

CSC561 NoSQL Databases

Lab 6: CouchDB

Couch is RESTful:

Why Couch:

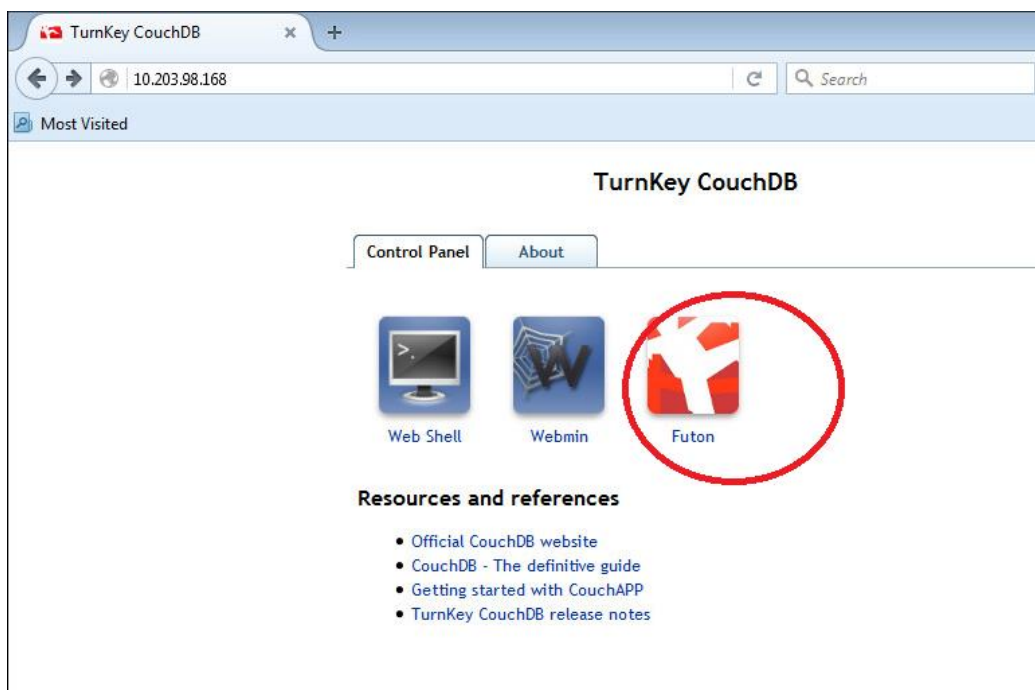
Apache CouchDB is the quintessential JSON- and REST-based document-oriented database. First released in 2005, CouchDB was designed with the Web in mind and all the innumerable flaws, faults, failures, and glitches that come with it. Consequently, CouchDB offers a robustness unmatched by most other databases. Whereas other systems tolerate occasional network drops, CouchDB thrives even when connectivity is only rarely available.

Somewhat like MongoDB, CouchDB stores *documents*—JSON objects consisting of key-value pairs where values may be any of several types, including other objects nested to any depth. There is no ad hoc querying, though; indexed views produced by incremental mapreduce are the principal way you find documents. Like Riak, all calls to CouchDB happen over its REST interface. Replication can be one-way or bidirectional and ad hoc or continuous. CouchDB gives you a lot of flexibility to decide how to structure, protect, and distribute your data. It is able to scale down as well as up, so CouchDB fits problem spaces of varying size and complexity with ease.

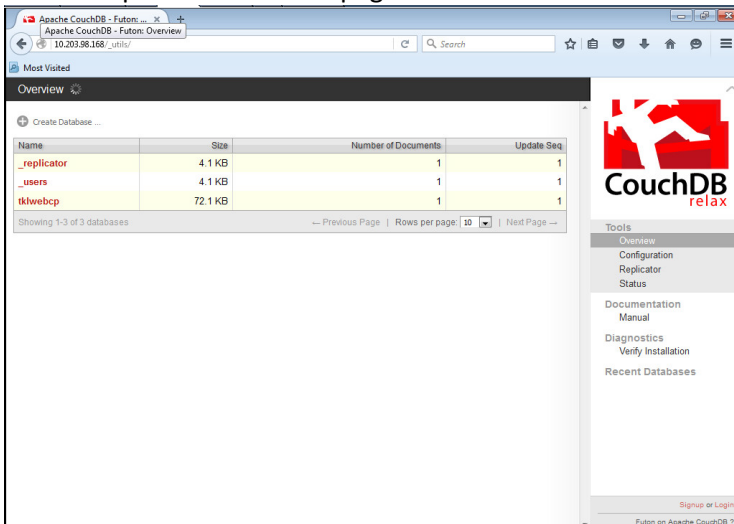
CouchDB lives up to its tag line: relax. Instead of focusing only on big-data cluster installations, CouchDB aims to support a variety of deployment scenarios from the datacenter down to the smartphone. You can run CouchDB on your Android phone, on your laptop, and in your datacenter. Written in Erlang, CouchDB is built to persist—the only way to shut it down is to kill the process. With its append-only storage model, your data is virtually incorruptible and easy to replicate, back up, and restore.

