Joshua Catoe

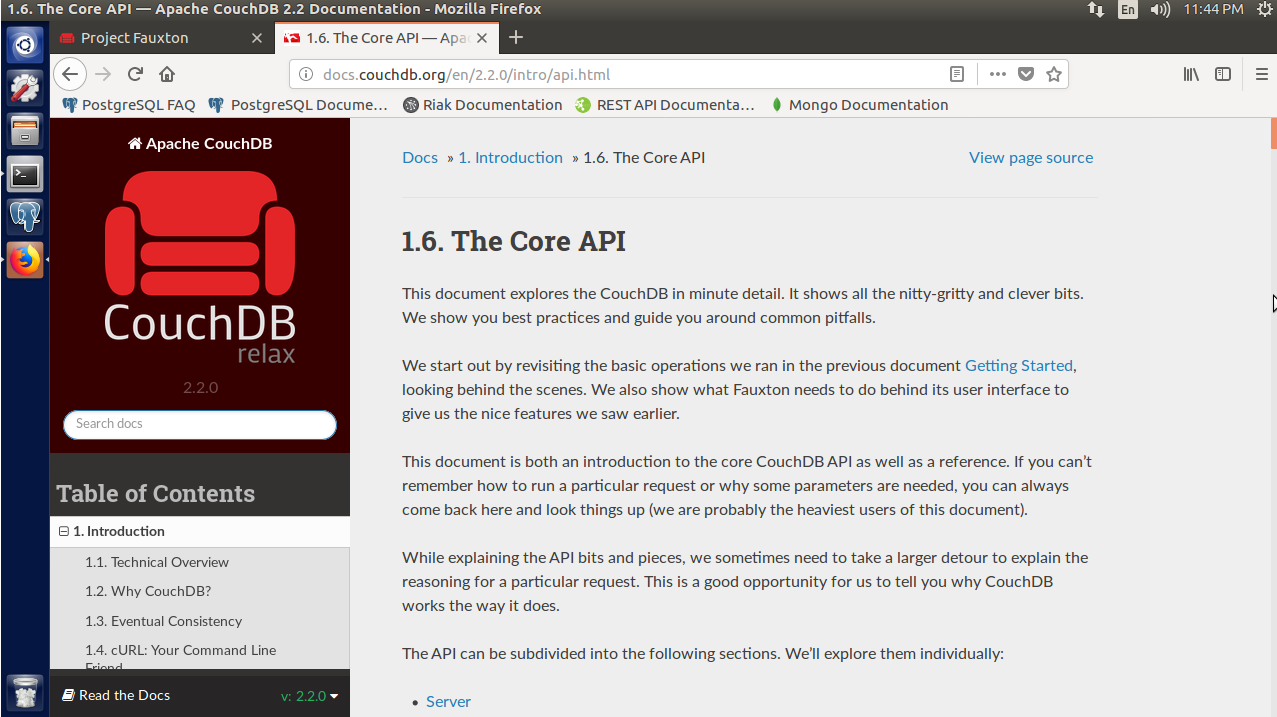
07/29/18

CouchDB

**Day 1**

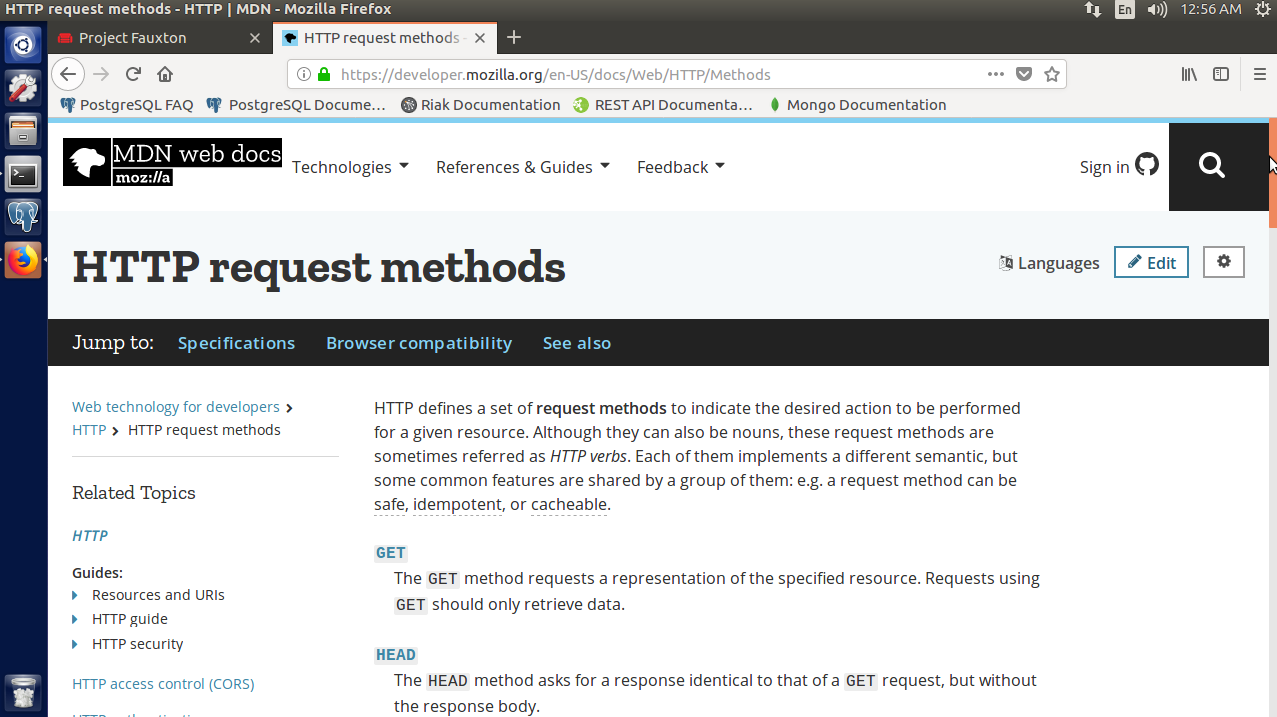
Find:

