



PouchDB

tutorialspoint

SIMPLY EASY LEARNING

www.tutorialspoint.com



<https://www.facebook.com/tutorialspointindia>



<https://twitter.com/tutorialspoint>

About this Tutorial

PouchDB is an open source **in-browser database API** written in JavaScript. It is modelled after CouchDB– a NoSQL database that powers npm. Using this API, we can build applications that work offline and online. PouchDB uses WebSQL and IndexedDB internally to store the data.

This tutorial discusses the basics of PouchDB along with relevant examples for easy understanding.

Audience

This tutorial has been prepared for beginners to help them understand the basic concepts of PouchDB. It will aid you to build applications which will work offline and online alike using PouchDB and CouchDB.

Prerequisites

The reader should have a basic knowledge of databases. It would be better to have a good command on programming languages, which are compatible with node.js such as JavaScript and CoffeeScript.

Copyright & Disclaimer

© Copyright 2016 by Tutorials Point (I) Pvt. Ltd.

All the content and graphics published in this e-book are the property of Tutorials Point (I) Pvt. Ltd. The user of this e-book is prohibited to reuse, retain, copy, distribute or republish any contents or a part of contents of this e-book in any manner without written consent of the publisher.

We strive to update the contents of our website and tutorials as timely and as precisely as possible, however, the contents may contain inaccuracies or errors. Tutorials Point (I) Pvt. Ltd. provides no guarantee regarding the accuracy, timeliness or completeness of our website or its contents including this tutorial. If you discover any errors on our website or in this tutorial, please notify us at contact@tutorialspoint.com

Table of Contents

About this Tutorial	i
Audience	i
Prerequisites	i
Copyright & Disclaimer	i
Table of Contents	ii
1. POUCHDB – OVERVIEW	1
What is PouchDB?	1
How Does it Work?	1
Features of PouchDB	1
Advantages of PouchDB	1
Browsers that Support PouchDB	2
2. POUCHDB – ENVIRONMENT	3
Installing PouchDB	3
Installing Pouch Using Node.js	4
Downloading CouchDB	4
Installing CouchDB	6
3. POUCHDB – CREATE DATABASE	7
4. POUCHDB – DATABASE INFO	8
Remote Database Info	9
5. POUCHDB – DELETE DATABASE	11
Deleting a Remote Database	12
6. POUCHDB – CREATE DOCUMENT	14
Inserting a Document in a Remote Database	15

7. POUCHDB – READ DOCUMENT.....	18
Reading a Document from a Remote Database	19
8. POUCHDB – UPDATE DOCUMENT	22
Updating a Document in a Remote Database	24
9. POUCHDB – DELETE DOCUMENT	27
Deleting a Document from a Remote Database	28
10. POUCHDB – CREATE BATCH	31
Inserting a Batch in a Remote Database	32
11. POUCHDB – FETCH BATCH	35
Reading a Batch from a Remote Database	37
12. POUCHDB – UPDATE BATCH.....	40
Updating Batch from a Remote Database	42
13. POUCHDB – DELETE BATCH.....	47
Deleting Batch from a Remote Database.....	49
14. POUCHDB – ADDING ATTACHMENT	52
Adding Attachment to an Existing Document	53
Adding Attachment to a Remote Document.....	55
15. POUCHDB – RETRIEVING ATTACHMENT	59
Retrieving Attachment from a Remote Document	60
16. POUCHDB – DELETING ATTACHMENT	63
Removing Attachment from a Remote Document.....	65

17. POUCHDB – REPLICATION	69
Replicating LocalDB to CouchDB.....	69
Replicating CouchDB to PouchDB	71
18. POUCHDB – SYNCHRONIZATION.....	74
19. POUCHDB – MISCELLANEOUS	79
Compaction.....	79
BulkGet Method.....	Error! Bookmark not defined.

1. PouchDB – Overview

This chapter provides a brief introduction to PouchDB along with its features and how it works.

What is PouchDB?

PouchDB is an open source **in-browser database API** written in JavaScript. It is modelled after [Couch DB](#) – a NoSQL database. Using this API, we can build applications that work offline and online. It internally uses WebSQL and IndexedDB to store data.

How Does it Work?

In PouchDB, when the application is offline, the data is stored locally using WebSQL and IndexedDB in the browser. When the application is back online, it is synchronized with CouchDB and compatible servers.

Using PouchDB, you can communicate with both local and remote databases seamlessly without noticing any difference.

Features of PouchDB

Following are the features of PouchDB -

- **Cross Browser:** The API provided by PouchDB works the same in every environment, therefore, we can run a PouchDB application in various browsers.
- **Light Weight:** PouchDB is a very light-weight API, it is also included easily just using a script tag.
- **Easy to Learn:** If you have a prior knowledge of any programming language, it is easy to learn PouchDB.
- **Open Source:** PouchDB is an Open Source Application and is available on GitHub.

Advantages of PouchDB

Following are the advantages of PouchDB -

- Since PouchDB resides inside the browser, there is no need to perform queries over the network, this results in faster execution of queries.
- You can synchronize the data with any of the supported server and by doing so you can run apps both online and offline.

Browsers that Support PouchDB

Following are the browsers that support PouchDB -

- Firefox 29+ (Including Firefox OS and Firefox for Android)
- Chrome 30+
- Safari 5+
- Internet Explorer 10+
- Opera 21+
- Android 4.0+
- iOS 7.1+
- Windows Phone 8+

2. PouchDB – Environment

This chapter explains how to download and install PouchDB in your system.

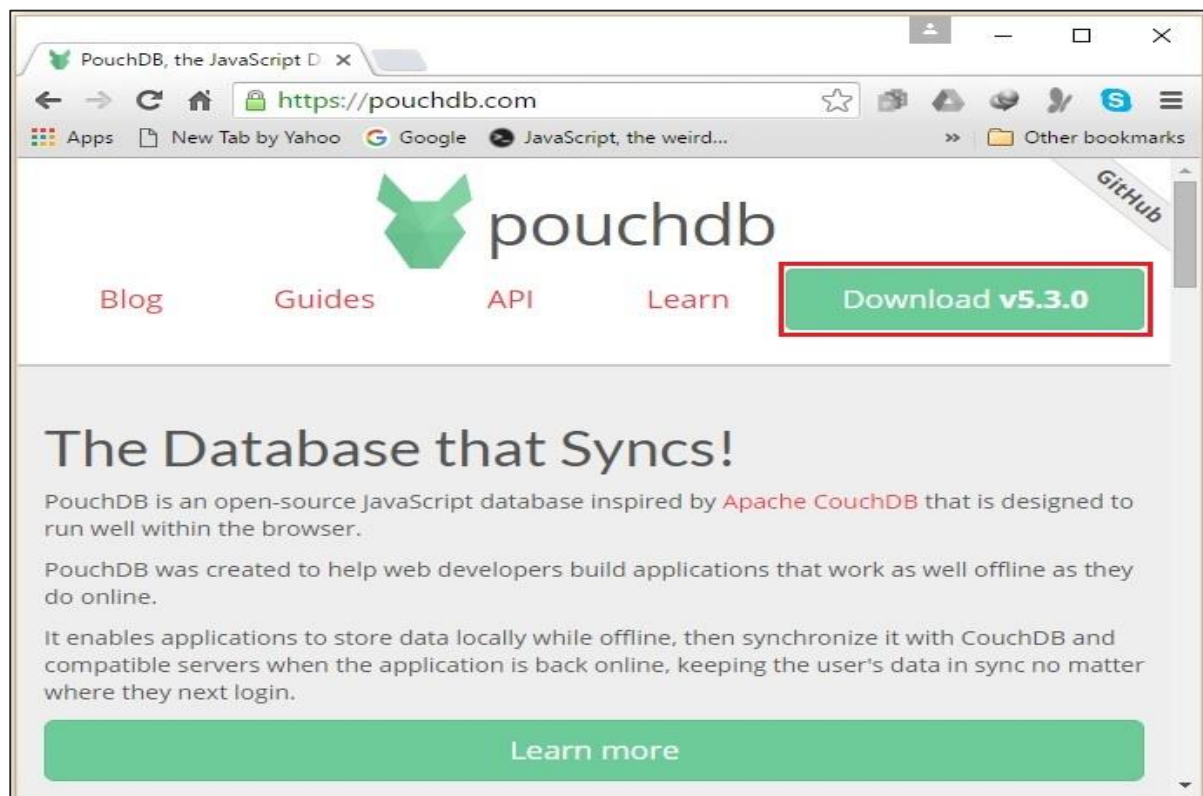
Installing PouchDB

In order to work with PouchDB, you need to download the file **.js** file and include it in your script. Following are the steps to install PouchDB.

Step1

Visit the homepage of PouchDB website, by clicking the following link –

<https://PouchDB.com/>



Step2

Click the Download button on the top right hand side of the web page as shown in the above screenshot. This will download **PouchDB-5.3.0.min.js** in your system.

Step3

Copy and paste the **PouchDB-5.3.0.min.js** to your working directory and include it in your JavaScript as shown in the following command.

```
<scriptsrc="PouchDB-5.3.0.min.js"></script>
```

Installing Pouch Using Node.js

You can also install PouchDB as Node.js module. Following are the steps to install PouchDB using Node.js.

Step1

Install Node.js by following the steps given in the Installing Node.js section of our [coffee script](#) tutorial.

Step2

Open the command prompt and execute the following command. This will install PouchDB node module in your system.

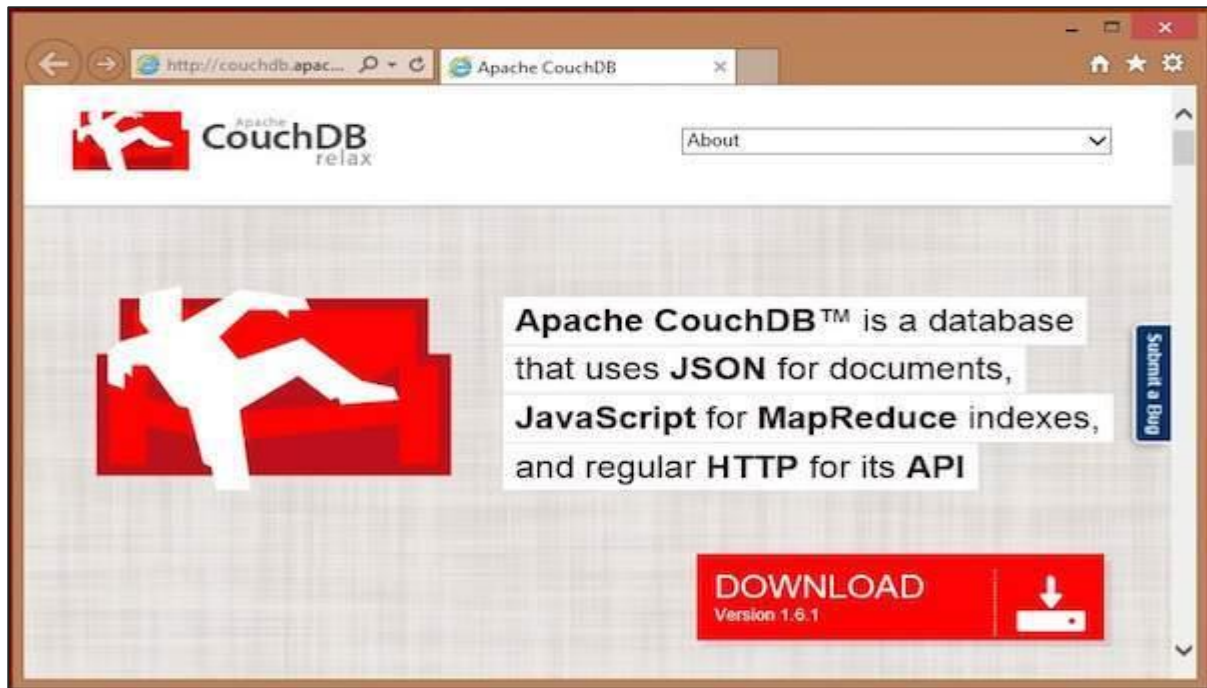
```
npm install --save PouchDB
```

Downloading CouchDB

When offline, PouchDB stores data locally and works like an app. You can access it online by connecting with compatible servers. As we know PouchDB can be connected to CouchDB, so, lets install CouchDB too. Following are the steps to install CouchDB.

Step1

The official website for CouchDB is <http://couchdb.apache.org>. If you click the given link, you can get the home page of CouchDB official website as shown in the following screenshot.



Step 2

If you click on the download button that will lead to a page where the download links of CouchDB in various formats are provided. The following snapshot illustrates the same.



Step 3

Choose the download link for Windows Systems and select one of the provided mirrors to start your download.

Installing CouchDB

A windows executable **setup-couchdb-1.6.1_R16B02.exe** file will be downloaded on your system. Run the setup file and proceed with the installation.

After installing CouchDB in your system successfully, open the folder where CouchDB was installed, go to the bin folder, and start the server by running a script file named **couchdb.bat**.

After installation, open built-in web interface of CouchDB by visiting the following link: <http://127.0.0.1:5984/>. If everything goes fine, this will give you a web page, which will have the following output.

```
{  "couchdb":"Welcome","uuid":"c8d48ac61bb497f4692b346e0f400d60",
  "version":"1.6.1",
  "vendor":{"
    "version":"1.6.1","name":"The Apache Software Foundation"
  }}
```

You can interact with CouchDB web interface by using the following URL –

```
http://127.0.0.1:5984/_utils/
```

This shows you the index page of Futon, which is the web interface of CouchDB.



3. PouchDB– Create Database

You can create a database in PouchDB using the PouchDB constructor.

Syntax

Following is the syntax of using the PouchDB constructor. To this, you need to pass the name of the database as a parameter.

```
New PouchDB(Database_name)
```

Example

To create a database in PouchDB using **node**, first of all, you need to require the PouchDB package using the **require()** method and then you can create a database as shown in the following example.

```
//Requiring the package
var PouchDB = require('PouchDB');

//Creating the database object
var db = new PouchDB('my_database');
console.log ("Database created Successfully.");
```

Save the above code in a file with the name **Create_Database.js**. Open the command prompt and execute the JavaScript file using **node** as shown below.

```
C:\PouchDB_Examples>node Create_Database.js
```

This will create a database locally (you can see the folder in the current directory) displaying the following message.

```
Database created Successfully.
```

4. PouchDB –Database Info

You can get the basic information about the database using the method named **info()**

Syntax

Following is the syntax of using the **info()** method of PouchDB. This method accepts a callback function.

```
db.info([callback])
```

Example

Following is an example of retrieving database information using the **info()** method. Here, we are displaying the information of the database named **my_database**. In case of error, the error will be displayed on the console.

```
//Requiring the package
var PouchDB = require('PouchDB');

//Creating the database object
var db = new PouchDB('my_database');

//Database information
db.info(function(err, info) {
  if (err) {
    return console.log(err);
  } else {
    console.log(info);
  }
});
```

Save the above code in a file with the name **Database_info.js**. Open the command prompt and execute the JavaScript file using **node** as shown below.

```
C:\PouchDB_Examples>node Database_info.js
```

This will display the info of the specified database as follows.

```
{ doc_count: 0,
  update_seq: 0,
  backend_adapter: 'LevelDOWN',
  db_name: 'my_database',
  auto_compaction: false,
  adapter: 'leveldb' }
```

Remote Database Info

In the same way, you get the information of a database that is saved remotely on the server (CouchDB). To do so, instead of a database name you need to pass the path to the required database in CouchDB.

