FREE eBook

LEARNING couchdb

Free unaffiliated eBook created from
Stack Overflow contributors.

#couchdb

Table of Contents

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: couchdb

It is an unofficial and free couchdb ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official couchdb.

The content is released under Creative Commons BY-SA, and the list of contributors to each chapter are provided in the credits section at the end of this book. Images may be copyright of their respective owners unless otherwise specified. All trademarks and registered trademarks are the property of their respective company owners.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with couchdb

Remarks

Why CouchDB?

CouchDB has a JSON document storage model with a powerful query engine based on a HTTP API.

Using JavaScript in *design* documents, you have control of ways to query, map, combine, and filter your data. Providing fault tolerance, extreme scalability, and incremental replication, CouchDB is a good choice for a non-relational, document database.

> While a traditional relational database requires you to model your data up front, CouchDB's schema-free design unburdens you with a powerful way to aggregate your data after the fact, just like we do with real-world documents. We'll look in depth at how to design applications with this underlying storage paradigm.

Versions

Version	Release Date
2.0.0	2016-09-20
1.6.1	2014-09-03

Examples

Installation and Setup

Ubuntu

On recent Ubuntu versions, you can install an up-to-date version of CouchDB with `sudo apt-get install couchdb`. For older versions, such as Ubuntu 14.04, you should run:

```
sudo add-apt-repository ppa:couchdb/stable -y
sudo apt-get update
sudo apt-get install couchdb -y
```

Fedora

To install couchdb in fedora ryou can do `sudo dnf install couchdb`

Mac OS X

To install CouchDB on Mac OS X, you can install the Mac app from the CouchDB downloads section.

Windows

To install CouchDB on Windows, you can simply download the executable from CouchDB downloads section.

Hello World

By default, CouchDB listens on port 5984. Visiting `http://127.0.0.1:5984` will yield a response that looks like this:

```
{"couchdb":"Welcome","version":"1.6.1"}
```

CouchDB comes out-of-the-box with a GUI called *Futon*. You can find this interface at `http://127.0.0.1:5984/_utils`. Here, you can easily set up an administrator account and configure other important settings.

Read Getting started with couchdb online: https://riptutorial.com/couchdb/topic/2649/getting-started-with-couchdb

Chapter 2: Design Documents

Remarks

Design documents behave like all documents in terms of revisions, replication, and conflicts. You can also add attachments to design documents.

Examples

_design/example

Design documents contain application logic. Any document in a database that has an _id starting with "_design/" can be used as design document. Usually there is one design document for each application.

```
{
    "_id": "_design/example",
    "view": {
        "foo": {
            "map": "function(doc){...};",
            "reduce": "function(keys, values, rereduce){...};"
        }
    }
}
```

The example above defines a **view** named *foo*, which can be requested from the following path, assuming the database is named *db*:

> http://localhost:5984/db/_design/example/_view/foo

Read Design Documents online: https://riptutorial.com/couchdb/topic/6958/design-documents

Chapter 3: Ektorp java client

Remarks

Test

Examples

Opening a connection to CouchDB

```
HttpClient httpClient = new StdHttpClient.Builder().
    url("http://yourcouchdbhost:5984").
    username("admin").
    password("password").
    build();

CouchDbInstance dbInstance = new StdCouchDbInstance(httpClient);
```

Connecting to a database

Given you have a valid CouchDbInstance instance, you connect to a database within CouchDB in the following manner

```
CouchDbConnector connector = dbInstance.createConnector("databaseName", true);
```

Simple CRUD with POJOs

One of the great things about Ektorp, is that it provides ORM like functionality, straight out of the box. This example will walk you through creating a simple POJO and doing standard CRUD operation on it

Creating a simple POJO

First off, we define a POJO as follows

```
import com.fasterxml.jackson.annotation.JsonInclude;
import com.fasterxml.jackson.annotation.JsonProperty;

@JsonInclude(JsonInclude.Include.NON_NULL)
public class Person {
    @JsonProperty("_id") private String id;
    @JsonProperty("_rev") private String revision;
    private String name;

    public String getId() {
        return id;
    }

    public String getRevision() {
        return revision;
    }

    public String getName() {
        return name;
    }


    public void setId(String id) {
        this.id = id;
    }

    public void setRevision(String revision) {
        this.revision = revision;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

So, whats happening here? The annotation `@JsonInclude(JsonInclude.Include.NON_NULL)` tells jackson not to serialize null fields into JSON. So, for example, if the id property is null you wont have the id property in the resulting JSON at all.

Additionally, the `@JsonProperty("_id")` and `@JsonProperty("_rev")` annotations are directives, informing the serializer/unserializer what JSON properties to map these values to. Documents in CouchDB must have both a _id and a _rev field, thus all POJOs which you intent to persist in CouchDB, must include a id and revision properties as above. You are free to name your properties differently in the POJO, as long as you don't change the annotations. For example,