Starting Couch:

CouchDB comes with a useful web interface called Futon. Once you have your CouchDB back-end machine started, start your Front-end machine and open a web browser to <http://<your back end machine IP>>. Click on the Futon icon **If you get a security warning you need to verify that you are not trying to go to the HTTPS version of the site.**



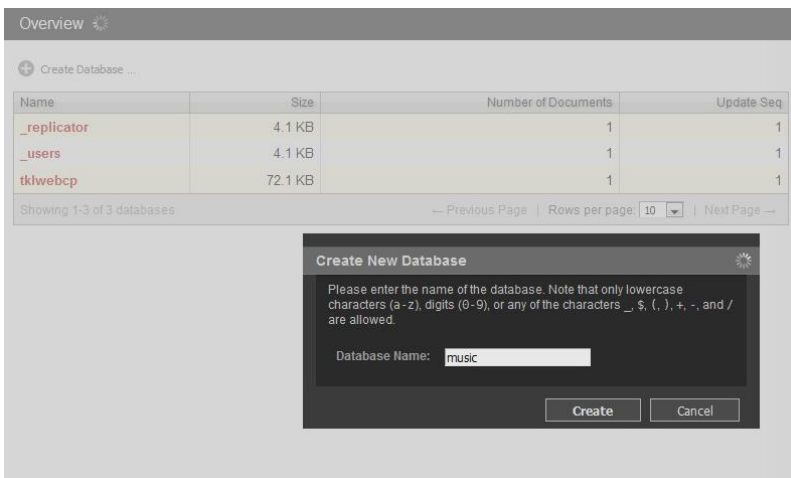
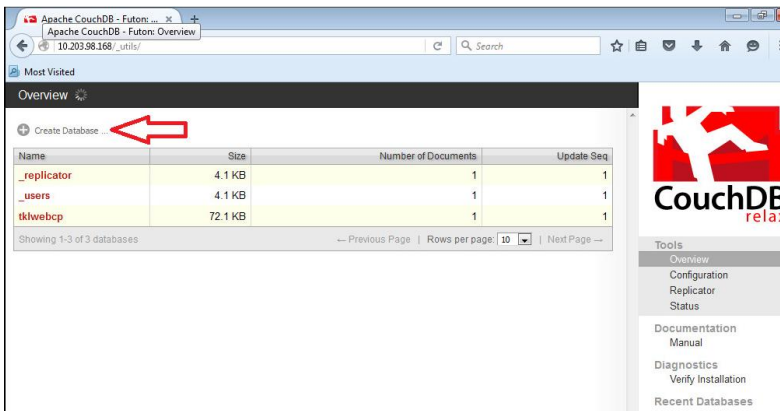
This will open the Overview page:



In the bottom right-hand corner of the screen is a login button. Login as “admin” using password “p@ssw0rd”.

CouchDB CRUD

Before we can start working with documents, we need to create a database to house them. We’re going to create a database to store musicians along with their album and track data. Click the Create Database... button. In the popup, enter “music” and click Create.



This will redirect you automatically to the database's page. From here, we can create new documents or open existing ones.

On the music database's page, click the New Document button. This will take you to a new page that looks like

Apache CouchDB - Futon: ...

10.203.98.168/_utils/document.html?music

Overview > music > 5d75cb1750481d84afe81680090000fd

Save Document Add Field Upload Attachment...