Example

Following is an example of retrieving information of a database that is saved in the CouchDB server. This code gives you information of a database named **my_database**.

```
//Requiring the package
var PouchDB = require('PouchDB');

//Creating the database object
var db = new PouchDB('http://localhost:5984/my_database');

//Database information
db.info(function(err, info) {
  if (err) {
    return console.log(err);
  } else {
    console.log(info);
  }
});
```

Save the above code in a file with the name **Database_Remote_info.js**. Open the command prompt and execute the JavaScript file using **node** as shown below.

```
C:\PouchDB_Examples>node Database_remote_info.js
```

This will display the info of the specified database as follows.

```
{ db_name: 'my_database',
```

```
doc_count: 0,  
doc_del_count: 0,  
update_seq: 0,  
purge_seq: 0,  
compact_running: false,  
disk_size: 79,  
data_size: 0,  
instance_start_time: '1458209191708486',  
disk_format_version: 6,  
committed_update_seq: 0,  
host: 'http://localhost:5984/my_database/',  
auto_compaction: false,  
adapter: 'http' }
```

5. PouchDB –Delete Database

You can delete a database in PouchDB using the **db.destroy()** method.

Syntax

Following is the syntax of using the **db.destroy()** method. This method accepts a callback function as a parameter.

```
db.destroy()
```

Example

Following is an example of deleting a database in PouchDB using the **destroy()** method. Here, we are deleting the database named **my_database**, created in the previous chapters.

```
//Requiring the package
var PouchDB = require('PouchDB');

//Creating the database object
var db = new PouchDB('my_database');

//deleting database
db.destroy(function (err, response) {
  if (err) {
    return console.log(err);
  } else {
    console.log ("Database Deleted");
  }
});
```

Save the above code in a file with the name **Delete_Database.js**. Open the command prompt and execute the JavaScript file using **node** as shown below.

```
C:\PouchDB_Examples >node Delete_Database.js
```

This will delete the database named **my_database** which is stored locally displaying the following message.

```
Database Deleted
```


Deleting a Remote Database

In the same way, you can delete a database that is stored remotely on the server (CouchDB).

To do so, instead of a database name, you need to pass the path to the database that is required to be deleted, in CouchDB.

Example

Suppose there is a database named **my_database** in the CouchDB server. Then, if you verify the list of databases in CouchDB using the URL http://127.0.0.1:5984/_utils/index.html you will get the following screenshot.



Following is an example of deleting a database named **my_database** that is saved in the CouchDB server.

```
//Requiring the package
var PouchDB = require('pouchdb');

//Creating the database object
var db = new PouchDB('http://localhost:5984/my_database');

//deleting database
db.destroy(function (err, response) {
  if (err) {
    return console.log(err);
  } else {
  }
```

```

        console.log("Database Deleted");
    }
});

```

Save the above code in a file with the name **Remote_Database_Delete.js**. Open the command prompt and execute the JavaScript file using **node** as shown below.

```
C:\PouchDB_Examples>node Remote_Database_Delete.js
```

This deletes the specified database from PouchDB displaying the following message.

```
Database Deleted
```

Verification

After executing the above program, if you visit the URL again, you will get the following screenshot. Here you can observe only two databases since **my_database** was deleted.



6. PouchDB – Create Document

You can create a document in PouchDB using the **db.put()** method.

Syntax

Following is the syntax of using the db.put() method of PouchDB. You can store the document that is to be created in PouchDB, in a variable and pass as a parameter to this method. In addition, this method also accepts a callback (optional) function as a parameter.

```
db.put(document, callback)
```

Example

Following is an example of creating a document in PouchDB using the **put()** method. The document we create should be of JSON format, a set of key-value pairs separated by comma (,) and enclosed within curly braces ({}).

```
//Requiring the package
var PouchDB = require('PouchDB');
//Creating the database object
var db = new PouchDB('my_database');
//Preparing the document
doc = {
    _id : '001',
    name: 'Raju',
    age : 23,
    designation : 'Designer'
}
//Inserting Document
db.put(doc, function(err, response) {
    if (err){
        return console.log(err);
    } else{
        console.log("Document created Successfully");
    }
});
```

Save the above code in a file with name **Create_Document.js**. Open the command prompt and execute the JavaScript file using **node** as shown below.

```
C:\PouchDB_Examples >node Create_Document.js
```

This creates the given document in PouchDB database named **my_database**, which is stored locally, displaying the following message.

```
Document created Successfully
```

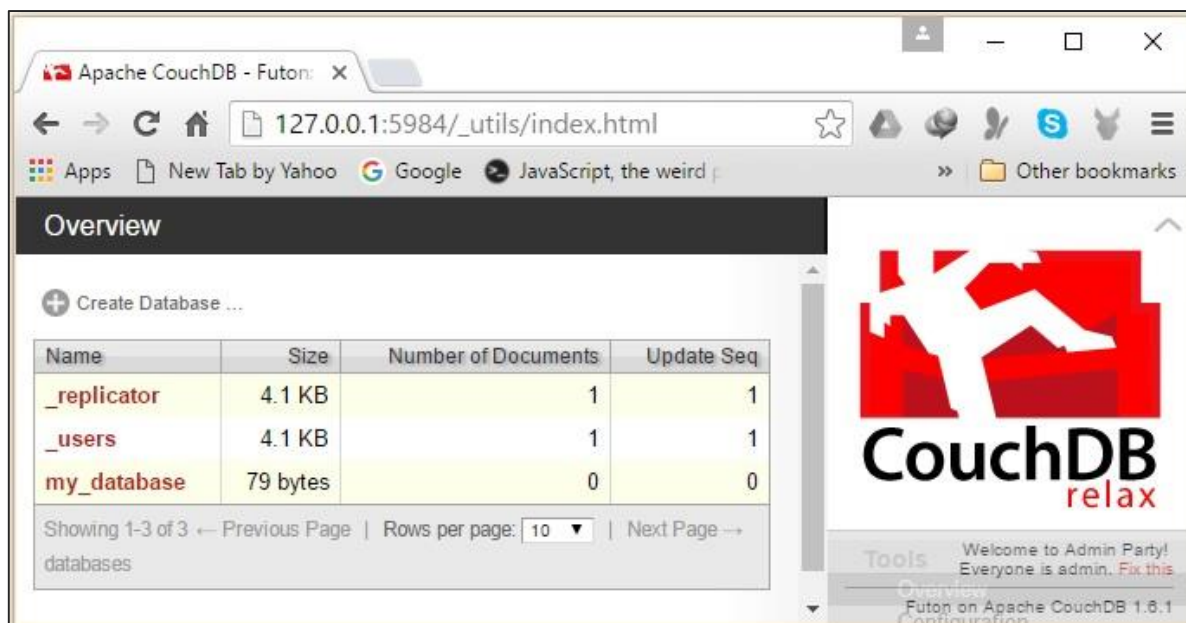
Inserting a Document in a RemoteDatabase

You can also insert a document in the database that is stored remotely on the server (CouchDB).

To do so, instead of database name you need to pass the path to the database where you want to create documents in CouchDB.

Example

Suppose there is a database named **my_database** in the CouchDB server. Then, if you verify the list of databases in CouchDB using the URL http://127.0.0.1:5984/_utils/index.html you will get the following screenshot.



Now, if you click on the database named **my_database**, you will find an empty database as shown in the following screenshot.



Following is an example of inserting a document in a database named **my_database** that is saved in the CouchDB server.

```
//Requiring the package
var PouchDB = require('PouchDB');

//Creating the database object
var db = new PouchDB('http://localhost:5984/my_database');
//Preparing the document
doc = {
  _id : '001',
  name: 'Raju',
  age : 23,
  designation : 'Designer'
}
//Inserting Document
db.put(doc, function(err, response) {
  if (err){
    return console.log(err);
  } else{
    console.log("Document created Successfully");
  }
});
```

Save the above code in a file with the name **Remote_Create_Document.js**. Open the command prompt and execute the JavaScript file using **node** as shown below.

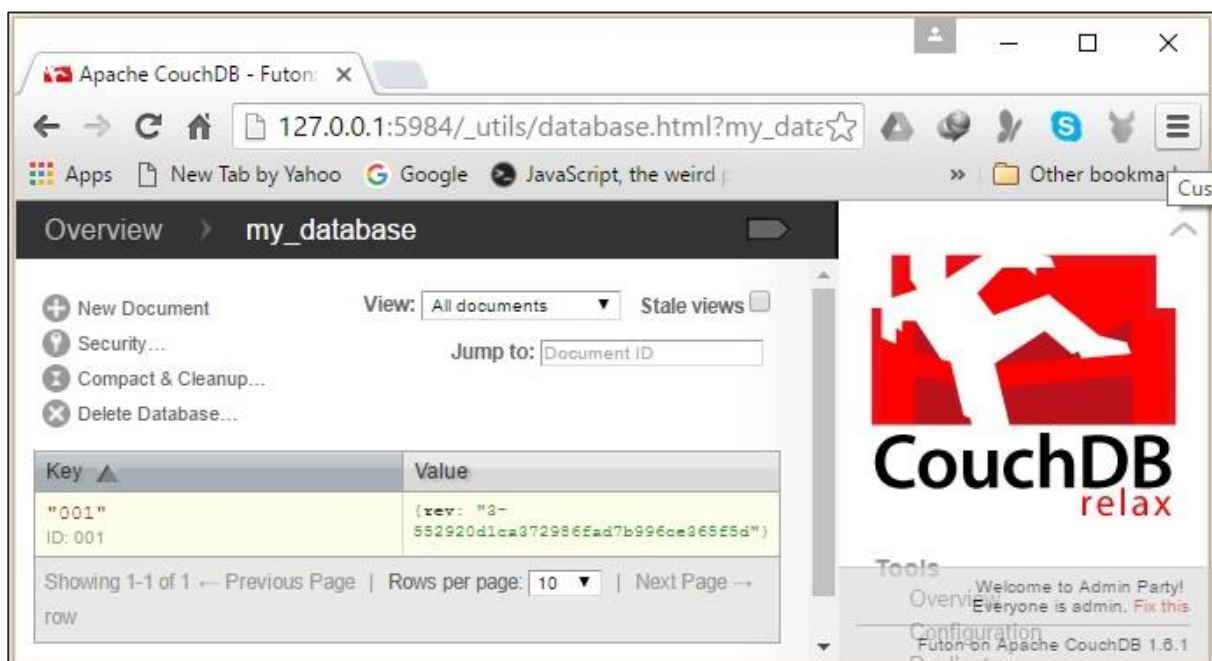
```
C:\PouchDB_Examples >node Remote_Create_Document.js
```

This creates the given document in PouchDB database named **my_database** which is stored in CouchDB, displaying the following message.

Document created Successfully

Verification

After executing the above program, if you visit the **my_database** again, you can observe the document created as shown in the following screenshot.



7. PouchDB – Read Document

You can read/retrieve the contents of a document in PouchDB using the **db.get()** method.

Syntax

Following is the syntax of using the **db.get()** method of PouchDB. This method accepts the **document id** and an optional callback function.

```
db.get (document, callback)
```

Example

Following is an example of reading the contents of a document in PouchDB using the **get()** method.

```
//Requiring the package
var PouchDB = require('PouchDB');

//Creating the database object
var db = new PouchDB('my_database');

//Reading the contents of a Document
db.get('001', function(err, doc) {
  if (err){
    return console.log(err);
  } else{
    console.log(doc);
  }
});
```

Save the above code in a file with name **Read_Document.js**. Open the command prompt and execute the JavaScript file using node as shown in the following command.

```
C:\PouchDB_Examples >node Read_Document.js
```

This reads the contents of the given document that exists in the database named **my_database** which is stored locally. The following message gets displayed on the console.


```
{name: 'Raju',
age: 23,
designation: 'Designer',
  _id: '001',
  _rev: '1-ba7f6914ac80098e6f63d2bfb0391637'}
```

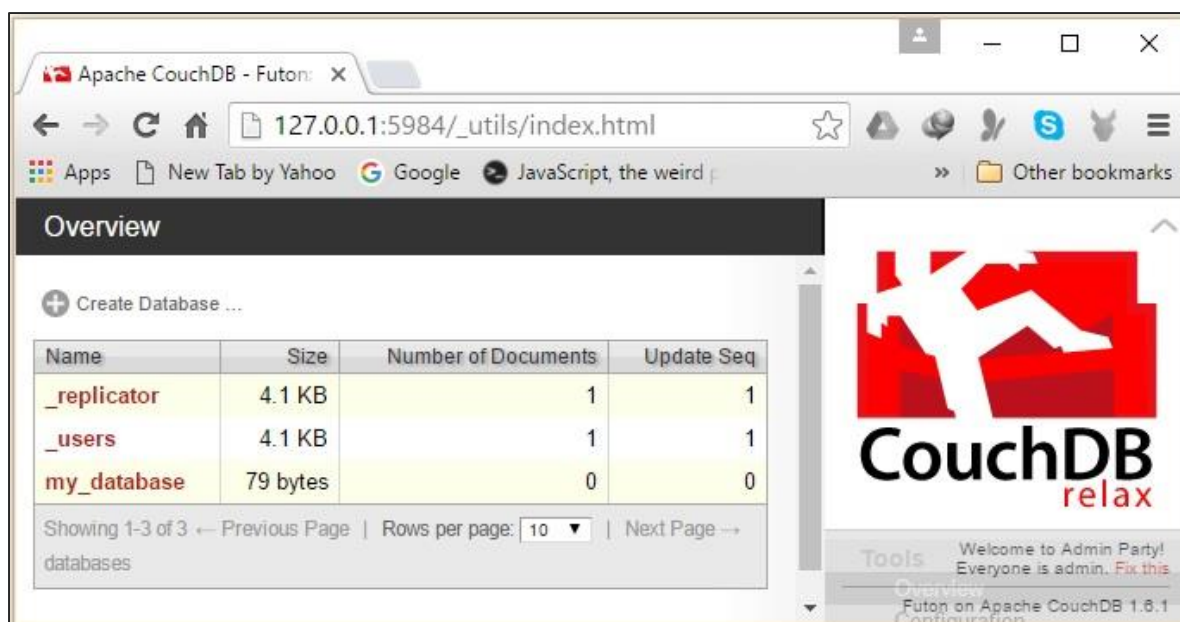
Reading a Document from a Remote Database

You can also read a document from the database that is stored remotely on the server (CouchDB).