1. Find the CouchDB HTTP Document API documentation online.



The API documentation is linked to on the main docs page of the CouchDB website. Strangely, the links from a Google search page are for older versions and lead to a page with a “Permission denied” error. It looks like the only way to get to the docs for the API is through the CouchDB webpage or a direct link.

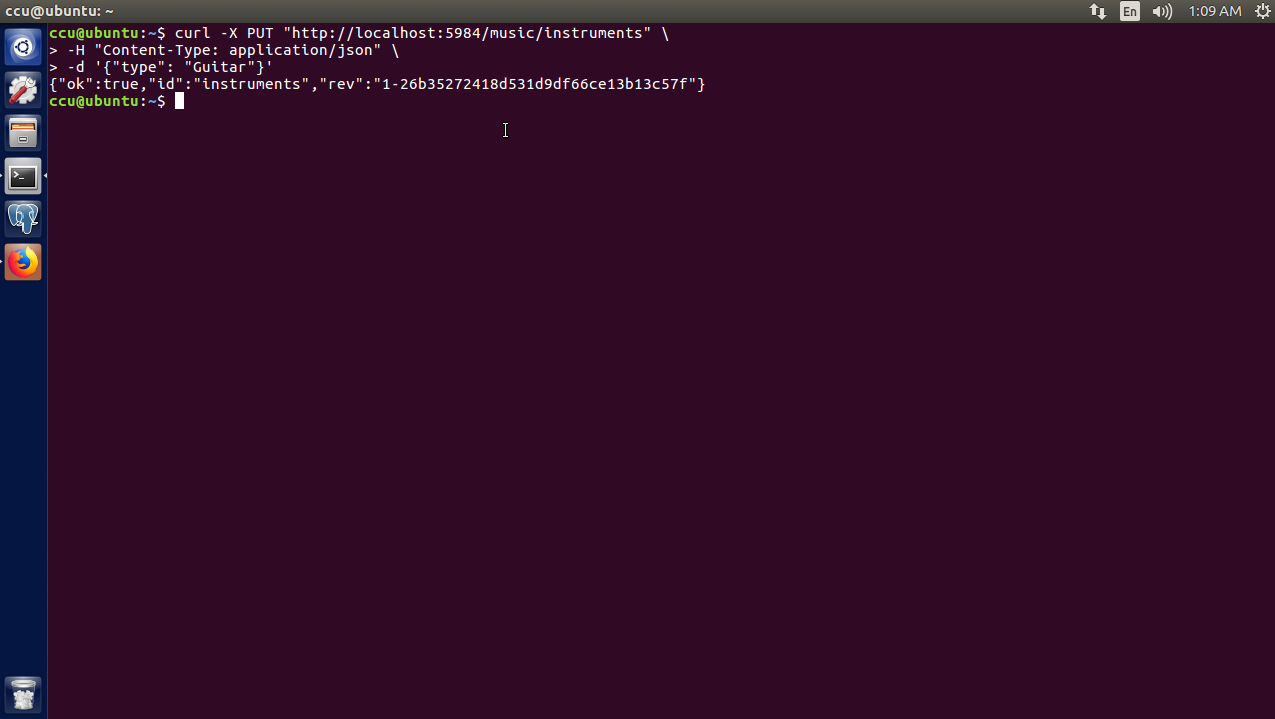
2. We’ve already used GET, POST, PUT, and DELETE. What other HTTP commands are supported?



I found a list of HTTP request methods from mozilla.org. Other than GET, POST, PUT, DELETE, there are HEAD, CONNECT, OPTIONS, TRACE, and PATCH.