Field	Value
_id	5d75cb1750481d84afe81680090000fd

Fields Source

← Previous Version | Next Version →

CouchDB relax

Tools

- Overview
- Configuration
- Replicator
- Status

Documentation

- Manual

Diagnostics

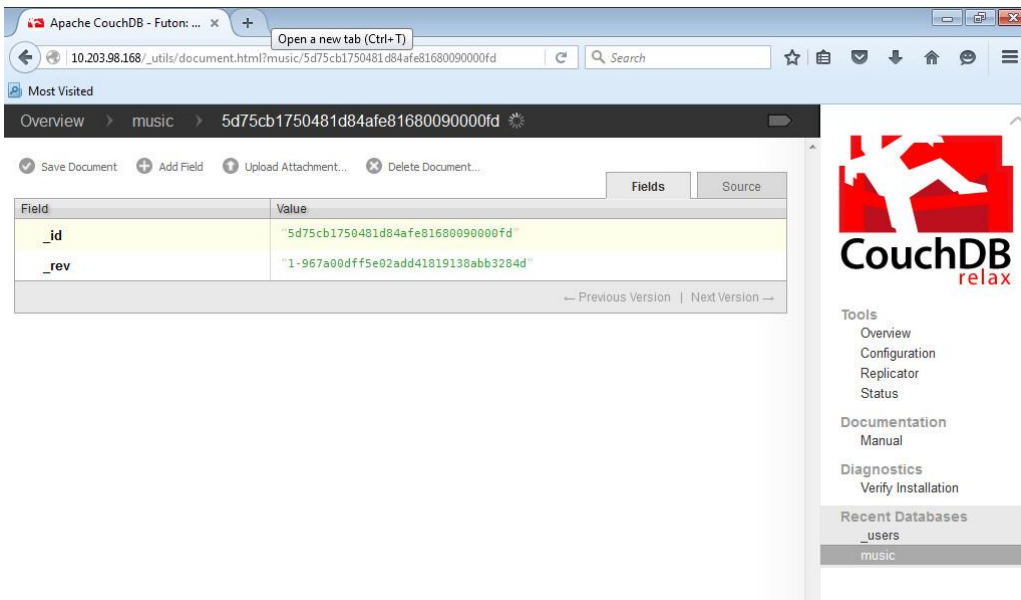
- Verify Installation

Recent Databases

- _users
- music

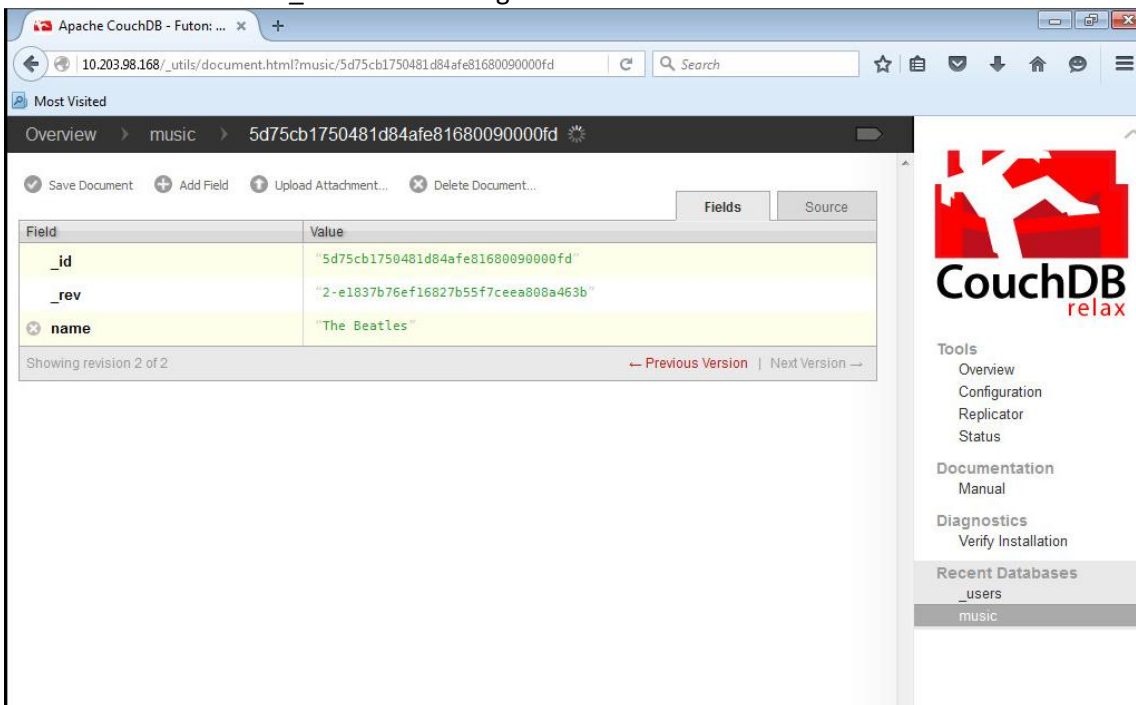
Just as in MongoDB, a document consists of a JSON object containing key-value pairs called fields. All documents in CouchDB have an `_id` field, which must be unique and can never be changed. You can specify an id explicitly, but if you don't, CouchDB will generate one for you. In our case, the default is fine, so click the green checkmark at the end of the `_id` field and then the Save Document to finish.