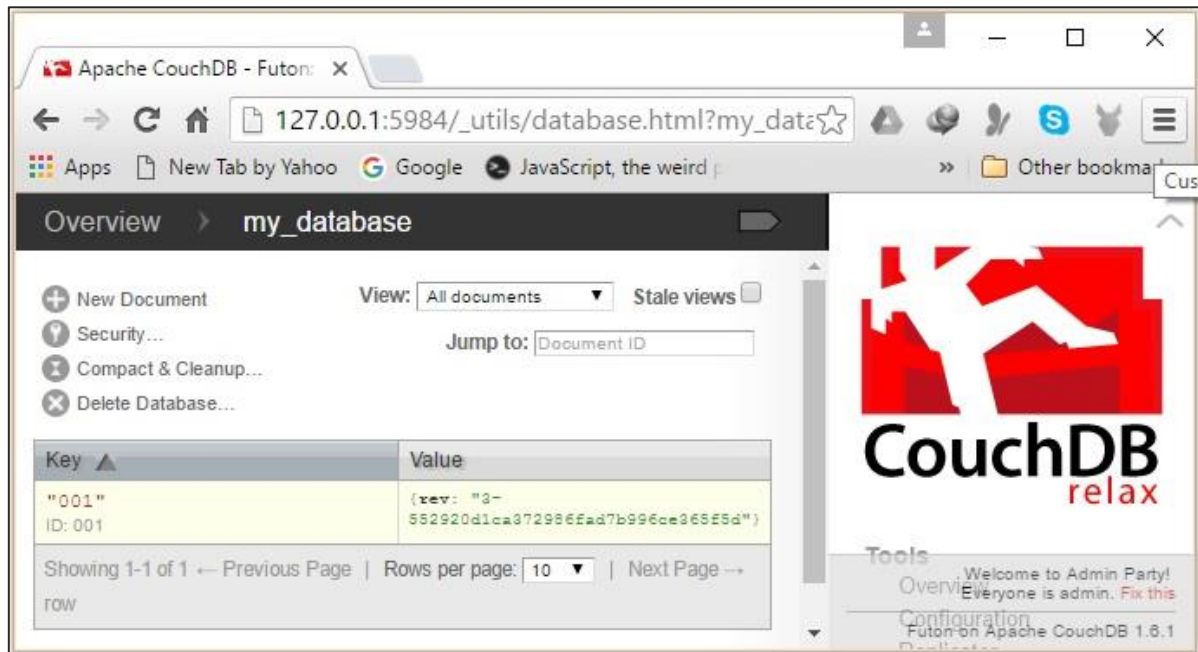
To do so, instead of a database name, you need to pass the path to the database in CouchDB, which contains the document that is to be read.

Example

Suppose, there is a database named **my_database** in the CouchDB server. Then, if you verify the list of databases in CouchDB using the URL http://127.0.0.1:5984/_utils/index.html you will get the following screenshot.



By clicking on the database named **my_database** you can see the following screenshot. Here, you can observe that this database contains a document with id **001**.



Following is an example of reading the contents of the document having id as "001" that exists in a database named **my_database**, which is stored in the CouchDB server.

```
//Requiring the package
var PouchDB = require('PouchDB');

//Creating the database object
var db = new PouchDB('http://localhost:5984/my_database');

//Reading the contents of a document
db.get('001', function(err, doc) {
  if (err){
    return console.log(err);
  } else{
    console.log(doc);
  }
});
```

Save the above code in a file with the name **Remote_Read_Document.js**. Open the command prompt and execute the JavaScript file using **node** as shown below.

```
C:\PouchDB_Examples >node Remote_Read_Document.js
```

This reads the contents of the given document that exists in the database named **my_database** which is stored in CouchDB. The following message is displayed on the console.

```
{ _id: '001',  
  _rev: '3-552920d1ca372986fad7b996ce365f5d',  
  name: 'Raju',  
  age: 23,  
  designation: 'Designer' }
```

8. PouchDB – Update Document

Whenever, we create a document in PouchDB, a new field **`_rev`** is generated, and it is known as **revision marker**. The **`_rev`**'s value is a unique random number, each time we make changes to the document the value of **`_rev`** is changed.

You can update an existing document in PouchDB using the (**`_rev`**). To do so, first of all retrieve the **`_rev`** value of the document we want to update. Now, place the contents that are to be updated along with the retrieved **`_rev`** value in a new document, and finally insert this document in PouchDB using the **`put()`** method.

Example

Assume we have a document in PouchDB with id **`001`** which has details of a person. In order to update this document, we should have its rev number. Therefore, to retrieve the contents of the document the following code is used.

```
//Requiring the package
var PouchDB = require('PouchDB');

//Creating the database object
var db = new PouchDB('my_database');

//Reading the contents of a Document
db.get('001', function(err, doc) {
  if (err){
    return console.log(err);
  } else{
    console.log(doc);
  }
});
```

On executing the above code, you will receive the following output.

```
{ _id: '001',
  _rev: '3-552920d1ca372986fad7b996ce365f5d',
  name: 'Raju',
  age: 23,
  designation: 'Designer' }
```

Now, using the **_rev** you can update the value of the key **"age"** to 26, as shown in the following code.

```
//Requiring the package
var PouchDB = require('PouchDB');

//Creating the database object
var db = new PouchDB('my_database');
//Preparing the document for update
doc = {
  age: 26,
  _rev: '3-552920d1ca372986fad7b996ce365f5d',
}

//Inserting Document
db.put(doc);

//Reading the contents of a Document
db.get('001', function(err, doc) {
  if (err){
    return console.log(err);
  } else{
    console.log(doc);
  }
});
```

Save the above code in a file with the name **Update_Document.js**. Open the command prompt and execute the JavaScript file using **node** as shown below.

```
C:\Pouch_Examples>node Update_Document.js
```

This updates the contents of the given document that exists in the database named **my_database** which is stored locally. The following message is displayed on the console.

```
{ name: 'Raju',
  age: 26,
  designation: 'Designer',
  _id: '001',
  _rev: '2-61b523ccdc4e41a8435bdfbb057a7a5' }
```

Updating a Document in a Remote Database

You can also update an existing document in a database that is stored remotely on the server (CouchDB).

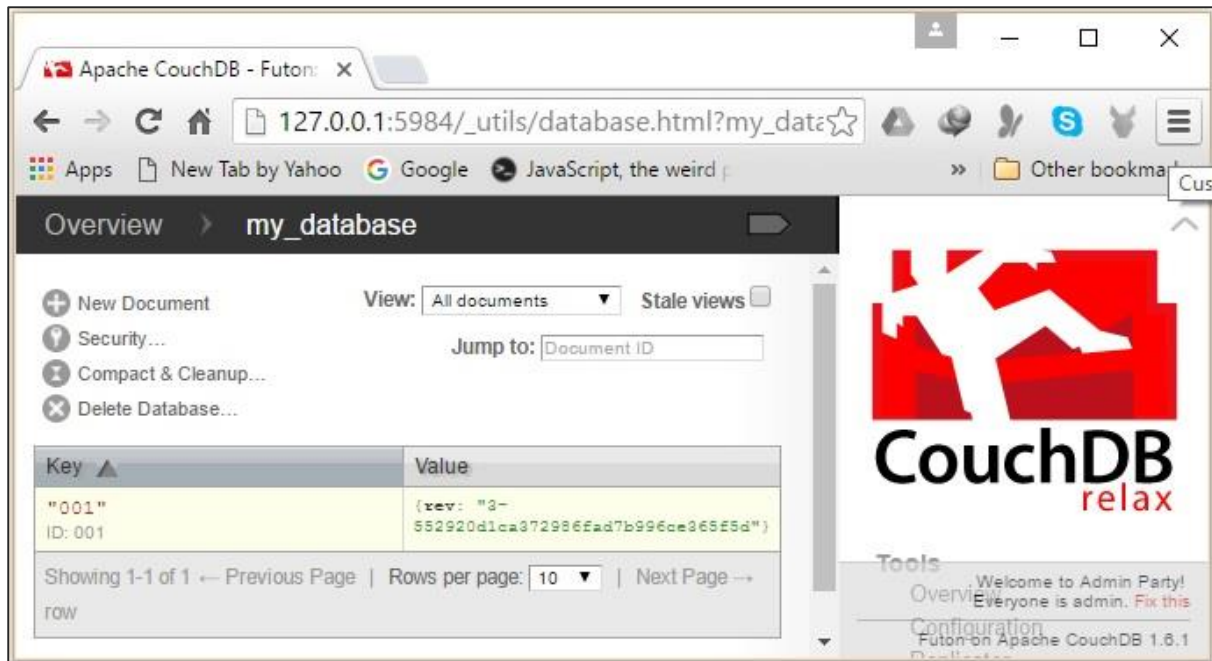
To do so, instead of a database name you need to pass the path to the database in CouchDB, which contains the document that is to be updated.

Example

Suppose there is a database named **my_database** in the CouchDB server. Then, if you verify the list of databases in CouchDB using the URL http://127.0.0.1:5984/_utils/index.html you will get the following screenshot.



By clicking on the database named **my_database**, you can see the following screenshot. Here, you can observe that this database contains a document with id **001**.



Following is an example of updating the age of the document having id as **"001"** that exists in a database named **my_database** which is stored in the CouchDB server.

```
//Requiring the package
var PouchDB = require('PouchDB');

//Creating the database object
var db = new PouchDB('http://localhost:5984/my_database');

//Preparing the document for update
doc = {
  age: 26,
  _rev: '3-552920d1ca372986fad7b996ce365f5d',
}

//Inserting Document
db.put(doc);

//Reading the contents of a Document
db.get('001', function(err, doc) {
  if (err){
    return console.log(err);
  } else{
    console.log(doc);
  }
});
```

Save the above code in a file with the name **Remote_Update_Document.js**. Open the command prompt and execute the JavaScript file using **node** as shown below.

```
C:\PouchDB_Examples >node Remote_Update_Document.js
```

This updates the contents of the given document that exists in the database named **my_database** which is stored in CouchDB. The following message is displayed on the console.

```
{ _id: '001',  
  _rev: '2-b9640bffbce582c94308905eed8bb545',  
  name: 'Raju',  
  age: 26,  
  designation: 'Designer' }
```

9. PouchDB – Delete Document

You can delete a document from a database that exists in PouchDB using the **db.remove()** method.

Syntax

Following is the syntax of using the **db.remove()** method of PouchDB. To this method, we have to pass **id** and **_rev** to delete an existing document as shown in the following code. This method accepts an optional callback function. We can also pass the complete document instead of id and _rev.

```
db.remove( docId, docRev, [callback] )
```

Example

Assume we have a document in PouchDB with id **001** which have the details of a person. In order to delete this document along with its **id** we should also have its **_rev** number. Therefore, retrieve the contents of the document as shown in the following code.

```
//Requiring the package
var PouchDB = require('PouchDB');

//Creating the database object
var db = new PouchDB('my_database');

//Reading the contents of a Document
db.get('001', function(err, doc) {
  if (err) {
    return console.log(err);
  } else {
    console.log(doc);
  }
});
```

Executing the above code gives the following output.

```
{ _id: '001',
```



```
_rev: '3-552920d1ca372986fad7b996ce365f5d',  
name: 'Raju',  
age: 23,  
designation: 'Designer' }
```

Now, using the **_rev** and id of the document you can delete this by using the **remove()** method as shown in the following code.

```
//Requiring the package  
var PouchDB = require('PouchDB');  
  
//Creating the database object  
var db = new PouchDB('my_database');  
  
//Deleting an existing document  
db.remove('001', '3-552920d1ca372986fad7b996ce365f5d', function(err) {  
  if (err) {  
    return console.log(err);  
  } else {  
    console.log("Document deleted successfully");  
  }  
});
```

Save the above code in a file with the name **Delete_Document.js**. Open the command prompt and execute the JavaScript file using **node** as shown below.

```
C:\PouchDB_Examples >node Delete_Document.js
```

This deletes the contents of the given document that exists in the database named **my_database** which is stored locally. The following message is displayed.

```
Document deleted successfully
```

Deleting a Document from a Remote Database

You can also delete an existing document from the database that is stored remotely on the server (CouchDB).

To do so, instead of a database name, you need to pass the path to the database in CouchDB, which contains the document that is to be read.

Example

Suppose there is a database named **my_database** in the CouchDB server. Then, if you verify the list of databases in CouchDB using the URL http://127.0.0.1:5984/_utils/index.html you will get the following screenshot.



By clicking on the database named **my_database** you can see the following screenshot. Here, you can observe that the database contains a document with id **001**.



Following is an example of deleting the contents of the document having id "001" that exists in a database named **my_database** which is stored in the CouchDB server.

```
//Requiring the package
var PouchDB = require('PouchDB');

//Creating the database object
var db = new PouchDB('http://localhost:5984/my_database');

//Deleting an existing document
db.remove('001', '3-552920d1ca372986fad7b996ce365f5d', function(err) {
  if (err) {
    return console.log(err);
  } else {
    console.log("Document deleted successfully");
  }
});
```

Save the above code in a file with name **Remote_Delete_Document.js**. Open the command prompt and execute the JavaScript file using **node** as shown below.

```
C:\PouchDB_Examples >node Remote_Delete_Document.js
```

This deletes the given document that exists in the database named **my_database** which is stored in CouchDB. The following message is displayed.

```
Document deleted successfully
```

10. PouchDB – Create Batch

You can create an array (batch) of documents in PouchDB using the **db.bulkDocs()** method. While creating documents, using this method if we do not provide `_id` values, on our behalf PouchDB generates unique ids for all the documents in the bulk.

Syntax

Following is the syntax of using the **db.bulkDocs()** method of PouchDB. You can store all the documents that are to be created in PouchDB in an array and pass it to this method as a parameter. In addition to it, this method also accepts a callback (optional) function as a parameter.

```
db.bulkDocs(docs, [options], [callback])
```

Example

Following is an example of creating multiple documents in PouchDB using the **db.bulkDocs ()** method. The documents we create should be of JSON format, a set of key-value pairs separated by comma (,) and enclosed within curly braces ({}).

```
//Requiring the package
var PouchDB = require('PouchDB');

//Creating the database object
var db = new PouchDB('my_database');

//Preparing the documents array
doc1 = {_id: '001', name: 'Ram', age: 23, Designation: 'Programmer'}
doc2 = {_id: '002', name: 'Robert', age: 24, Designation: 'Programmer'}
doc3 = {_id: '003', name: 'Rahim', age: 25, Designation: 'Programmer'}
docs = [doc1, doc2, doc3]

//Inserting Documents
db.bulkDocs(docs, function(err, response) {
  if (err){
    return console.log(err);
  } else {
    console.log("Documents created Successfully");
  }
});
```

Save the above code in a file with name **Create_Batch.js**. Open the command prompt and execute the JavaScript file using **node** as shown below.

```
C:\PouchDB_Examples>node Create_Batch.js
```

This creates the given document in PouchDB database named **my_database** which is stored locally. The following message gets displayed.

```
Documents created Successfully
```

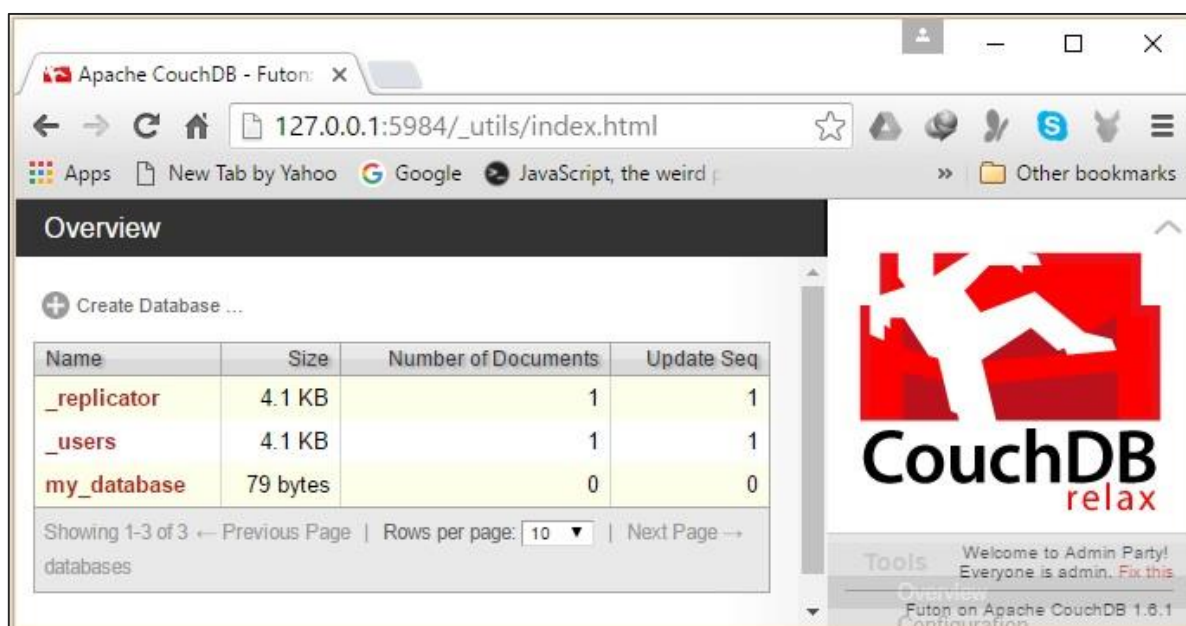
Inserting a Batch in a Remote Database

You can insert an array of documents in the database that is stored remotely on the server (CouchDB).