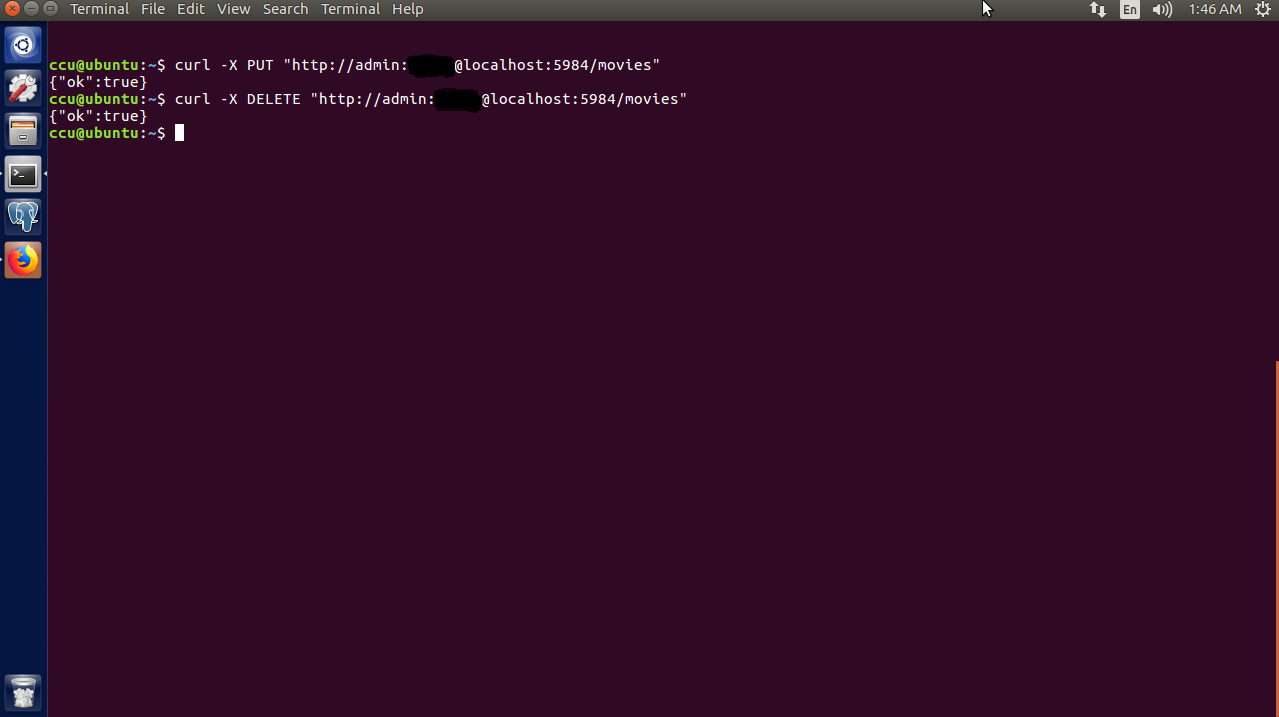
Do:

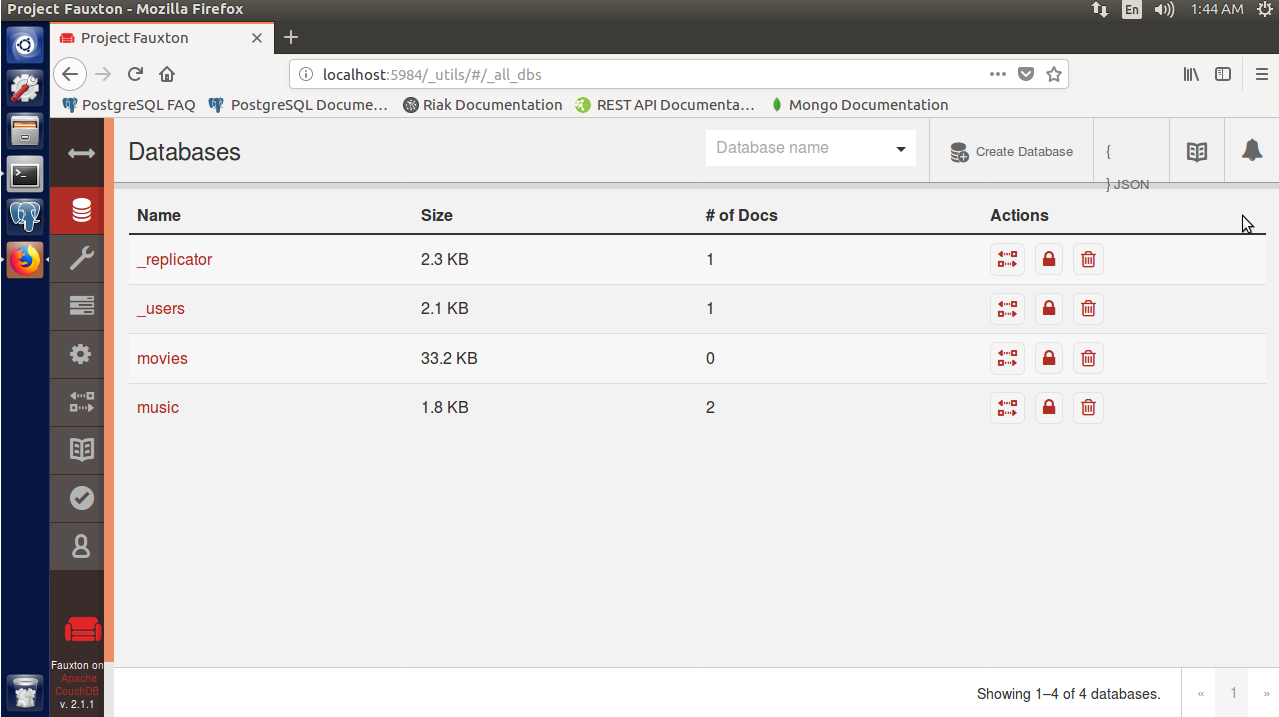
1. Use cURL to PUT a new document into the music database with a specific \_id of your choice.