Immediately after saving the document, CouchDB will assign it an additional field called `_rev`. The `_rev` field will get a new value every time the document changes. The format for the revision string consists of an integer followed by a dash and then a pseudorandom unique string. The integer at the beginning denotes the numerical revision, in this case 1. Field names that begin with an underscore have special meaning to CouchDB, and `_id` and `_rev` are particularly important. To update or delete an existing document, you must provide both an `_id` and the matching `_rev`. If either of these do not match, CouchDB will reject the operation. This is how it prevents conflicts by ensuring only the most recent document revisions are modified.



There are no transactions or locking in CouchDB. To modify an existing record, you first read it out, taking note of the `_id` and `_rev`. Then, you request an update by providing the full document, including the `_id` and `_rev`. All operations are first come, first served. By requiring a matching `_rev`, CouchDB ensures that the document you think you're modifying hasn't been altered by another user.

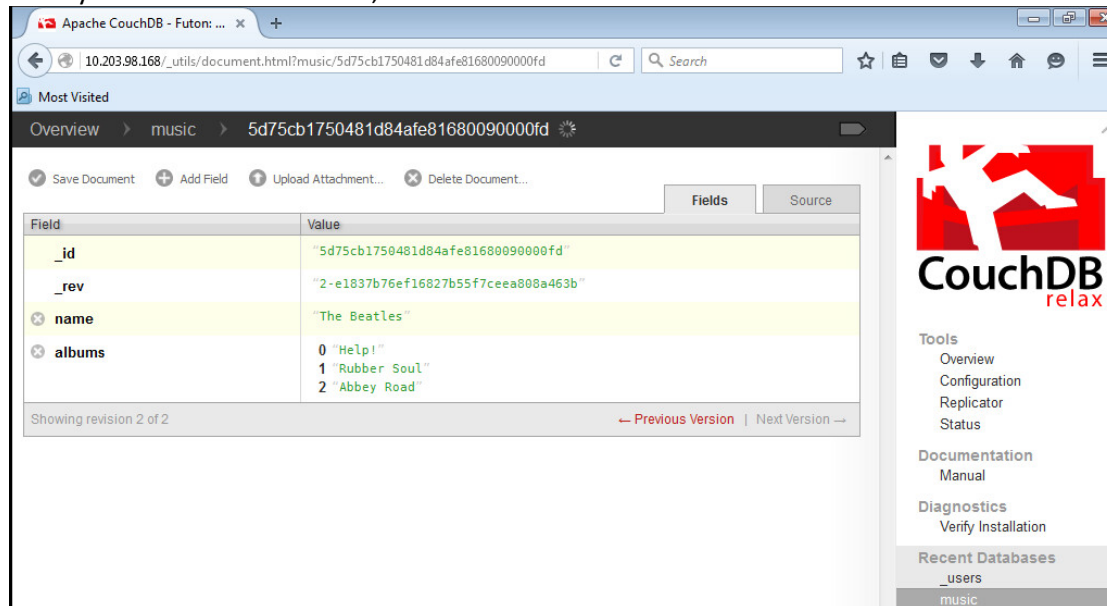
With the document page still open, click the Add Field button. In the Field column, enter name, and in the Value column, enter The Beatles. Click the green check mark next to the value to ensure it sticks, and then click the Save Document button. Notice how the `_rev` field now begins with 2.