To do so, instead of a database name you need to pass the path to the database where we want to create documents in CouchDB.

Example

Suppose there is a database named **my_database** in the CouchDB server. Then, if you verify the list of databases in CouchDB using the URL http://127.0.0.1:5984/_utils/index.html you will get the following screenshot.



Following is an example of inserting an array of documents in the database named **my_database** which is saved in the CouchDB server.

```
//Requiring the package
var PouchDB = require('PouchDB');

//Creating the database object
var db = new PouchDB('http://localhost:5984/my_database');

//Preparing the documents array

doc1 = {_id: '001', name: 'Ram', age: 23, Designation: 'Programmer'}
doc2 = {_id: '002', name: 'Robert', age: 24, Designation: 'Programmer'}
doc3 = {_id: '003', name: 'Rahim', age: 25, Designation: 'Programmer'}

docs = [doc1, doc2, doc3]

//Inserting Documents
db.bulkDocs(docs, function(err, response) {
  if (err){
    return console.log(err);
  } else{
    console.log("Documents created Successfully");
  }
});
```

Save the above code in a file with the name **Remote_Create_Batch.js**. Open the command prompt and execute the JavaScript file using **node** as shown below.

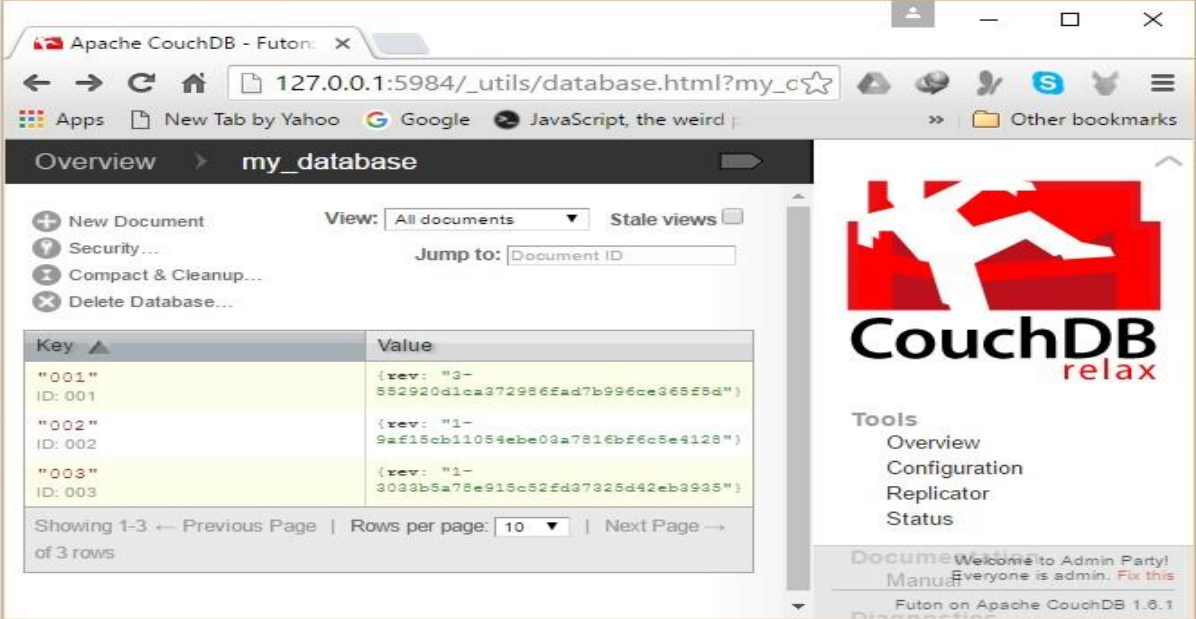
```
C:\PouchDB_Examples>node Remote_Create_Batch.js
```

This creates the given documents in PouchDB database named **my_database** which is stored in CouchDB. The following message is displayed.

```
Document created Successfully
```

Verification

After executing the above program if you visit the **my_database** again, you can observe the documents created as shown in the following screenshot.



The screenshot shows the Apache CouchDB Futon interface for a database named 'my_database'. The interface is divided into a main content area and a sidebar. The main content area displays a table of documents with the following data:

Key	Value
"001" ID: 001	{ "rev": "3-552920d1ca972986fad7b996ce365f5d" }
"002" ID: 002	{ "rev": "1-9af15cb11054ebe03a7816bf6c5e4128" }
"003" ID: 003	{ "rev": "1-3033b5a78e915c52fd37325d42eb3935" }

Below the table, it indicates 'Showing 1-3 of 3 rows' and 'Rows per page: 10'. The sidebar on the right contains the CouchDB logo, the text 'CouchDB relax', and a 'Tools' section with links to Overview, Configuration, Replicator, and Status. At the bottom of the sidebar, there is a 'Documents' section with a 'Manual' link and a 'Welcome to Admin Party!' message.

11. PouchDB – Fetch Batch

You can read/retrieve multiple/bulk documents from a database in PouchDB using the **allDocs()** method.

Syntax

Following is the syntax of using the **db.allDocs()** method of PouchDB. This method accepts an optional callback function.

```
db. allDocs()
```

Example

Following is an example of retrieving all the documents in a database named **my_database** that is stored locally, using **db.allDocs()** method. This method retrieves the array of documents in the form of objects, to get the contents of each document you need to call as **docs.rows**.

```
//Requiring the package
var PouchDB = require('PouchDB');

//Creating the database object
var db = new PouchDB('my_database');

//Retrieving all the documents in PouchDB
db.allDocs(function(err, docs) {
  if (err) {
    return console.log(err);
  } else {
    console.log (docs. rows);
  }
});
```

Save the above code in a file with the name **Read_All_Document.js**. Open the command prompt and execute the JavaScript file using **node** as shown below.

```
C:\PouchDB_Examples >node Read_All_Document.js
```

This reads all the documents that exists in the database named **my_database** which is stored locally. The following message is displayed on the console.


```
[ { id: '001',
  key: '001',
  value: { rev: '1-9dc57f5faa7ea90eeec22eba8bfd05f5' } },
  { id: '002',
    key: '002',
    value: { rev: '1-9bf80afcedb9f8b5b35567292affb254' } },
  { id: '003',
    key: '003',
    value: { rev: '1-1204f108e41bf8baf867856d5da16c57' } } ]
```

In general, as shown in the above result, using **allDocs()** method you can see only the **_id**, **key** and **_rev** fields of each document. However, to include the whole document in the result, you have to make the optional parameter **include_docs** true as shown below.

```
//Requiring the package
var PouchDB = require('PouchDB');

//Creating the database object
var db = new PouchDB('my_database');

//Retrieving all the documents in PouchDB
db.allDocs({include_docs: true}, function(err, docs) {
  if (err) {
    return console.log(err);
  } else {
    console.log (docs. rows);
  }
});
```

Executing the above code gives you a list of complete documents in the specified documents as shown in the following code.

```
[ { id: '001',
```

```

key: '001',
value: { rev: '1-9dc57f5faa7ea90eeec22eba8bfd05f5' },
doc:
{ name: 'Ram',
age: 23,
  Designation: 'Programmer',
  _id: '001',
  _rev: '1-9dc57f5faa7ea90eeec22eba8bfd05f5' } },
{ id: '002',
key: '002',
value: { rev: '1-9bf80afcedb9f8b5b35567292affb254' },
doc:
{ name: 'Robert',
age: 24,
  Designation: 'Programmer',
  _id: '002',
  _rev: '1-9bf80afcedb9f8b5b35567292affb254' } },
{ id: '003',
key: '003',
value: { rev: '1-1204f108e41bf8baf867856d5da16c57' },
doc:
{ name: 'Rahim',
age: 25,
  Designation: 'Programmer',
  _id: '003',
  _rev: '1-1204f108e41bf8baf867856d5da16c57' } }]}

```

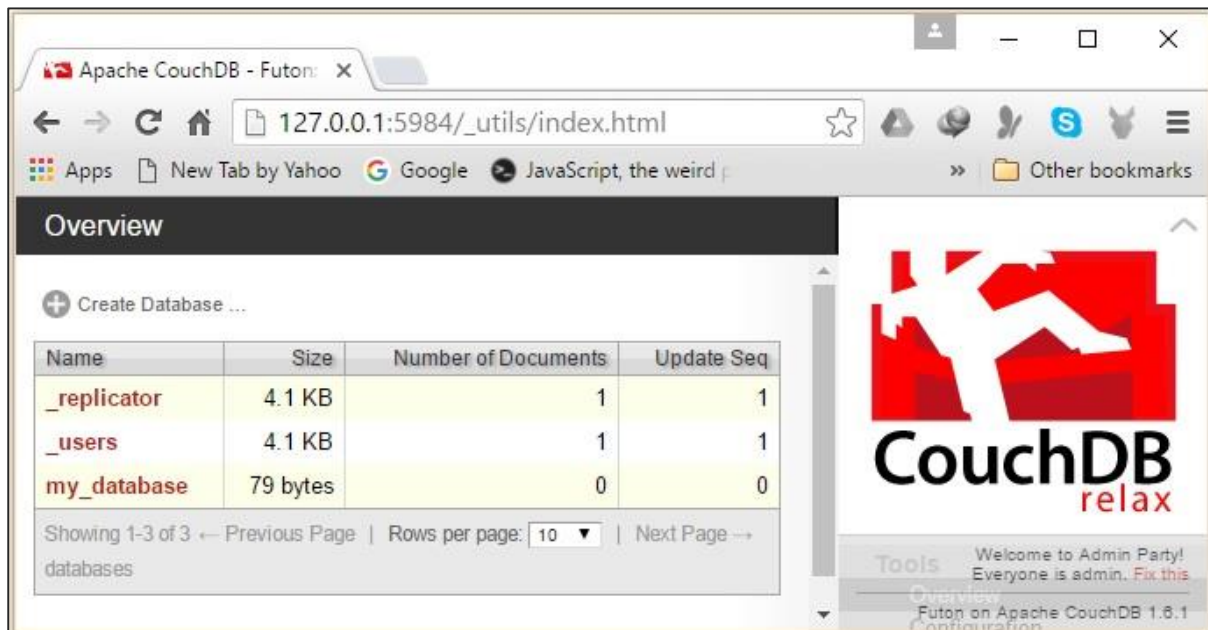
Reading a Batch from a Remote Database

You can also fetch all the documents from the database that is stored remotely on the server (CouchDB).

To do so instead of a database name, you need to pass the path to the database in CouchDB, which contains the document that is to be read.

Example

Suppose there is a database named **my_database** in the CouchDB server. Then, if you verify the list of databases in CouchDB using the URL http://127.0.0.1:5984/_utils/index.html you will get the following screenshot.



Following is an example of reading all the documents that exist in a database named **my_database** which is stored in the CouchDB server.

```
//Requiring the package
var PouchDB = require('PouchDB');

//Creating the database object
var db = new PouchDB('http://localhost:5984/my_database');

//Retrieving all the documents in PouchDB
db.allDocs({include_docs: true}, function(err, docs) {
  if (err){
    return console.log(err);
  }
  else {
    console.log(docs.rows);
  }
});
```

Save the above code in a file with the name **Remote_Read_AllDocument.js**. Open the command prompt and execute the JavaScript file using **node** as shown below.

```
C:\PouchDB_Examples >node Remote_Read _AllDocument.js
```

This reads the contents of the given document that exists in the database named **my_database** which is stored in CouchDB, and displays on the console as shown below.

```
[ { id: '001',
  key: '001',
  value: { rev: '3-552920d1ca372986fad7b996ce365f5d' },
  doc:
    { _id: '001',
      _rev: '3-552920d1ca372986fad7b996ce365f5d',
      name: 'Raju',
      age: 23,
      designation: 'Designer' } },
  { id: '002',
    key: '002',
    value: { rev: '1-9af15cb11054ebe03a7816bf6c5e4128' },
    doc:
      { _id: '002',
        _rev: '1-9af15cb11054ebe03a7816bf6c5e4128',
        name: 'Robert',
        age: 24,
        Designation: 'Programmer' } },
  { id: '003',
    key: '003',
    value: { rev: '1-3033b5a78e915c52fd37325d42eb3935' },
    doc:
      { _id: '003',
        _rev: '1-3033b5a78e915c52fd37325d42eb3935',
        name: 'Rahim',
        age: 25,
        Designation: 'Programmer' } } ]
```

12. PouchDB – Update Batch

You can update an array of documents in PouchDB at once using the **bulkDocs()** method. To do so you need to create an array of documents where, each document contains **_id**, **_rev** and the values are to be updated.

Suppose the database named **my_database** that is stored locally in PouchDB contains 3 documents namely doc1, doc2, doc3 with the following contents.

```
doc1 = {_id: '001', name: 'Ram', age: 23, Designation: 'Programmer'}
doc2 = {_id: '002', name: 'Robert', age: 24, Designation: 'Programmer'}
doc3 = {_id: '003', name: 'Rahim', age: 25, Designation: 'Programmer'}
```

Suppose we have to increase the age values in all the 3 documents by 2 years. For this to happen, first you need to get the **_rev** values. Therefore, fetch the contents of these documents using the following code.

```
//Requiring the package
var PouchDB = require('PouchDB');

//Creating the database object
var db = new PouchDB('my_database');

//Retrieving all the documents in PouchDB
db.allDocs({include_docs: true},function(err, docs) {
  if (err){
    return console.log(err);
  } else{
    console.log(docs.rows);
  }
});
```

Save the above code as **bulk_fetch.js**. On executing, the above program gives you the **_id** and **_rev** values of the documents in the database as shown below.

```
[ { id: '001',
  key: '001',
  value: { rev: '1-1604b0c3ff69dc1e261265fd60808404' } },
  { id: '002',
    key: '002',
```

```
value: { rev: '1-b5e49db7e984841bf12a13e3ee548125' } },
{ id: '003',
key: '003',
  value: { rev: '1-a7b342786ecc707aa91f3b321a177b51' } } ]
```

Now, you can update the documents using their respective **_id** and **_rev** values as shown below.

```
//Requiring the package
var PouchDB = require('PouchDB');

//Creating the database object
var db = new PouchDB('my_databas');
//Preparing the document
docs = [{_id : '001', _rev: '1-1604b0c3ff69dc1e261265fd60808404', age : 25, },
        {_id : '002', _rev: '1-b5e49db7e984841bf12a13e3ee548125', age : 26,
},
        {_id : '003', _rev: '1-a7b342786ecc707aa91f3b321a177b51', age : 27
}]

//Updating the documents in bulk
db.bulkDocs(docs, function(err, response) {
  if (err){
    return console.log(err);
  } else {
    console.log("Documents Updated Successfully");
  }
});
```

Save the above code in a file with the name **Update_All_Document.js**. Open the command prompt and execute the JavaScript file using node as shown below.

```
C:\PouchDB_Examples >node Update_All_Document.js
```

This updates all the documents that exists in the database named **my_database** which is stored locally, displaying the following message.

```
Documents Updated Successfully
```

Now, if you execute the **bulk_fetch.js** program by adding **{include_docs: true}** as a parameter to **allDocs()** function, before the callback, then, you will can see the values of the documents updated, as shown below.

```
[ { id: '001',
  key: '001',
  value: { rev: '2-77f3a9974dd578d12f3f2a33aae64c8d' },
  doc:
    { age: 25,
      _id: '001',
      _rev: '2-77f3a9974dd578d12f3f2a33aae64c8d' } },
  { id: '002',
    key: '002',
    value: { rev: '2-43966007568ce9567c96422195fcfa0d' },
    doc:
      { age: 26,
        _id: '002',
        _rev: '2-43966007568ce9567c96422195fcfa0d' } },
  { id: '003',
    key: '003',
    value: { rev: '2-6c5349652527f4f39583ff14f23cd677' },
    doc:
      { age: 27,
        _id: '003',
        _rev: '2-6c5349652527f4f39583ff14f23cd677' } } ]
```

Updating Batch from a Remote Database

You can update all the documents from the database that is stored remotely on the server (CouchDB).