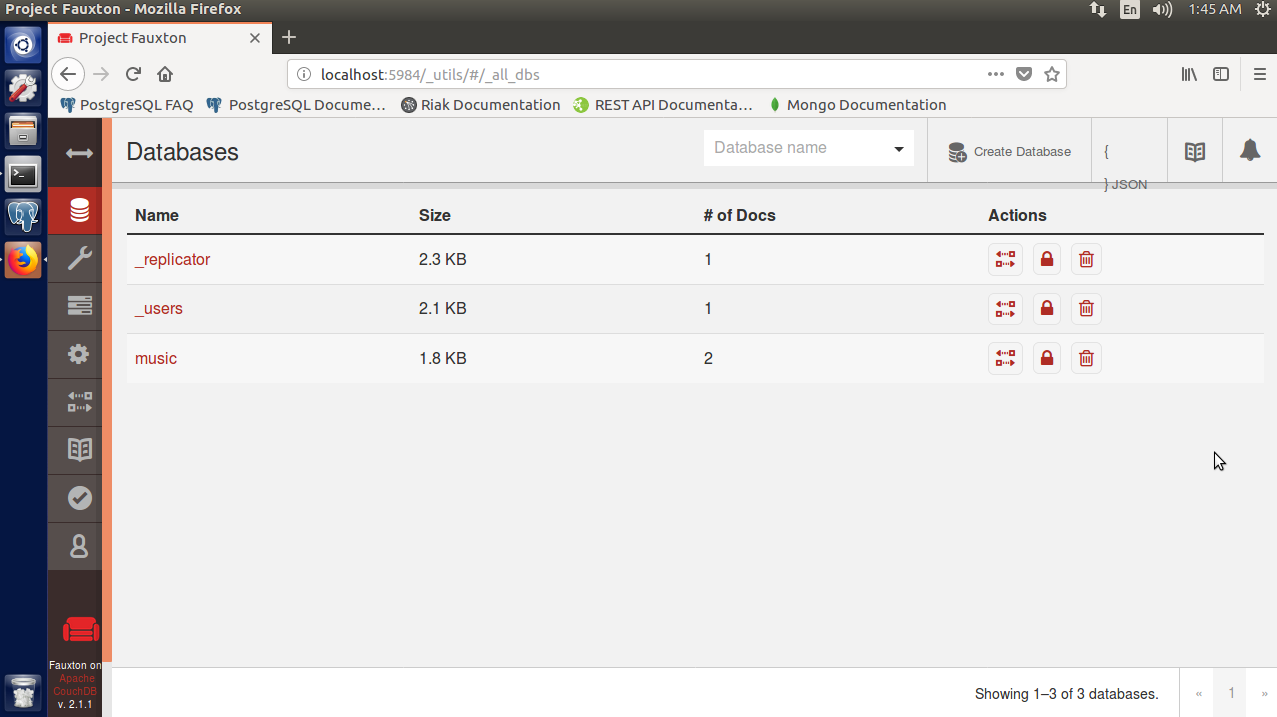


All this took was a simple cURL PUT command. For the **\_id**, I decided on **instruments**, keeping with the music theme, and I also went ahead and put a in a **type** field with the **“Guitar”** type.