CouchDB is not limited to storing string values. It can handle any JSON structure nested to any depth. Click the Add Field button again. This time, set Field to `albums`, and for Value enter the following (this is not an exhaustive list):

```
[{"Help!","Rubber Soul","Abbey Road"]
```

After you click Save Document, it should look like:



The screenshot shows the Apache CouchDB Futon interface. The browser address bar displays the URL `10.203.98.168/_utils/document.html?music/5d75cb1750481d84afe81680090000fd`. The document is titled `music/5d75cb1750481d84afe81680090000fd`. The document content is as follows:

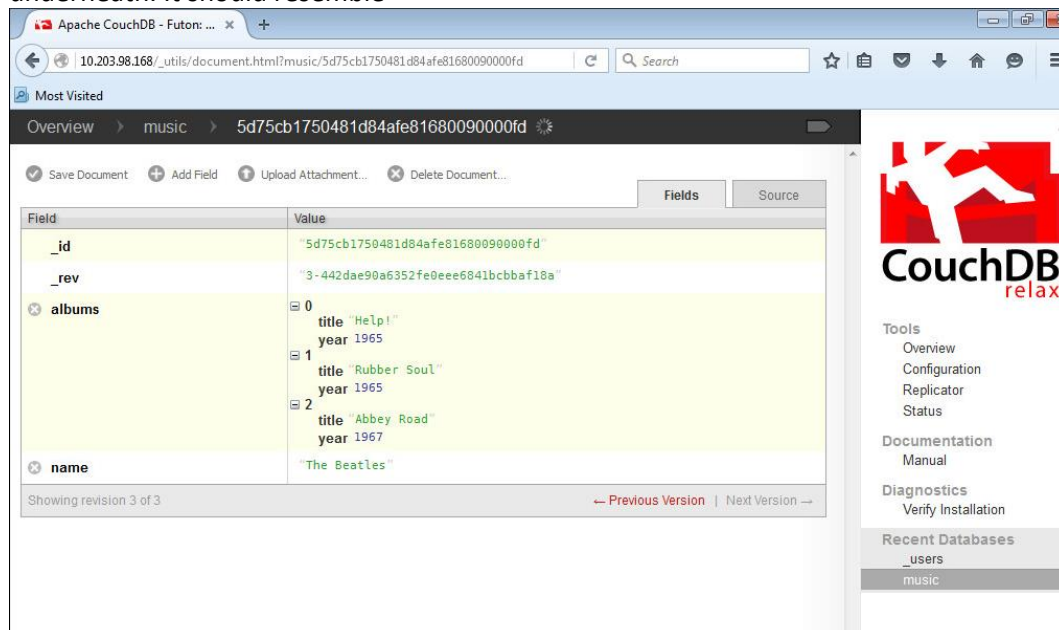
Field	Value
<code>_id</code>	<code>"5d75cb1750481d84afe81680090000fd"</code>
<code>_rev</code>	<code>"2-e1837b76ef16827b55f7ceea808a463b"</code>
<code>name</code>	<code>"The Beatles"</code>
<code>albums</code>	<code>[0 "Help!", 1 "Rubber Soul", 2 "Abbey Road"]</code>

The interface shows the document is at revision 2 of 2. The right sidebar contains links for Tools (Overview, Configuration, Replicator, Status), Documentation (Manual), Diagnostics (Verify Installation), and Recent Databases (_users, music).

There's more relevant information about an album than just its name, so let's add some. Modify the albums field and replace the value you just set with this:

```
{{"title": "Help!",
"year": 1965
},{
"title": "Rubber Soul",
"year": 1965
},{
"title": "Abbey Road",
"year": 1969
}}
```

After you save the document, this time you should be able to expand the albums value to expose the nested documents underneath. It should resemble



The screenshot shows the Apache CouchDB Futon interface after saving the document. The document is now at revision 3 of 3. The document content is as follows:

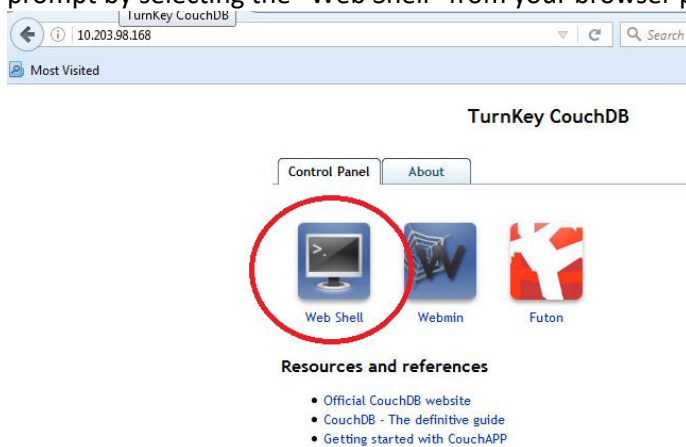
Field	Value
<code>_id</code>	<code>"5d75cb1750481d84afe81680090000fd"</code>
<code>_rev</code>	<code>"3-442dae90a6352fe0eee6841bcbbaf18a"</code>
<code>albums</code>	<code>[0 {"title": "Help!", "year": 1965}, 1 {"title": "Rubber Soul", "year": 1965}, 2 {"title": "Abbey Road", "year": 1967}]</code>
<code>name</code>	<code>"The Beatles"</code>