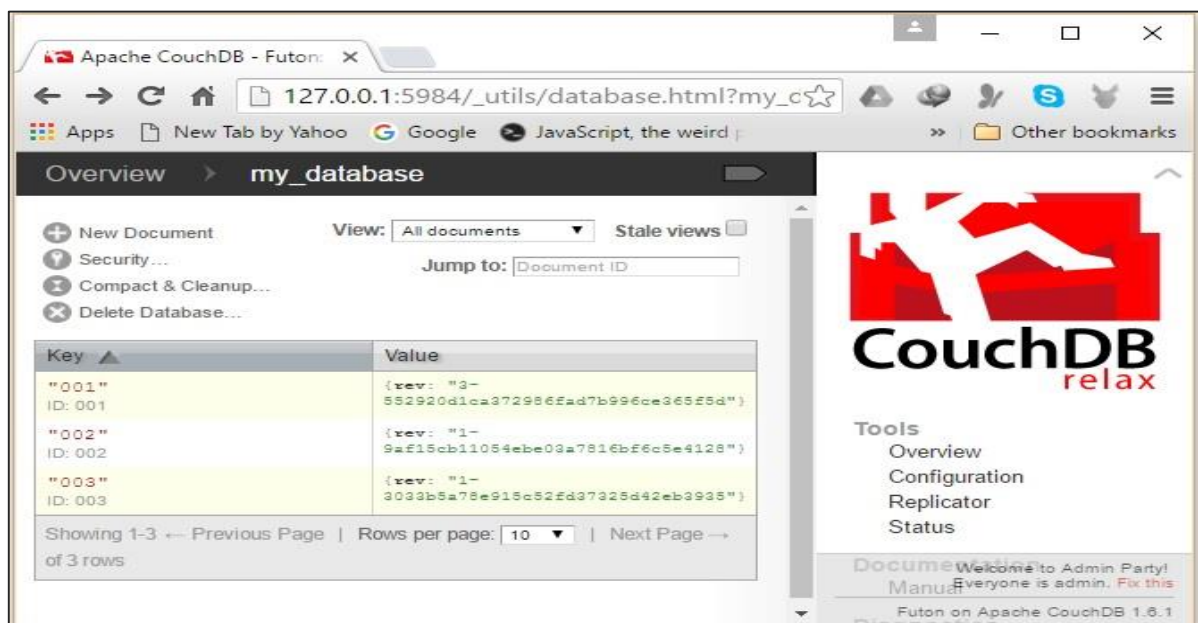
To do so, instead of a database name, you need to pass the path to the database in CouchDB, which contains the document that is to be read.

Example

Suppose there is a database named **my_database** in the CouchDB server. Then, if you verify the list of databases in CouchDB using the URL http://127.0.0.1:5984/_utils/index.html you will get the following screenshot.



And assume if we select the database named **my_database**, you can observe that it contains 3 documents as shown in the following screenshot.



Now, fetch the contents of these documents using the following code.

```
//Requiring the package
var PouchDB = require('PouchDB');

//Creating the database object
var db = new PouchDB('http://localhost:5984/my_database');
```



```
//Retrieving all the documents in PouchDB
db.allDocs({include_docs: true}, function(err, docs) {
  if (err){
    return console.log(err);
  } else{
    console.log(docs.rows);
  }
});
```

Save the above code as **remote_bulk_fetch.js**. On executing, the above program gives you the contents of all the documents in the database as shown below.

```
[ { id: '001',
  key: '001',
  value: { rev: '3-552920d1ca372986fad7b996ce365f5d' },
  doc:
    { _id: '001',
      _rev: '3-552920d1ca372986fad7b996ce365f5d',
      name: 'Raju',
      age: 23,
      designation: 'Designer' } },
  { id: '002',
    key: '002',
    value: { rev: '1-9af15cb11054ebe03a7816bf6c5e4128' },
    doc:
      { _id: '002',
        _rev: '1-9af15cb11054ebe03a7816bf6c5e4128',
        name: 'Robert',
        age: 24,
        Designation: 'Programmer' } },
  { id: '003',
    key: '003',
    value: { rev: '1-3033b5a78e915c52fd37325d42eb3935' },
    doc:
      { _id: '003',
        _rev: '1-3033b5a78e915c52fd37325d42eb3935',
```

```
name: 'Rahim',
age: 25,
  Designation: 'Programmer' } } ]
```

Following is an example of updating all the documents that exists in a database named **my_database** which is stored in the CouchDB server.

```
//Requiring the package
var PouchDB = require('PouchDB');

//Creating the database object
var db = new PouchDB('http://localhost:5984/my_database');
//Preparing the document
docs = [{_id : '001', _rev: '3-552920d1ca372986fad7b996ce365f5d', age : 24, },
        {_id : '002', _rev: '1-9af15cb11054ebe03a7816bf6c5e4128', age : 26, },
        {_id : '003', _rev: '1-3033b5a78e915c52fd37325d42eb3935', age :
27}]

//Inserting Document
db.bulkDocs(docs, function(err, response) {
  if (err){
    return console.log(err);
  } else {
    console.log(+ "Documents Updated Successfully");
  }
});
```

Save the above code in a file with the name **Remote_Update_Document.js**. Open the command prompt and execute the JavaScript file using node as shown below.

```
C:\PouchDB_Examples >node Remote_Update_Document.js
```

This updates the contents of all given document that exists in the database named **my_database** which is stored in CouchDB, and displays the following message.

```
Documents Updated Successfully
```

Now, if you execute the **remote_bulk_fetch.js** program you will can see the values of the documents updated, as shown below.

```
[ { id: '001',
```

```
key: '001',
value: { rev: '4-6bc8d9c7a60fed2ed1667ec0740c1f39' },
doc:
{ _id: '001',
  _rev: '4-6bc8d9c7a60fed2ed1667ec0740c1f39',
  age: 25 } },
{ id: '002',
key: '002',
value: { rev: '2-1aa24ce77d96bb9d2a0675cdf1e113e0' },
doc:
{ _id: '002',
  _rev: '2-1aa24ce77d96bb9d2a0675cdf1e113e0',
  age: 26 } },
{ id: '003',
key: '003',
value: { rev: '2-fa113149ba618eda77f73072974a2bc1' },
doc:
{ _id: '003',
  _rev: '2-fa113149ba618eda77f73072974a2bc1',
  age: 27 } } ]
```

13. PouchDB – Delete Batch

You can delete an array of documents in PouchDB at once using the **bulkDocs()** method. To do so you need to create an array of documents that are to be deleted where, each document should contain **_id** and **_rev**. In addition to these you have to add another key-value pair **_deleted: true**.

Suppose the database named **my_database** that is stored locally in PouchDB contains 3 documents namely doc1, doc2, doc3 with the following contents.

```
doc1 = {_id: '001', name: 'Ram', age: 23, Designation: 'Programmer'}
doc2 = {_id: '002', name: 'Robert', age: 24, Designation: 'Programmer'}
doc3 = {_id: '003', name: 'Rahim', age: 25, Designation: 'Programmer'}
```

And say, we have to delete all the three documents. Then, first of all you need to get their **_rev** values. Therefore, fetch the contents of these documents using the following code.

```
//Requiring the package
var PouchDB = require('PouchDB');

//Creating the database object
var db = new PouchDB('my_database');

//Retrieving all the documents in PouchDB
db.allDocs({include_docs: true},function(err, docs) {
  if (err){
    return console.log(err);
  } else {
    console.log(docs.rows);
  }
});
```

Save the above code as **bulk_fetch.js**. Executing the above program gives you the `_id` and `_rev` values of the documents in the database as shown below.

```
[ { id: '001',
  key: '001',
  value: { rev: '1-1604b0c3ff69dc1e261265fd60808404' } },
  { id: '002',
  key: '002',
  value: { rev: '1-b5e49db7e984841bf12a13e3ee548125' } },
  { id: '003',
  key: '003',
    value: { rev: '1-a7b342786ecc707aa91f3b321a177b51' } } ]
```

Now, you can delete the documents using their respective `_id` and `_rev` values as shown below.

```
//Requiring the package
var PouchDB = require('PouchDB');

//Creating the database object
var db = new PouchDB('my_database');

//Preparing the document
docs = [{_id : '001', _rev: '2-77f3a9974dd578d12f3f2a33aae64c8d', _deleted :
true },
        {_id : '002', _rev: '2-43966007568ce9567c96422195fcfa0d', _deleted :
true },
        {_id : '003', _rev: '2-6c5349652527f4f39583ff14f23cd677', _deleted
: true }]

//Deleting Documents
db.bulkDocs(docs, function(err, response) {
  if (err){
    return console.log(err);
  } else{
    console.log(response+"Documents deleted Successfully");
  }
});
```

Save the above code in a file with the name **Delete_All_Document.js**. Open the command prompt and execute the JavaScript file using node as shown below.

```
C:\PouchDB_Examples >node Delete_All_Document.js
```

This deletes all the documents that exists in the database named **my_database** which is stored locally, displaying the following message.

```
Documents Deleted Successfully
```

Now, if you execute the **bulk_fetch.js** program, you can observe an empty brace on the console indicating that the database is empty, as shown below.

```
[]
```

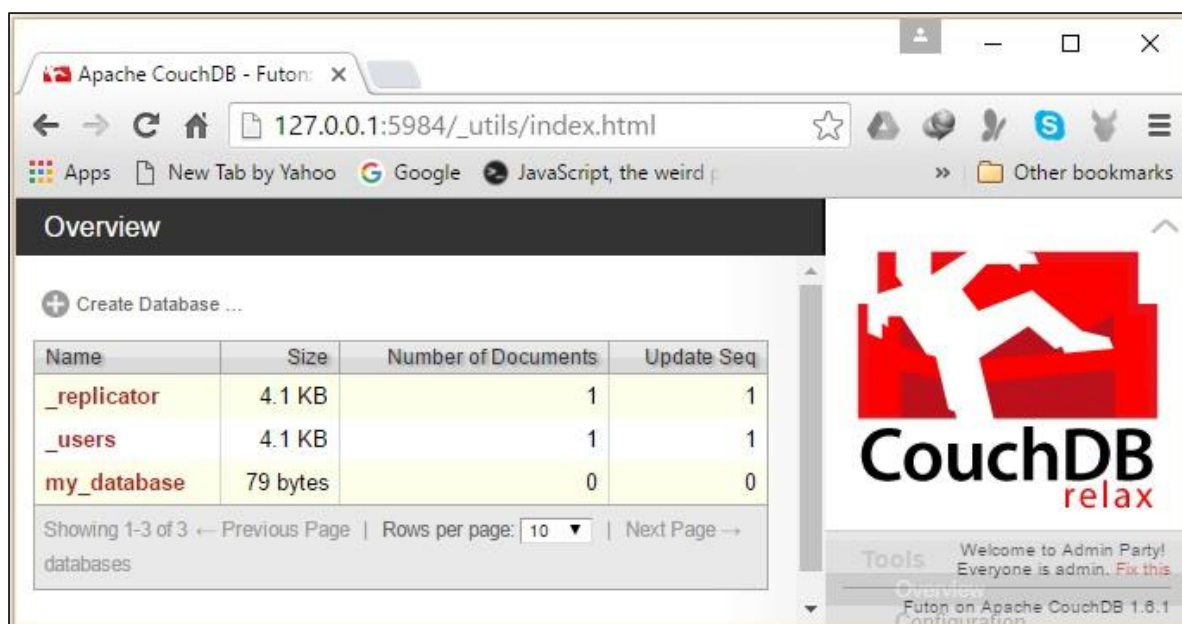
Deleting Batch from a Remote Database

You can update all the documents from the database that is stored remotely on the server (CouchDB).

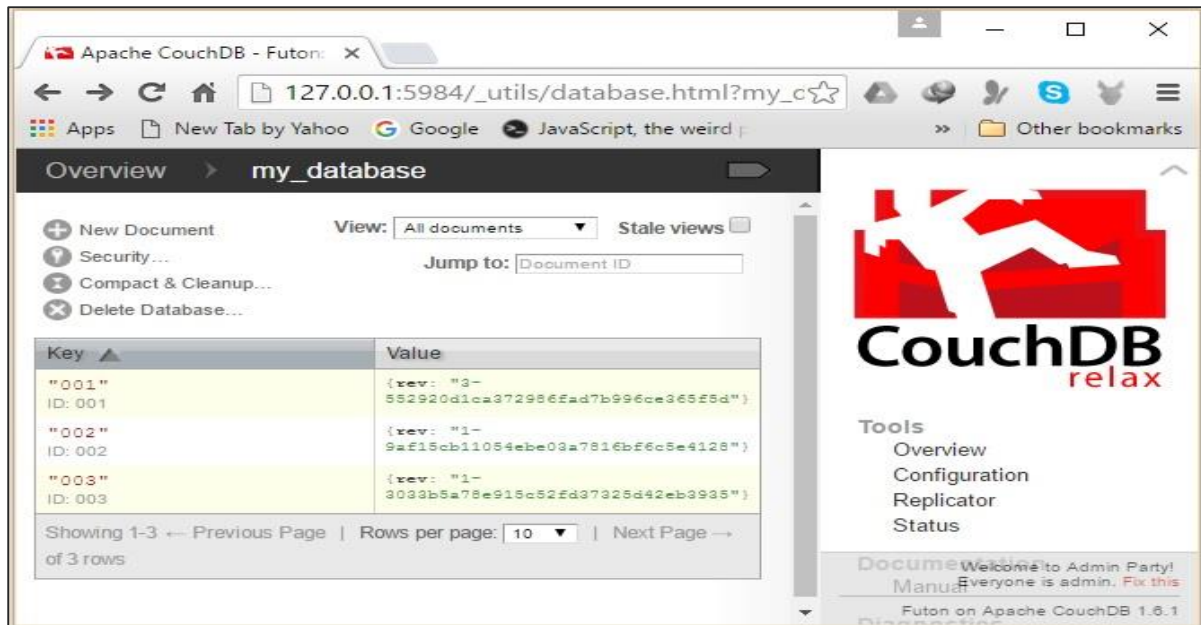
To do so, instead of a database name, you need to pass the path to the database in CouchDB, which contains the document that is to be read.

Example

Suppose there is a database named **my_database** in the CouchDB server. Then, if you verify the list of databases in CouchDB using the URL http://127.0.0.1:5984/_utils/index.html you will get the following screenshot.



If we select the database named **my_database**, you can observe that it contains 3 documents as shown in the following screenshot.