2. Use curl to create a new database with a name of your choice, and then delete that database also via cURL.

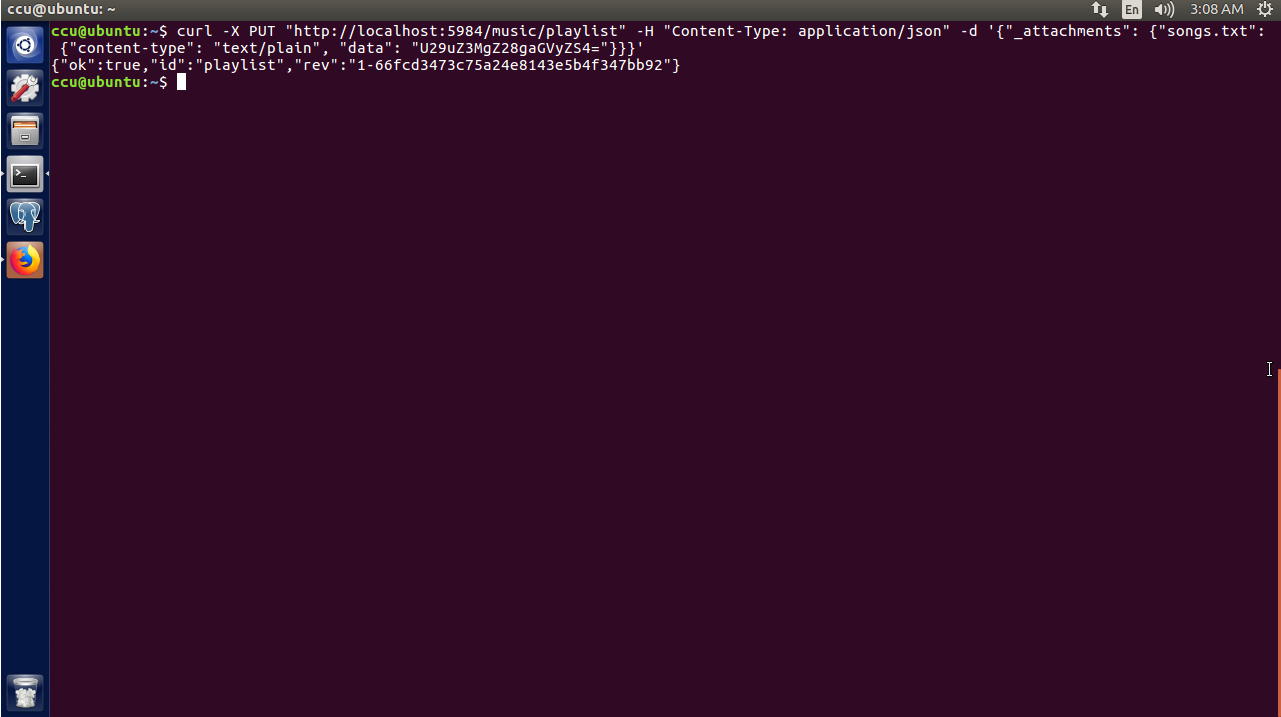


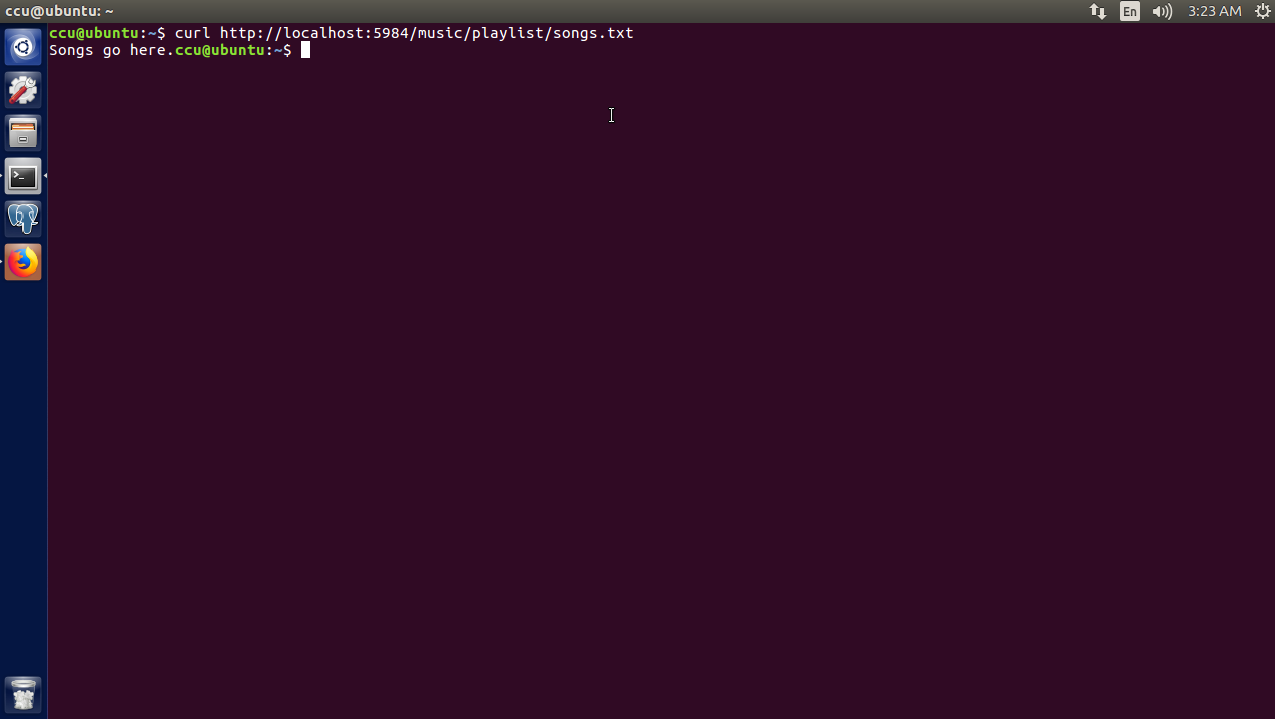




When I first tried creating a database, I was getting an error that said “You are not a server admin.”. I found out that to create a database, you have to be a server admin, but creating and modifying documents does not require you to be one. The process to become an admin was easy, as I just went to the account section of Fauxton (no longer Futon), and created an admin account. Now, when creating a database, I just had to use the standard PUT command with my admin username and password in the URL, like so: “http://admin:password@localhost:5984/database”. Deleting the database also required using a username and password.

3. Again using cURL, create a new document that contains a text document as an attachment. Lastly, craft and execute a cURL request that will return just that document’s attachment.