The interface shows the document is at revision 3 of 3. The right sidebar contains links for Tools (Overview, Configuration, Replicator, Status), Documentation (Manual), Diagnostics (Verify Installation), and Recent Databases (_users, music).

Performing RESTful CRUD Operations with cURL

All communication with CouchDB is REST-based, and this means issuing commands over HTTP. CouchDB isn't the first database we've studied with this quality. Riak also relies on REST for all client communication. And like we did with Riak, we can communicate with CouchDB using the command-line tool cURL.

Here we'll perform some basic CRUD operations before moving on to the topic of views. To start, open a command prompt by selecting the "Web Shell" from your browser page



Login using user: couchdb (there is no password on this account) and run the following:

```
$ HOST="http://admin:p%40ssw0rd@localhost:5984"  
$ curl $HOST
```

It should return

```
$ HOST="http://admin:p%40ssw0rd@localhost:5984"  
$ curl $HOST  
{  
  "couchdb": "Welcome",  
  "uuid": "a40923284854e24e2e513085b14f3a4a",  
  "version": "1.6.1",  
  "vendor": {  
    "name": "The Apache Software Foundation",  
    "version": "1.6.1"  
  }  
}
```

The HOST command sets a variable so that we can access the database as our admin account without having to log in each time. This is NOT secure as we are sending the username and password over HTTP in plain text, but it will work for now for our convenience.

Issuing GET requests (cURL's default) retrieves information about the thing indicated in the URL. Accessing the root as you just did merely informs you that CouchDB is up and running and what version is installed. Next let's get some information about the music database we created earlier

```
$ curl $HOST/exitmusic/
```

```
$ curl $HOST/music  
{  
  "db_name": "music",  
  "doc_count": 1,  
  "doc_del_count": 0,  
  "update_seq": 4,  
  "purge_seq": 0,  
  "compact_running": false,  
  "disk_size": 16479,  
  "data_size": 417,  
  "instance_start_time": "1480541343008055",  
  "disk_format_version": 6,  
  "committed_update_seq": 4  
}
```

This returns some information about how many documents are in the database, how long the server has been up, and how many operations have been performed.

Reading a Document with GET

To retrieve a specific document, append its `_id` to the database URL like so:


```
$ curl $HOST/music/<your document _id>
```

For my machine it is:

```
curl $HOST/music/c961b1dcc8db6a01d81b37388c000662
```

Entering it returns the document we made:

```
$ curl $HOST/music/c961b1dcc8db6a01d81b37388c000662
{"_id":"c961b1dcc8db6a01d81b37388c000662","_rev":"3-2b2852e02242b27f09cbde9c15e4ee06","name":"The Beatles","albums":[{"title":"Help!","year":1965}, {"title":"Rubber Soul","year":1965}, {"title":"Abbey Road","year":1969}]}
$
```

In CouchDB, issuing GET requests is always safe. CouchDB won't make any changes to documents as the result of a GET. To make changes, you have to use other HTTP commands like PUT, POST, and DELETE.

Creating a Document with POST

To create a new document, use POST. Make sure to specify a Content-Type header with the value application/json; otherwise, CouchDB will refuse the request.

```
curl -i -X POST "$HOST/music/" -H "Content-Type: application/json" -d '{"name": "Wings"}'
```

This should return:

```
$ curl -i -X POST "$HOST/music/" -H "Content-Type: application/json" -d '{"name": "Wings"}'
HTTP/1.1 201 Created
Server: CouchDB/1.6.1 (Erlang OTP/17)
Location: http://localhost:5984/music/c961b1dcc8db6a01d81b37388c000799
ETag: "1-2fe1dd1911153eb9df8460747dfe75a0"
Date: Wed, 30 Nov 2016 23:00:48 GMT
Content-Type: text/plain; charset=utf-8
Content-Length: 95
Cache-Control: must-revalidate

{"ok":true,"id":"c961b1dcc8db6a01d81b37388c000799","rev":"1-2fe1dd1911153eb9df8460747dfe75a0"}
$
```

The HTTP response code 201 Created tells us that our creation request was successful. The body of the response contains a JSON object with useful information such as the _id and _rev values.