Following is an example of deleting all the documents that exist in a database named **my_database** which is stored in the CouchDB server.

```
//Requiring the package
var PouchDB = require('PouchDB');
//Creating the database object
var db = new PouchDB('http://localhost:5984/my_database');
//Preparing the document
docs = [{_id : '001', _rev: '4-6bc8d9c7a60fed2ed1667ec0740c1f39', _deleted :
true },
      {_id : '002', _rev: '2-1aa24ce77d96bb9d2a0675cdf1e113e0', _deleted :
true },
      {_id : '003', _rev: '2-fa113149ba618eda77f73072974a2bc1', _deleted :
true }]
//Deleting Documents
db.bulkDocs(docs, function(err, response) {
  if (err) {
    return console.log(err);
  } else {
    console.log("Documents deleted Successfully");
  }
});
```

Save the above code in a file with name **Remote_delete_AllDocuments.js**. Open the command prompt and execute the JavaScript file using node as shown in the following screenshot.

```
C:\PouchDB_Examples >node Remote_Delete_AllDocuments.js
```

This deletes the contents of all given document that exists in the database named **my_database** which is stored in CouchDB, and displays the following message.

```
Documents Deleted Successfully
```


14. PouchDB – Adding Attachment

You can attach a binary object to a document using the **putAttachment()** method in PouchDB.

Syntax

Following is the syntax of the **putAttachment()**. To this method, we have to pass the document id, attachment id, MIME type along with the attachment. This method also accepts an optional callback function.

```
db.putAttachment( docId, attachmentId, attachment, type, [callback] );
```

We can prepare attachment using blob or buffer objects, where **blob** is used while working with the browser and **buffer** is used while working with **Node.js**, since we are demonstrating our programs in Node.js, we use buffer objects to prepare documents.

Example

Following is an example of creating a document with an attachment, within a database named **my_database** in PouchDB using **putAttachment()** method.

```
//Requiring the package
var PouchDB = require('PouchDB');

//Creating the database object
var db = new PouchDB('my_database');

//Preparing the attachment
var my_attachment = new Buffer(['Welcome to tutorialspoint'], {type:
'text/plain'});

//Adding attachment to a document
db.putAttachment('001', 'att_1.txt', my_attachment, 'text/plain', function(err,
res) {
    if (err) {
        return console.log(err);
    } else {
        console.log(res+"Attachment added successfully")
    }
});
```

Save the above code in a file with name **Add_Attachment.js**. Open the command prompt and execute the JavaScript file using **node** as shown below.

```
C:\PouchDB_Examples >node Add_Attachment.js
```

This creates an empty document adding an attachment to it, in the database named **my_database** which is stored in PouchDB, and displays the following message.

```
Attachment added successfully
```

You can verify whether the attachment is added by reading the document using the following code.

```
//Requiring the package
var PouchDB = require('PouchDB');

//Creating the database object
var db = new PouchDB('my_database');

//Reading the Document
db.get('001',{attachments: true}, function(err, doc) {
  if (err){
    return console.log(err);
  } else{
    console.log(doc);
  }
});
```

Save the above code as **read_doc.js** and execute it. Executing this program, you can see the following contents of the document.

```
{ _attachments:
  { att_1.txt:
    { content_type: 'text/plain',
      digest: 'md5-k7iFr4NoInN9jSQT9WfcQ==',
      data: 'AA==' } },
  _id: '001',
  _rev: '1-620fd5f41d3328fcbf9ce7504338a51d' }
```

Adding Attachment to an Existing Document

Suppose, there is a document in a database by the name **my_database** PouchDB with id **'002'**. You can get the contents of it by executing the **read_doc.js** by changing the id value to **002**, as shown below.

```
{ name: 'Raju',
  age: 23,
  designation: 'Designer',
  _id: '002',
  _rev: '1-05ca7b5f3f4762a9fb2d119cd34c8d40' }
```

Now, you can add an attachment to this document using its **_rev** value.

```
//Requiring the package
var PouchDB = require('PouchDB');

//Creating the database object
var db = new PouchDB('my_database');

//Adding attachment to existing document
var my_attachment = new Buffer (['Welcome to tutorialspoint'], {type:
'text/plain'});

rev = '1-05ca7b5f3f4762a9fb2d119cd34c8d40';
db.putAttachment('002','att_1.txt',rev, my_attachment, 'text/plain',
function(err, res) {
  if (err) {
    return console.log(err);
  } else {
    console.log (res+"Attachment added successfully")
  }
});
```

Save the above code in a file with the name **Add_Attachment_to_doc.js**. Open the command prompt and execute the JavaScript file using **node** as shown below.

```
C:\PouchDB_Examples >node Add_Attachment_to_doc.js
```

This adds an attachment to the specified document displaying the following message.

```
Attachment added successfully
```

If you change the id value in **read_doc.js** to **002** and execute it, you will get the following output.



```
{ name: 'Raju',
  age: 23,
  designation: 'Designer',
  _attachments:
  { att_1:
    { content_type: 'text/plain',
      digest: 'md5-k7iFr4NoInN9jSQT9WfcQ==',
      data: 'AA==' } },
  _id: '002',
  _rev: '2-3bb4891b954699bce28346723cc7a709' }
```

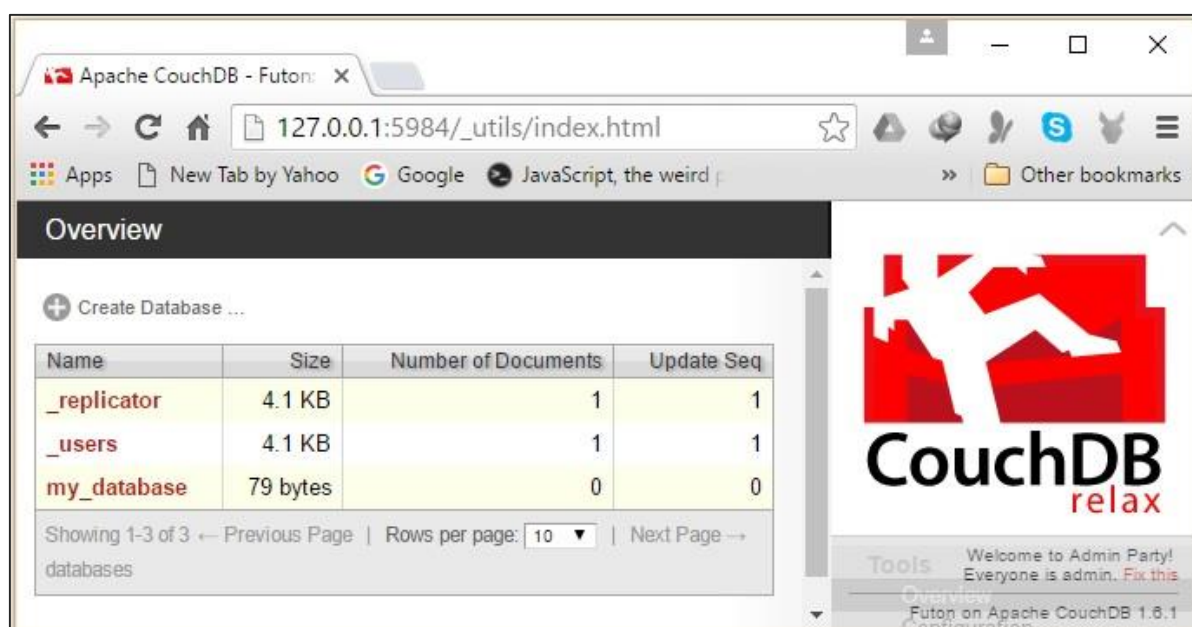
Adding Attachment to a Remote Document

You can even add an attachment to the document existing in a database that is stored remotely on the server (CouchDB).

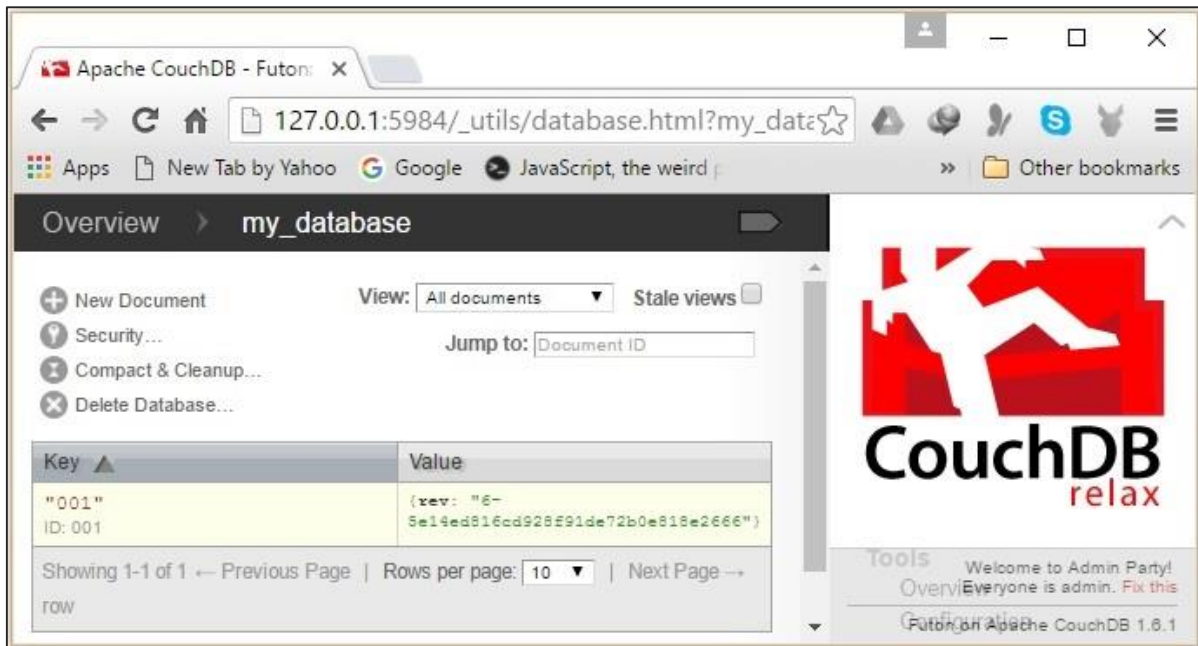
To do so, instead of a database name, you need to pass the path to the database in CouchDB, which contains the document that is to be read.

Example

Suppose there is a database named **my_database** in the CouchDB server. Then, if you verify the list of databases in CouchDB using the URL http://127.0.0.1:5984/_utils/index.html you will get the following screenshot.



And if you select the database named **my_database**, you can view its contents as shown below.



Following is an example of adding an attachment to the document **001** stored in a database named **my_database** which is stored in the CouchDB server.

```
//Requiring the package
var PouchDB = require('PouchDB');

//Creating the database object
var db = new PouchDB('http://localhost:5984/my_database');

//Adding attachment to existing document
var my_attachment = new Buffer(['Welcome to tutorialspoint'], {type:
'text/plain'});

rev = '1-36c34fdcf29a652876219065f9681602';
db.putAttachment('001', 'att_1.txt', rev, my_attachment, 'text/plain',
function(err, res) {
  if (err) {
```

```
        return console.log(err);  
    } else {  
        console.log (res+ "Attachment added successfully")  
    }  
});
```

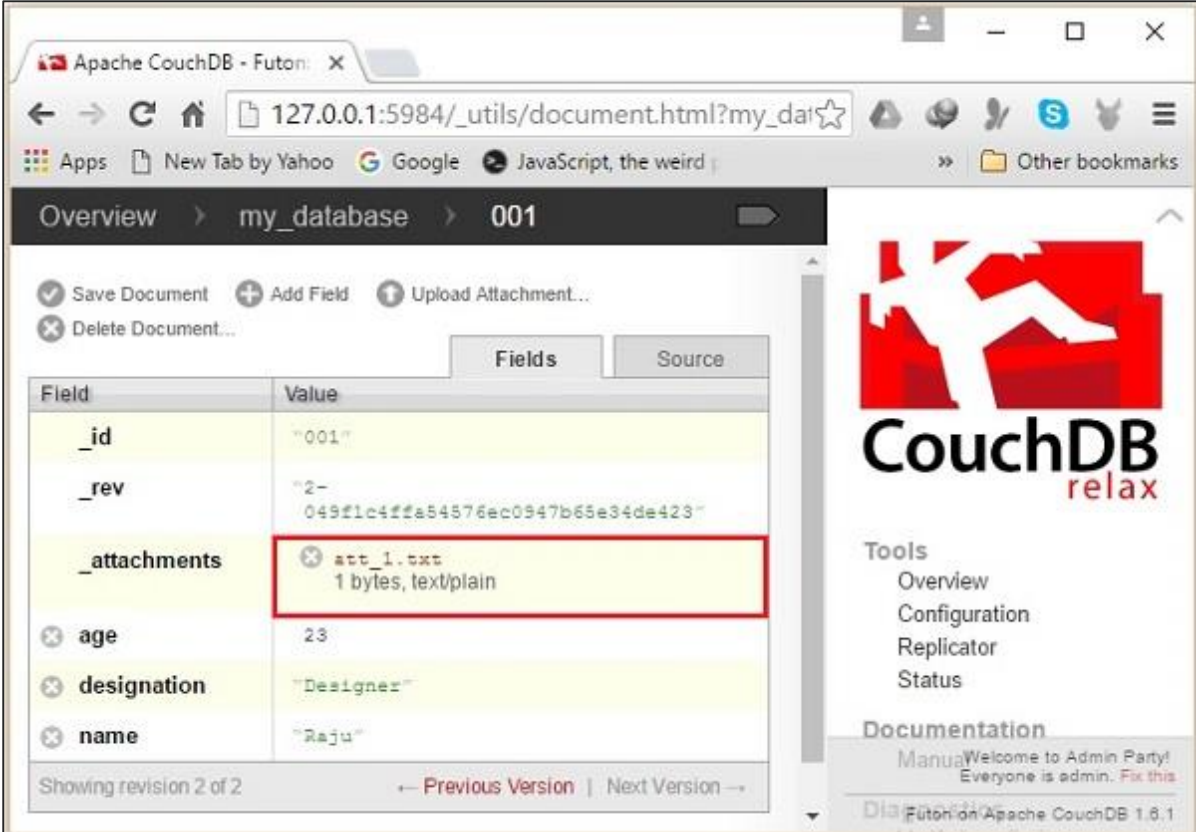
Save the above code in a file with the name **Remote_Add_Attachment.js**. Open the command prompt and execute the JavaScript file using **node** as shown below.

```
C:\PouchDB_Examples >node Remote_Add_Attachment.js
```

This adds an attachment to the specified document displaying the following message.

```
Attachment added successfully
```

Now, if you verify the document, you can observe the attachment added to it as shown in the following screenshot.



Apache CouchDB - Futon: X

127.0.0.1:5984/_utils/document.html?my_dai

Overview > my_database > 001

Save Document Add Field Upload Attachment...
Delete Document...

Field	Value
_id	"001"
_rev	"2-049f1c4ffa54576ec0947b65e34de423"
_attachments	att_1.txt 1 bytes, text/plain
age	23
designation	"Designer"
name	"Raju"

Showing revision 2 of 2 Previous Version Next Version

CouchDB relax

Tools

- Overview
- Configuration
- Replicator
- Status

Documentation

Manual Welcome to Admin Party!
Everyone is admin. [Fix this](#)

Dis...

15. PouchDB – Retrieving Attachment

You can retrieve an attachment from PouchDB using the **getAttachment()** method. This method always returns blob or buffer objects.