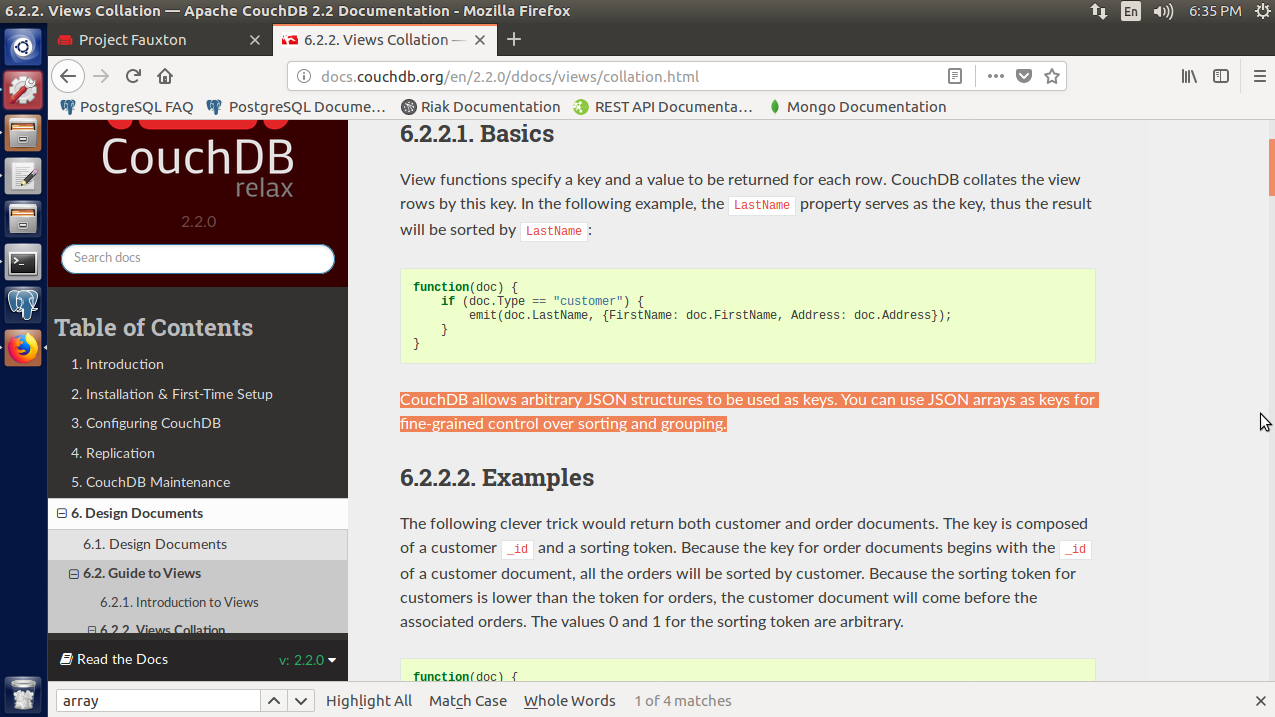


I had to do some research to learn how to create a document with an attachment. All it involves is adding the **“\_attachments”** field in the data portion of the JSON and under that field, adding the attachment name and the content type and data of the attachment. Since I used plain text as the attachment type, its data had to be encoded in Base64, so I went to base64encode.org and encoded the contents to be used in the file. To display just the attachment contents, I used a standard cURL GET request and pointed to the name of the attachment.

**Day 2**

Find:

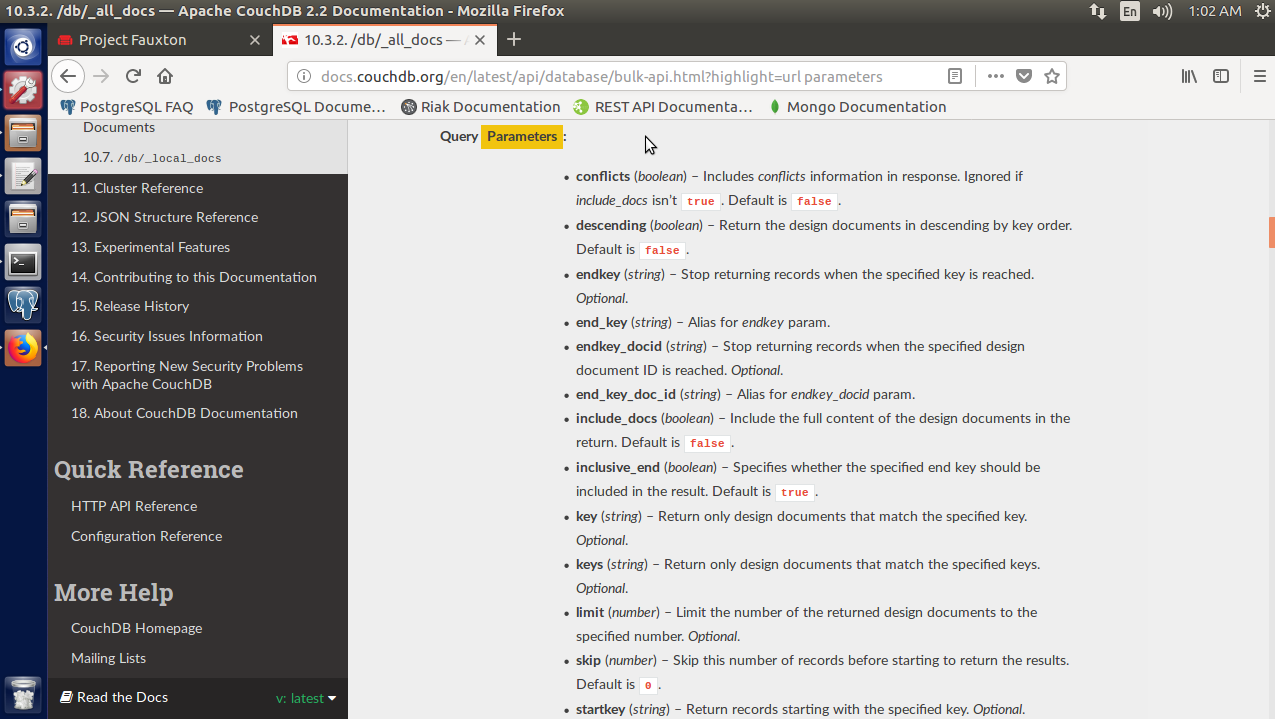
1. We’ve seen that the emit() method can output keys that are strings. What other types of values does it support? What happens when you emit an array of values as a key?



According to the documentation under 6. Design Documents -> 6.2 Guide to Views, any JSON object can be used as a key, including arrays which offer greater control over grouping/sorting.