```
@JsonProperty("_id") private String identity;
@JsonProperty("_rev") private String version;
```

Persisting new instances to CouchDB

Now, creating a brand new document, in the database is done as follows, presuming you have a valid CouchDbInstance instance, and that you wish to persist the document in a database named *person*

```
CouchDbConnector connector = dbInstance.createConnector("person", true);

Person person = new Person();
person.setName("John Doe");
connector.create(person);
```

Now, in this scenario, CouchDB will automatically create a new ID and Revision for you. If you dont want this, you can use

```
connector.create("MyID", person);
```

Loading, updating and deleting documents

Presuming you have a CouchDBConnector instance ready, we can load instances of our POJO as follows

```
Person person = connector.get(Person.class, "id");
```

then we can manipulate it, and update it as follows

```
person.setName("Mr Bean");
connector.update(person);
```

Notice, if the revision of the document in couch, doesn't match the revision of the document you are sending, the update will fail, and you need to load the latest version from the database and merge your instances accordingly.

and finally, should we wish to delete the instance, its as simple as

```
connector.delete(person);
```

Read Ektorp java client online: https://riptutorial.com/couchdb/topic/6417/ektorp-java-client

Chapter 4: Views

Examples

Views for people

To show you how work the views, we will assume that we want to query the document of type *people*. To do so, we will first need a design document that will hold our views.

*Note: for the purpose of the example, we will use many views inside of 1 design document. Therefore, in a production environment, you may prefer to have 1 view per design document. The reason is that every time you update the design document, all the views are rerun (**at least for Cloudant**).*

So at this point, I assume you know what is a design document and how it works. Our design document will look like this :

```
{
    "_id":"_design/people",
    "language":"javascript"
}
```

Then, inside of this document, you will have a property of views. This property holds an object containing the views. Each view has its own object that contains a **map** function and optionally, a **reduce** function. Here is how it looks if we have a view that fetches all the people from the database :

```
{
    "_id":"_design/people",
    "language":"javascript",
    "views":{
        "all":{
            "map":"function(doc){if(doc.type ===\"people\")emit(doc._id);}"
        }
    }
}
```

Such a view would return something like this :

```
{
    "total_rows": 2,
    "offset": 0,
    "rows": [
        { "id": "people_23929319009123", "key": "people_23929319009123", "value": null },
        { "id": "people_11482871000723", "key": "people_11482871000723", "value": null }
    ]
}
```

What we have made so far is the view that gives us all the people. The equivalent in SQL would be : `SELECT * FROM table WHERE type="people"`. I will explain in detail how work the map function.

Map function : all

```
function(doc) {
    if (doc.type === "people") emit(doc._id);
}
```

First, you need to know that the map function will be executed for each document. Now for the map function, you need to know that it takes one parameter : **doc**. Inside your map function, your logic will determine if the doc needs to be mapped or not. If yes, you will use the **emit()** function to index it. The emit function takes 2 parameters.

1. The key to index
2. The value to emit

At the end, it will create an array with 3 columns : **id**,**key**,**value**.

*Note: NEVER BUT NEVER emit the doc as the value. This is totally useless since using the **include_docs** parameter will fetch the documents associated to the id.*

Complex keys

Now let's say that we want to fetch the people according to different parameters. Let's say that I want to query the users on their name, their gender and their children count.

In this case, we would have a view like this :

```
function(doc) {
    if (doc.type === "people") {
        emit([doc.name,doc.gender,doc.childrenCount]);
    }
}
```

For the example, I didn't validate that the objects had the required parameter since I won't cause me any problem. Therefore, it may vary from your context. You might want to check if they have the parameter birthDate for example.

So now, as you can see, we still have **one** key but it's a complex one. The trick here is that our key is an array so we can have multiple keys.

Now, you might be asking yourself, but hey, how do I use this? It's weird! Stay calm, I will show you how!

When you query multiple keys, it's a good idea to know how works the comparison in CouchDB. For more info, take a look at this. The most important thing to know is that, if you are using ranges and you want to query all the elements on one key, you need to use the `starkey=[null]&endkey=[\ufff0]`. Since null is the lowest value and \ufff0 is the highest character, it will get everything between this.

So, If I want to get all the women named Julia, I would do the following:

```
http://localhost:5984/db/_design/people/_view/byNameGenderChildren?starkey=["Julia","Female"]&endkey=["J
```

Basically, we take all the rows with the key [Julia,Female] and for the third part, we take anything between the lowest value(null) and the highest(\ufff0) wich means everything.

Next, if I want to fetch all the male with 3 children? Easy as this :

```
http://localhost:5984/db/_design/people/_view/byNameGenderchildren?startkey[null,"Male",3]&endkey=[\ufff
```

Read Views online: https://riptutorial.com/couchdb/topic/7007/views

Credits

S. No	Chapters	Contributors
1	Getting started with couchdb	Alexis Côté, Bernhard Gschwantner, Community, Flimzy, Giancarlo Corzo, Luke Taylor, pwagner
2	Design Documents	pwagner
3	Ektorp java client	JustDanyul
4	Views	Alexis Côté