Syntax

Following is the syntax of the **getAttachment()**. To this method, we have to pass the document id and attachment id. This method also accepts an optional callback function.

```
db.getAttachment( docId, attachmentId, [callback] );
```

Example

Following is an example of retrieving an attachment of a document stored in PouchDB, using **getAttachment()** method. Using this code, we are trying to retrieve an attachment **att_1.txt** from the document **001**.

```
//Requiring the package
var PouchDB = require('PouchDB');

//Creating the database object
var db = new PouchDB('my_database');

//Retrieving an attachment from a document
db.getAttachment('001', 'att_1.txt', function(err, blob_buffer) {
  if (err) {
    return console.log(err);
  } else{
    console.log(blob_buffer);
  }
});
```

Save the above code in a file with the name **Retrive_Attachment.js**. Open the command prompt and execute the JavaScript file using **node** as shown below.

```
C:\PouchDB_Examples >node Retrive_Attachment.js
```

This retrieves the attachment of the document and displays on the console as shown below.

<Buffer 00>

Retrieving Attachment from a Remote Document

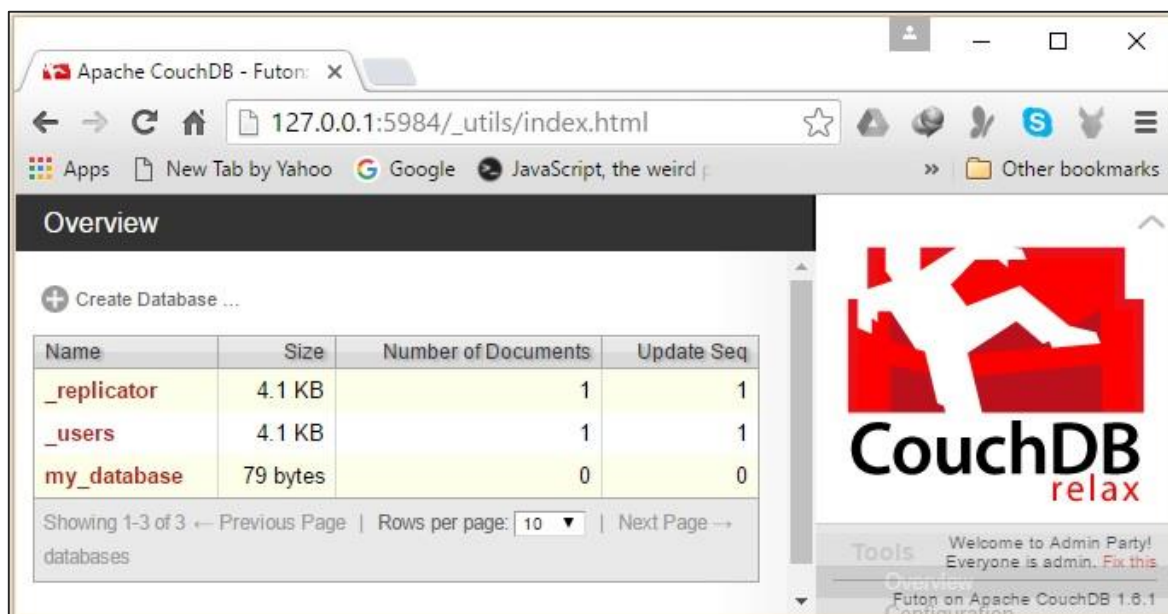
You can also retrieve an attachment of a document existing in the database that is stored remotely on the server (CouchDB).

To do so, instead of a database name, you need to pass the path to the database in CouchDB, which contains the document that is to be read.

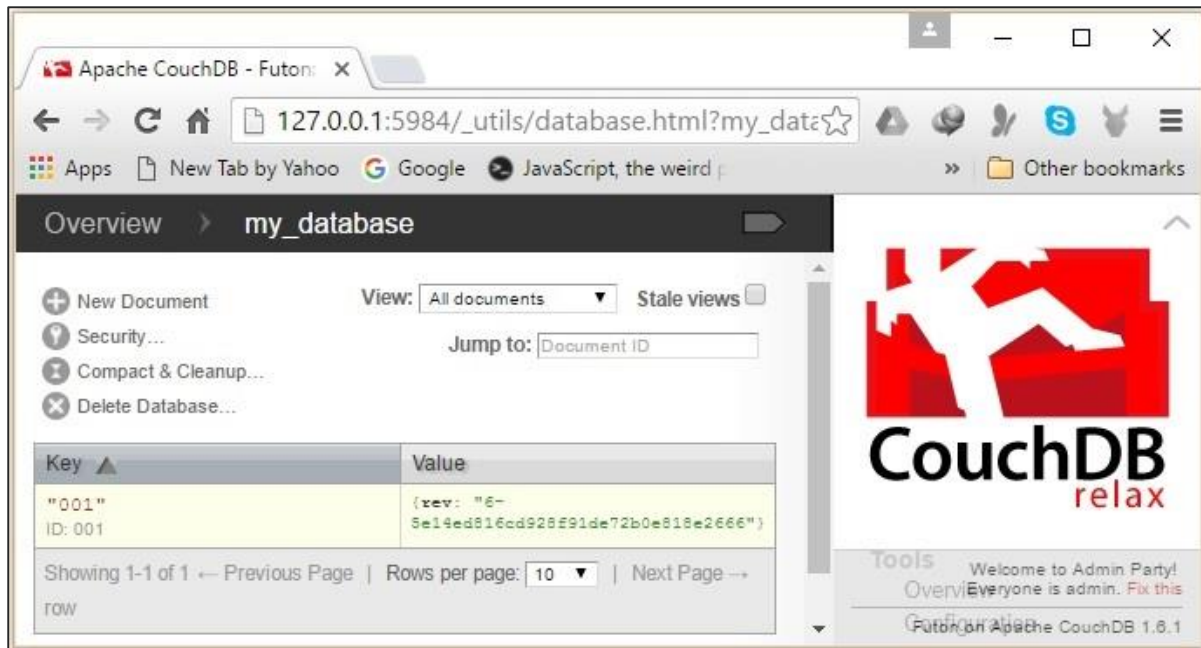
Example

Suppose there is a database named **my_database** in the CouchDB server. Then, if you verify the list of databases in CouchDB using the URL

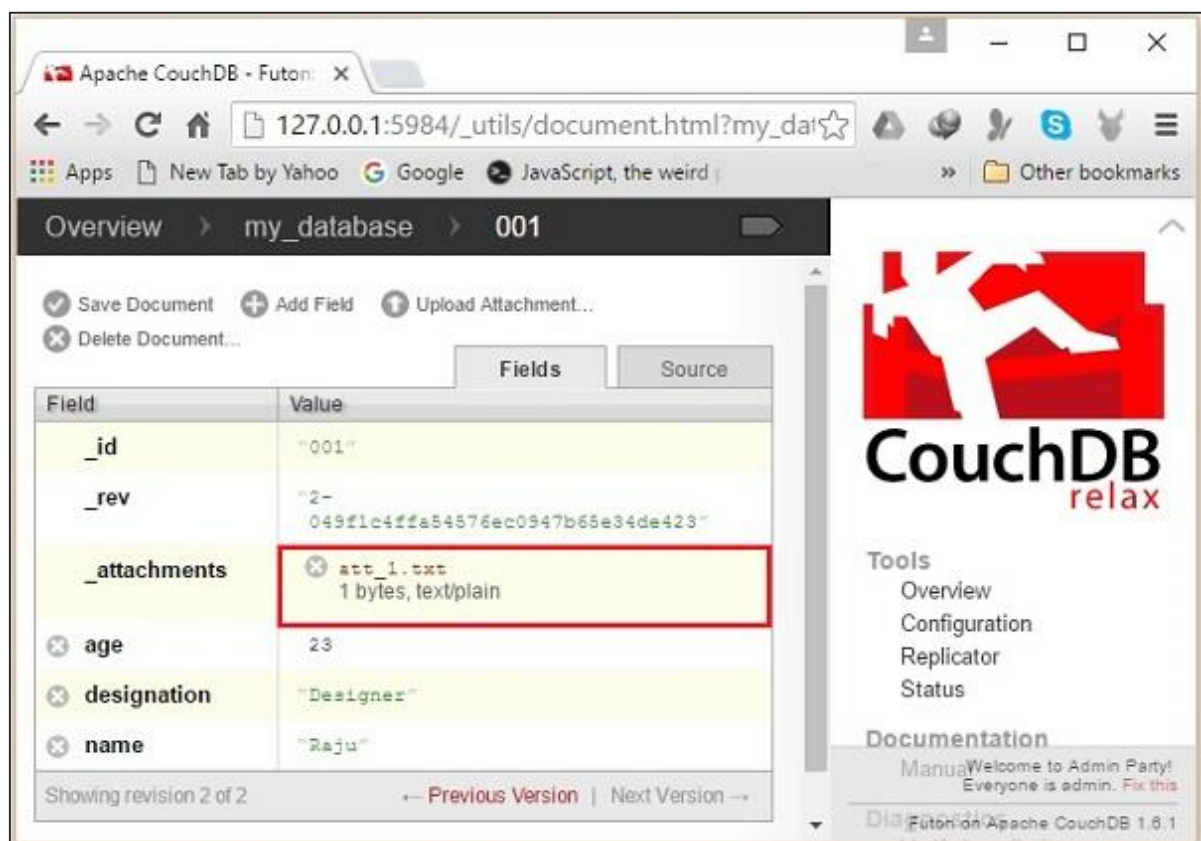
http://127.0.0.1:5984/_utils/index.html you will get the following screenshot.



If you select the database named **my_database**, you can view its contents as shown below.



Suppose, there is an attachment in this document as shown below.



Following is an example of retrieving an attachment of the document **001** that exists in a database named **my_database**, which is stored in the CouchDB server.

```
//Requiring the package
var PouchDB = require('PouchDB');

//Creating the database object
var db = new PouchDB('http://localhost:5984/my_database');

//Retrieving an attachment from a document
db.getAttachment('001', 'att_1.txt', function(err, blob_buffer) {
  if (err) {
    return console.log(err);
  } else {
    console.log(blob_buffer);
  }
});
```

Save the above code in a file with the name **Remote_Retrieve_Attachment.js**. Open the command prompt and execute the JavaScript file using **node** as shown below.

```
C:\PouchDB_Examples >node Remote_Retrieve_Attachment.js
```

This retrieves the document attachment and displays it on the console as shown below.

```
<Buffer 00>
```

16. PouchDB – Deleting Attachment

You can delete an attachment from PouchDB using the **removeAttachment()** method.

Syntax

Following is the syntax of the **removeAttachment()** method. To this method, we have to pass the document id, attachment id, and _rev value. This method also accepts an optional callback function.

```
db.removeAttachment(docId, attachmentId, rev, [callback]);
```

Example

Suppose there is a document in PouchDB with id **001**, which contains id, name, age, designation of an employee along with an attachment as shown below.

```
{ name: 'Raju',  
  age: 23,  
  designation: 'Designer',  
  _attachments:  
  { 'att_1.txt':  
    { content_type: 'text/plain',  
      digest: 'md5-k7iFr4NoInN9jSQT9WfcQ==',  
      data: 'AA==' } },  
  _id: '001',  
  _rev: '2-cdec6c9f45ddbee7d456945654742d43' }
```

Following is an example of deleting the attachment of this document **001** stored in PouchDB, using **removeAttachment()** method.

```
//Requiring the package  
var PouchDB = require('PouchDB');  
  
//Creating the database object  
var db = new PouchDB('my');  
  
db.removeAttachment('001', 'att_1.txt', '2-cdec6c9f45ddbee7d456945654742d43',  
function(err, res) {
```

```

    if (err) {
        return console.log(err);
    }
    else {
        console.log(res+"Attachment Deleted successfully")
    }
});

```

Save the above code in a file with the name **Remove_Attachment.js**. Open the command prompt and execute the JavaScript file using **node** as shown below.

```
C:\PouchDB_Examples >node Remove_Attachment.js
```

This removes the attachment of the document and displays a message on the console as shown below.

```
Attachment deleted successfully
```

After deletion, you can verify the contents of the document by executing the following code.

```

//Requiring the package
var PouchDB = require('PouchDB');

//Creating the database object
var db = new PouchDB('my_d');

//Reading the Document
db.get('001',{attachments: true}, function(err, doc) {
    if (err){
        return console.log(err);
    }
    else{
        console.log(doc);
    }
});

```

Save this code as **read.js** and execute it. On executing, you will get the contents of the document after deleting the attachment, as shown below.

```
{ name: 'Raju',
  age: 23,
  designation: 'Designer',
  _id: '001',
  _rev: '3-da775487a6ed0495f2e49c543384f8e8' }
```

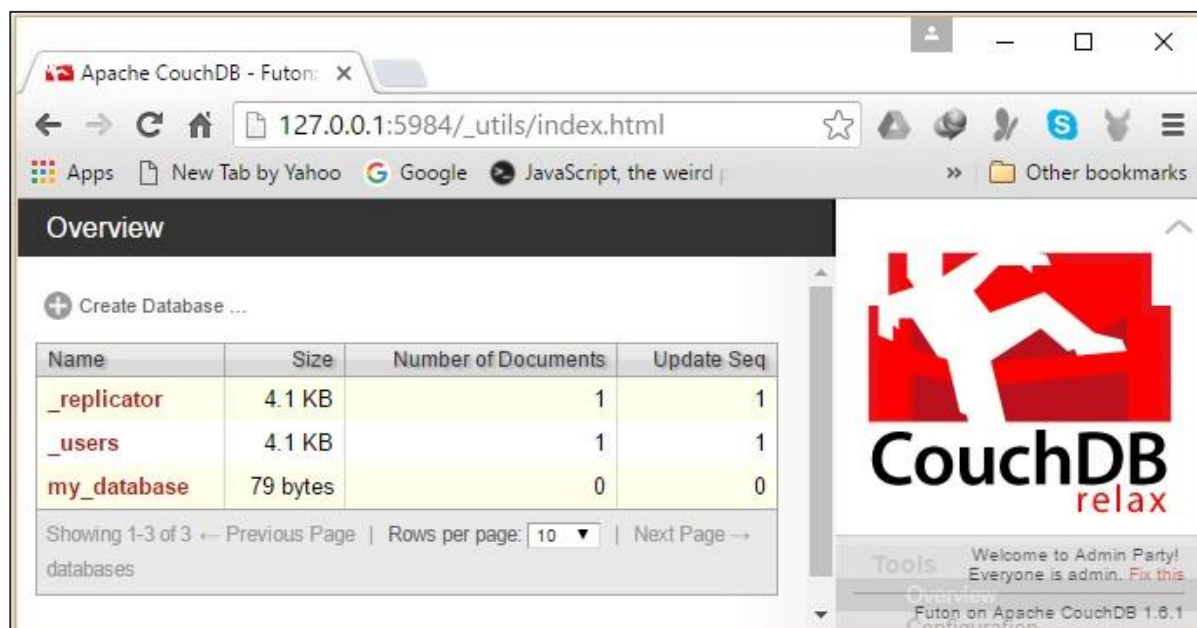
Removing Attachment from a Remote Document

You can delete an attachment of an existing document in the database that is stored remotely on the server (CouchDB).

To do so, instead of a database name, you need to pass the path to the database in CouchDB, which contains the document that is to be read.

Example

Suppose there is a database named **my_database** in the CouchDB server. Then, if you verify the list of databases in CouchDB using the URL http://127.0.0.1:5984/_utils/index.html you will get the following screenshot.



And if you select the database named **my_database**, you can view its contents as shown below.