Updating a Document with PUT

The PUT command is used to update an existing document or create a new one with a specific _id. Just like GET, the URL for a PUT URL consists of the database URL followed by the document's _id (returned in the response from the POST command we just entered).

```
$ curl -i -X PUT \
"$HOST/music/c961b1dcc8db6a01d81b37388c000799" \
-H "Content-Type: application/json" \
-d '{"_id":"c961b1dcc8db6a01d81b37388c000799",
  "_rev":"1-2fe1dd1911153eb9df8460747dfe75a0",
  "name": "Wings",
```

```
"albums": ["Wild Life", "Band on the Run", "London Town"]
}'
```

This should return:

```
$ curl -i -X PUT "$HOST/music/c961b1dcc8db6a01d81b37388c000799" -H "Content-Type: application/json"
-d '{"_id":"c961b1dcc8db6a01d81b37388c000799", "_rev": "1-2fe1dd1911153eb9df8460747dfe75a0","name":
"Wings", "albums": ["Wild Life", "Band on the Run", "London Town"] }'
HTTP/1.1 201 Created
Server: CouchDB/1.6.1 (Erlang OTP/17)
Location: http://localhost:5984/music/c961b1dcc8db6a01d81b37388c000799
ETag: "2-17e4ce41cd33d6a38f04a8452d5a860b"
Date: Wed, 30 Nov 2016 23:13:10 GMT
Content-Type: text/plain; charset=utf-8
Content-Length: 95
Cache-Control: must-revalidate

{"ok":true,"id":"c961b1dcc8db6a01d81b37388c000799","rev":"2-17e4ce41cd33d6a38f04a8452d5a860b"}
$
```

Unlike MongoDB, in which you modify documents in place, with CouchDB you always overwrite the entire document to make any change. The Futon web interface we saw earlier may have made it look like you could modify a single field in isolation, but behind the scenes it was rerecording the whole document when you hit Save.

As we mentioned earlier, both the `_id` and `_rev` fields must exactly match the document being updated, or the operation will fail. To see how, try executing the same PUT operation again. You should get an HTTP 409 Conflict response with a JSON object describing the problem. This is how CouchDB enforces consistency.

Removing a Document with DELETE

Finally, we can use the DELETE operation to remove a document from the database.

```
$ curl -i -X DELETE "$HOST/music/c961b1dcc8db6a01d81b37388c000799" \
-H "If-Match: 2-17e4ce41cd33d6a38f04a8452d5a860b"
```

```
$ curl -i -X DELETE "$HOST/music/c961b1dcc8db6a01d81b37388c000799" -H "If-Match: 2-17e4ce41cd33d6a38
f04a8452d5a860b"
HTTP/1.1 200 OK
Server: CouchDB/1.6.1 (Erlang OTP/17)
ETag: "3-42aafb7411c092614ce7c9f4ab79dc8b"
Date: Wed, 30 Nov 2016 23:17:50 GMT
Content-Type: text/plain; charset=utf-8
Content-Length: 95
Cache-Control: must-revalidate

{"ok":true,"id":"c961b1dcc8db6a01d81b37388c000799","rev":"3-42aafb7411c092614ce7c9f4ab79dc8b"}
$
```

The DELETE operation will supply a new revision number, even though the document is gone. It's worth noting that the document wasn't really removed from disk, but rather a new empty document was appended, flagging the document as deleted. Just like with an update, CouchDB does not modify documents in place. But for all intents and purposes, it's deleted.

CONTINUE TO NEXT PAGE FOR WORK TO SUBMIT

Work on your own to submit

Be sure to do a 'git pull' to sync your repo. When you submit your assignment to GitHub, use couchdb for the server name in your curl statements.

1. Use cURL to create a new database with a name of restaurants.
2. Add 3 documents for the restaurants database with the fields `_id` (the id should be the name of the restaurant with underscores – for example, `taste_of_thai`), `name`, `food_type` (American, Chinese, Mexican, Thai, etc), `phonenum`, and `website`. The `food_type` will be an array.
3. To submit your lab, edit the `lab6a.sh` script in GitHub with the appropriate curl commands to the `lab6a` folder.