Unfortunately, I was unable to find a handy list of supported values, however, from all of the examples I have seen, **emit()** can emit anything as long as it is a key/value pair in the document.

2. Find a list of available URL parameters (like limit and startkey) that can be appended to view requests and what they do.



A list of valid query parameters can be found in the CouchDB docs by searching for “query parameters”, selecting section 10.3.2 and scrolling down to section 10.3.3, which covers design documents. What the book calls “URL parameters” seem to be more accurately referred to as “query parameters” around the internet, so it was hard to find information on them until I discovered that.

Do:

1. The import script import\_from\_jamendo.rb assigned a random number to each artist by adding a property called random. Create a mapper function that will emit key-value pairs where the key is the random number and the value is the band’s name. Save this in a new design document named \_design/random with the view name artist.

I could not import the Jamendo data, as something was wrong with the libxml package, so I could not complete this task as well as the next two.

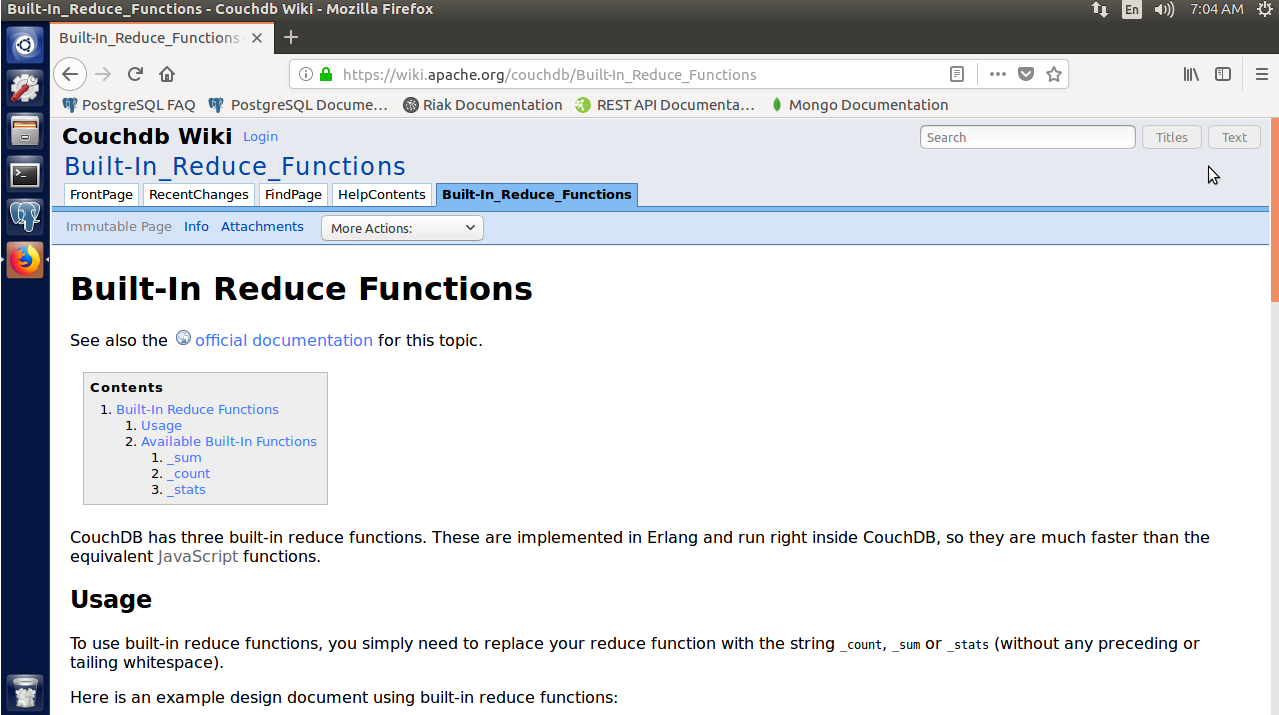
2. Craft a cURL request that will retrieve a random artist.

3. The import script also added a random property for each album, track, and tag. Create three additional views in the \_design/random design document with the view names album, track, and tag to match the earlier artist view.

