiben Sie mir eine E-Mail und berichten Sie gerne. Übrigens, CouchDB hat eine extrem gute, aktive und hilfsbereite Community. Schauen Sie doch mal vorbei [6].

Andy Wenk ist Software Developer bei SinnerSchrader in Hamburg, hat momentan viel Spaß mit JavaScript und Ruby on Rails und mag Datenbanken wie PostgreSQL und CouchDB. Zusammen mit Till Klampaeckel (@klimpong) aus Berlin schreibt er momentan das erste deutschsprachige CouchDB-Buch. Sie erreichen ihn unter andy@nms.de und @awenkhh.

Links & Literatur

[1] Wenk, Andreas: „Entspannt auf der Couch lümmeln – Ein PHP-Wrapper für die CouchDB“, in: PHP Magazin 5.2010, S. 66 ff., http://it-republik.de/php/artikel/Ein-PHP-Wrapper-fuer-die-CouchDB-3463.html

[2] http://www.couchone.com/get#mac

[3] http://www.couchone.com

[4] http://wiki.apache.org/couchdb/Reference

[5] http://labs.mudynamics.com/wp-content/uploads/2009/04/icouch.html

[6] http://couchdb.apache.org/community/