**Day 3**

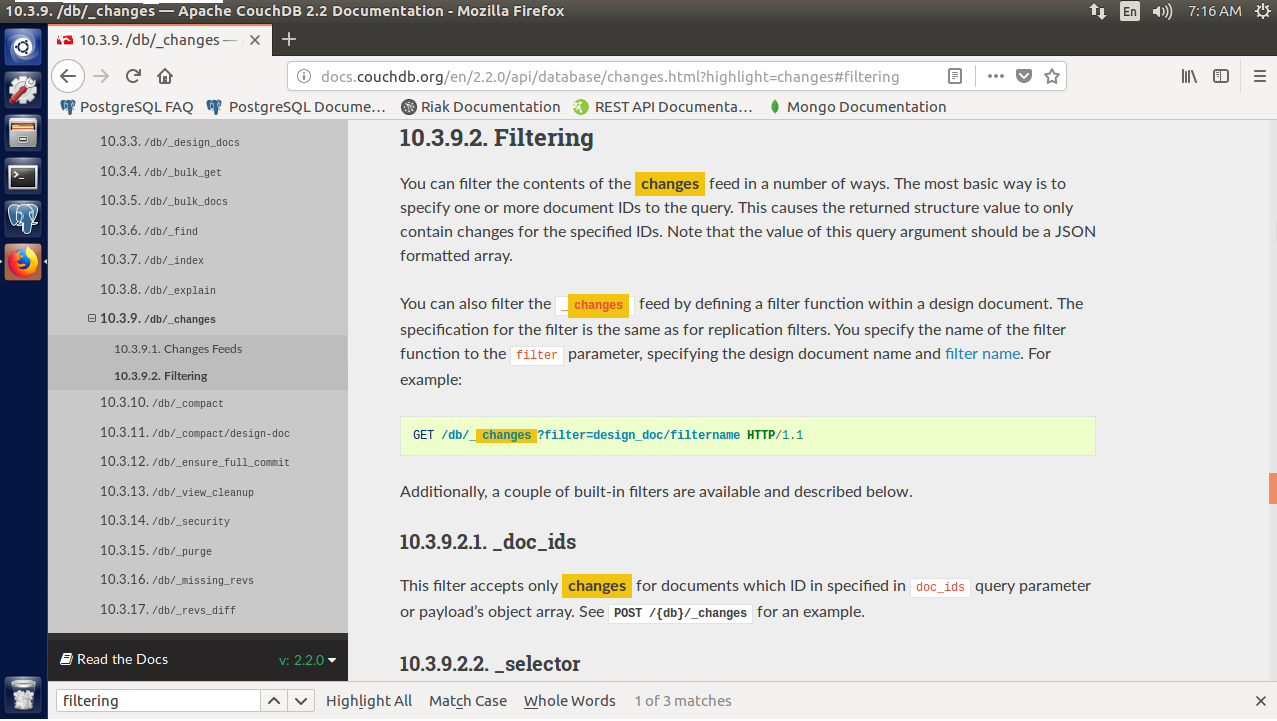
Find:

1. What native reducers are available in CouchDB? What are the benefits of using native reducers over custom JavaScript reducers?



The CouchDB wiki has a list of native reduce functions. According to the wiki page, native reducers run inside of CouchDB unlike JavaScript, so the benefit is that they are faster.

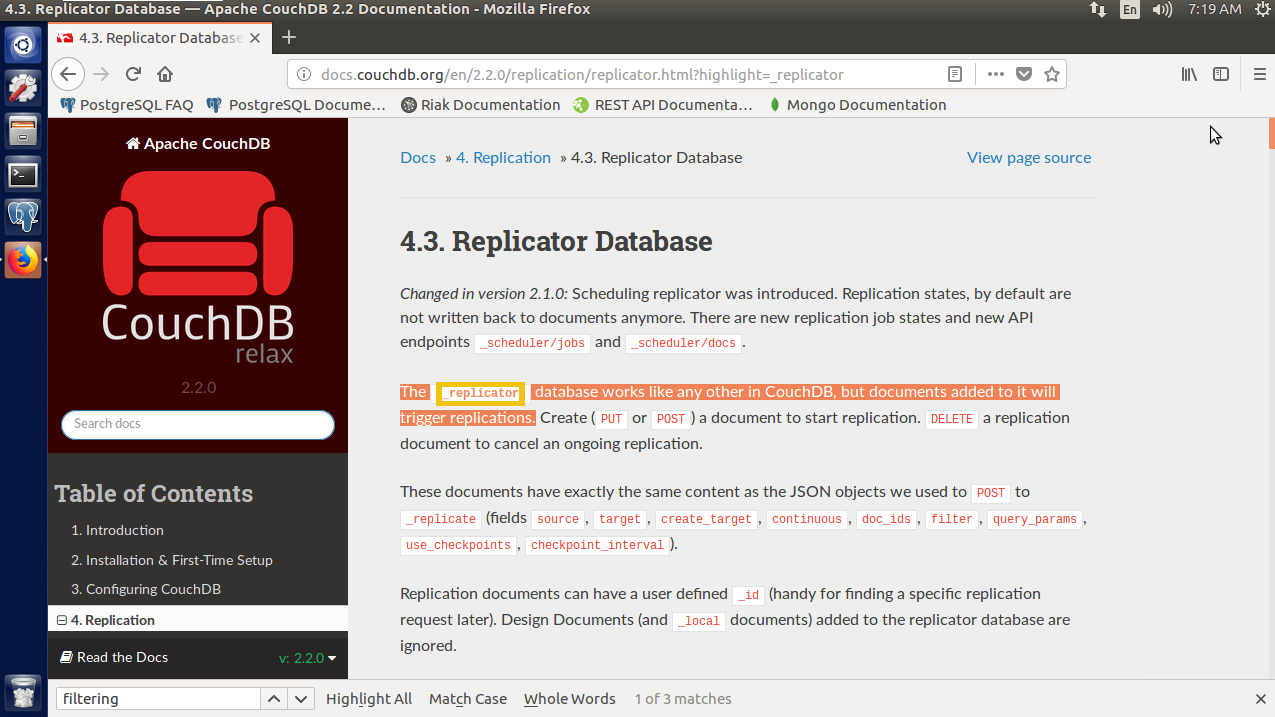
2. How can you filter the changes coming out of the \_changes API on the server side?



A list of filtering methods can be found in section 10.3.9.2 of the CouchDB docs.

3. Like everything else in CouchDB, the tasks of initializing and canceling replication are controlled by HTTP commands under the hood. What are the REST commands to set up and remove replication relationships between servers?

4. How can you use the \_replicator database to persist replication relationships?



According to the documentation, documents inside the **\_replicator** database trigger replications, so all you would have to do is put the document in this database.