Apache CouchDB - Futon: X

127.0.0.1:5984/_utils/database.html?my_database

Overview > my_database

+ New Document
 Security...
 Compact & Cleanup...
 X Delete Database...

View: All documents ▼ Stale views ☐

Jump to:

Key ▲	Value
"001" ID: 001	{_rev: "6- 8e14ed816cd928f91de72b0e818e2666"}

Showing 1-1 of 1 ← Previous Page | Rows per page: 10 ▼ | Next Page →

Tools

Welcome to Admin Party!
Everyone is admin. [Fix this](#)

Futon on Apache CouchDB 1.6.1

Suppose there is an attachment in this document as shown below.

Apache CouchDB - Futon: X

127.0.0.1:5984/_utils/document.html?my_database/001

Overview > my_database > 001

✓ Save Document
 X Delete Document...

+ Add Field
 Upload Attachment...

Field	Value
_id	"001"
_rev	"2- 049f1c4ffa54576ac0947b65e34de423"
_attachments	X att_1.txt 1 bytes, text/plain
age	23
designation	"Designer"
name	"Raju"

Showing revision 2 of 2 ← Previous Version | Next Version →

Tools

Overview
Configuration
Replicator
Status

Documentation

Manual Welcome to Admin Party!
Everyone is admin. [Fix this](#)

Diagnosis Futon on Apache CouchDB 1.6.1

Following is an example of deleting the above mentioned attachment of the document **001** that exists in a database named **my_database** which is stored in the CouchDB server.

```
//Requiring the package
var PouchDB = require('PouchDB');

//Creating the database object
var db = new PouchDB('http://localhost:5984/my_database');

db.removeAttachment('001', 'att_1.txt', '2-049f1c4ffa54576ec0947b65e34de423',
function(err, res) {
    if (err) {
        return console.log(err);
    } else{
        console.log(res+"Attachment Deleted successfully")
    }
});
```

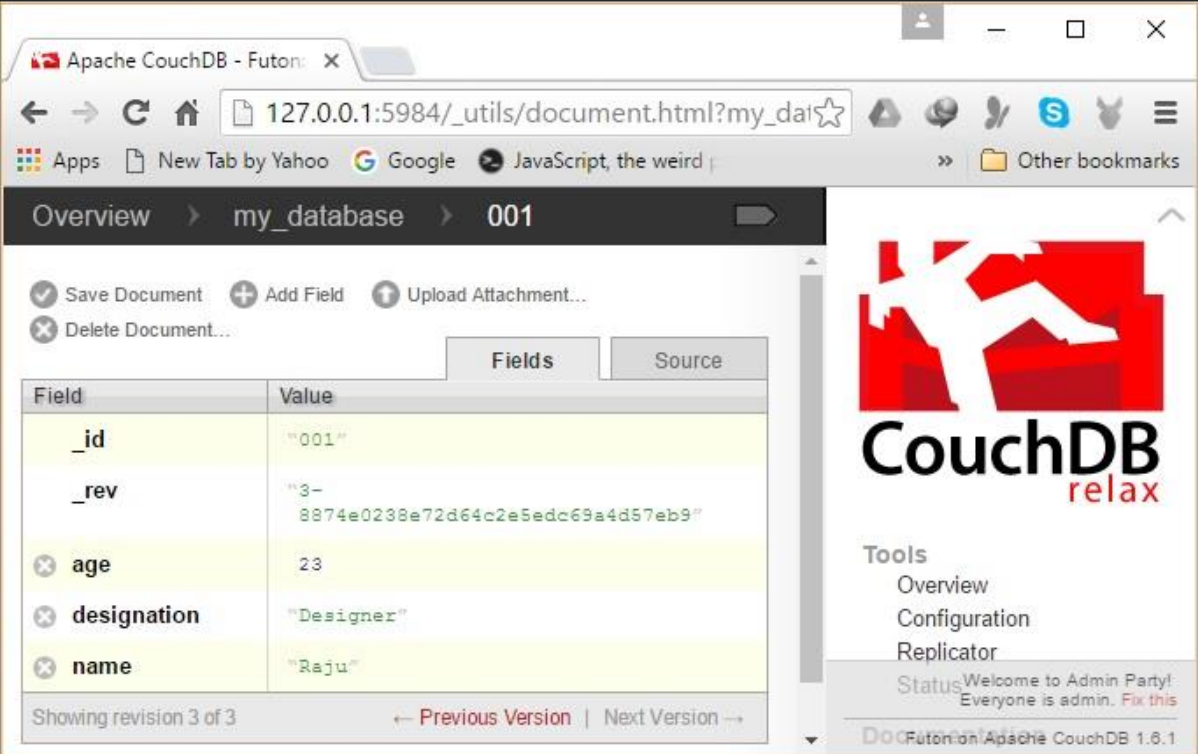
Save the above code in a file with the name **Remote_Delete_Attachment.js**. Open the command prompt and execute the JavaScript file using **node** as shown below.

```
C:\PouchDB_Examples >node Remote_Delete_Attachment.js
```

This removes the existing attachment and displays the following message.

```
Attachment Deleted successfully
```


If you visit the document again, you can notice that the attachment was deleted as shown in the following screenshot.



The screenshot shows the Apache CouchDB Futon web interface in a browser. The address bar displays the URL `127.0.0.1:5984/_utils/document.html?my_database=001`. The breadcrumb navigation shows `Overview > my_database > 001`. The document is in revision 3 of 3. The document fields are as follows:

Field	Value
<code>_id</code>	<code>"001"</code>
<code>_rev</code>	<code>"3-8874e0238e72d64c2e5edc69a4d57eb9"</code>
<code>age</code>	23
<code>designation</code>	<code>"Designer"</code>
<code>name</code>	<code>"Raju"</code>

On the right side, the CouchDB logo with the tagline "relax" is visible. Below it, the "Tools" menu includes links for Overview, Configuration, and Replicator. The "Status" section shows a message: "Welcome to Admin Party! Everyone is admin. [Fix this](#)". The footer indicates "Futon on Apache CouchDB 1.6.1".

17. PouchDB–Replication

One of the most important features of PouchDB is replication, i.e. you can make a copy of a database. You can replicate either a PouchDB instance stored locally or a CouchDB instance stored remotely.

Syntax

Following is the syntax of replicating a database in PouchDB. Here, a copy of the **source database** is the target. To this method, you can directly pass the location of source and destination databases in String format, or you can pass objects representing them.

```
PouchDB.replicate(source, target, [options])
```

Both the source and targets can be either PouchDB instances or CouchDB instances.

Replicating LocalDB to CouchDB

Suppose there is a database with the name **sample_database** in PouchDB, and it contains 3 documents doc1, doc2, and doc3, having contents as shown below.

```
doc1 = {_id: '001', name: 'Ram', age: 23, Designation: 'Programmer'}  
doc2 = {_id: '002', name: 'Robert', age: 24, Designation: 'Programmer'}  
doc3 = {_id: '003', name: 'Rahim', age: 25, Designation: 'Programmer'}
```

Following is an example which makes a copy of the database named **sample_database** that is stored locally in CouchDB.

```
//Requiring the package  
var PouchDB = require('PouchDB');  
  
var localdb = 'sample_database';  
  
//Creating remote database object  
var remotedb = 'http://localhost:5984/sample_database';  
  
//Replicating a local database to Remote  
PouchDB.replicate(localDB, remoteDB);  
console.log ("Database replicated successfully");
```

Save the above code in a file with name **Replication_example.js**. Open the command prompt and execute the JavaScript file using **node** as shown below.

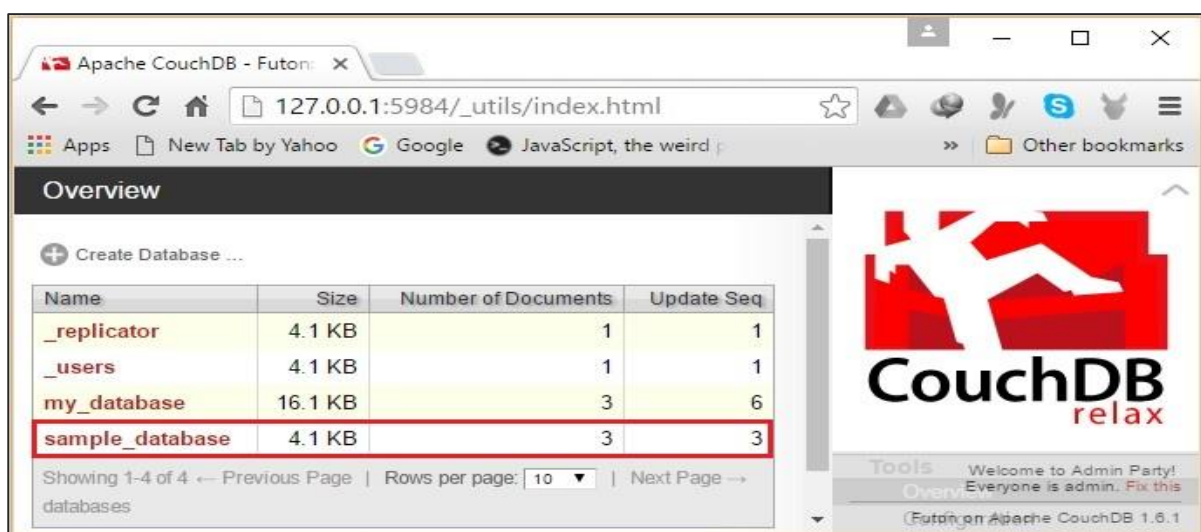
```
C:\PouchDB_Examples >node Replication_example.js
```

This makes a copy of the database named **sample_database** in CouchDB instance and displays a message on the console as shown below.

```
Database replicated successfully
```

You can verify whether the database is replicated in your CouchDB instance by clicking the following link http://127.0.0.1:5984/_utils/index.html.

On clicking, you can see the list of databases in your CouchDB. You can also observe that a copy of the database **sample_database** is created here.



If you select the replicated database, you can view its contents as shown below.



Replicating CouchDB to PouchDB

Suppose there is a database with the name **Remote_Database** in CouchDB and it contains 3 documents, doc1, doc2, and doc3, having contents as shown below.

```
doc1 = {_id: '001', name: 'Geeta', age: 25, Designation: 'Programmer'}  
doc2 = {_id: '002', name: 'Zara Ali', age: 24, Designation: 'Manager'}  
doc3 = {_id: '003', name: 'Mary', age: 23, Designation: 'Admin'}
```

Following is an example which makes a copy of the database named **Remote_Database** that is stored in CouchDB in the local storage.

```
//Requiring the package  
var PouchDB = require('PouchDB');  
  
var localdb = 'sample_database';  
  
var remotedb = 'http://localhost:5984/sample_database1';  
  
//Replicating a local database to Remote  
PouchDB.replicate( remotedb, localdb);  
console.log("Database replicated successfully");
```

Save the above code in a file with the name **Replication_example2.js**. Open the command prompt and execute the JavaScript file using **node** as shown below.

```
C:\PouchDB_Examples >node Replication_example2.js
```

This makes a copy of the database named **remote_database** in PouchDB instance and displays a message on the console as shown below.

```
Database replicated successfully
```

You can verify whether the database is replicated in your Pouch instance by executing the following code.

```
//Requiring the package
var PouchDB = require('PouchDB');

//Creating the database object
var db = new PouchDB('remote_database');

//Retrieving all the documents in PouchDB
db.allDocs({include_docs: true, attachments: true}, function(err, docs) {
  if (err) {
    return console.log(err);
  }
  else {
    console.log(docs.rows);
  }
});
```

If the database is replicated on executing the above code, you will get the contents of the replicated database as shown below.

```
[ { id: '001',
  key: '001',
  value: { rev: '1-23cf3767e32a682c247053b16caecedb' },
  doc:
    { name: 'Geeta',
      age: 25,
      Designation: 'Programmer',
      _id: '001',
      _rev: '1-23cf3767e32a682c247053b16caecedb' } },
  { id: '002',
    key: '002',
    value: { rev: '1-d5bcfafbd4d4fae92fd7fc4fdcaa3a79' },
    doc:
      { name: 'Zara Ali',
        age: 24,
        Designation: 'Manager',
        _id: '002',
```

```
    _rev: '1-d5bcfafbd4d4fae92fd7fc4fdcaa3a79' } },  
{ id: '003',  
  key: '003',  
  value: { rev: '1-c4cce025dbd30d21e40882d41842d5a4' },  
  doc:  
    { name: 'Mary',  
      age: 23,  
        Designation: 'Admin',  
        _id: '003',  
        _rev: '1-c4cce025dbd30d21e40882d41842d5a4' } } ]
```

18. PouchDB – Synchronization

You can synchronize the databases stored locally in PouchDB with those that are stored in CouchDB. In the previous chapter, we have seen how to replicate databases using PouchDB. There we have used the method **PouchDB.replicate(source, destination)**.

In addition to this, we can also replicate the data, from the local database to the remote database, and from the remote database to the local database using **replicate.to()** and **replicate.from()** methods as shown below.

```
//Replicating data from local database to remote database
localDB.replicate.to(remoteDB);

//Replicating data from remote database to local database
localDB.replicate.from(remoteDB);
```

Where, **localDB** is an object of database stored locally in PouchDB and **remoteDB** is an object of a database that is stored in CouchDB.

Example

Suppose there is a database with the name **local_database** in PouchDB, and it contains 3 documents, doc1, doc2, and doc3, having contents as shown below.

```
doc1 = {_id: '003', name: 'Ram', age: 26, Designation: 'Programmer'}
doc2 = {_id: '004', name: 'Robert', age: 27, Designation: 'Programmer'}
doc3 = {_id: '005', name: 'Rahim', age: 28, Designation: 'Programmer'}
```

And there is a database with the name **Remote_Database** in CouchDB and it contains 2 documents doc1, doc2, having contents as shown below.

```
doc1 = {_id: '001', name: 'Geeta', age: 25, Designation: 'Programmer'}
doc2 = {_id: '002', name: 'Zara Ali', age: 24, Designation: 'Manager'}
```

Following is an example of synchronizing these two databases, where one is stored in PouchDB and other is stored in CouchDB, using the **replicate.to()** and **replicate.from()** methods.

```
//Requiring the package
var PouchDB = require('PouchDB');

//Creating local database object
var localDB = new PouchDB('local_database');
```

```
//Creating remote database object
var remoteDB = new PouchDB('http://localhost:5984/remote_database');

//Synchronising both databases
localDB.replicate.to(remoteDB);
remoteDB.replicate.from(localDB);
console.log("Databases synchronized successfully");
```

Save the above code in a file with the name **Synchronising_databases.js**. Open the command prompt and execute the JavaScript file using **node** as shown below.

```
C:\PouchDB_Examples >node Synchronising_databases.js
```

This synchronizes the two databases remoteDB and localDB, and displays a message on the console as shown below.

```
Databases synchronized successfully.
```

After synchronizing the two databases visit the http://127.0.0.1:5984/_utils/index.html and select the **remote_database**. You can observe that the documents of local database (003, 004, 005) were copied in this database as shown below.

Key	Value
"001" ID: 001	{rev: "1-23ce3767e32a682c247053b16caecedb"}
"002" ID: 002	{rev: "1-d5bcfa3bd4d4fae92fd7fc9fdcaa3a79"}
"003" ID: 003	{rev: "1-b24619471ac346fdded46cfa8fb23587f"}
"004" ID: 004	{rev: "1-29b82803958c994e3eb37912a45d869c"}
"005" ID: 005	{rev: "1-0eb89f71998ffa8430a640fdb081abd2"}

In the same way, if you fetch the contents of the **local_database** stored in PouchDB you can get to observe that documents of the database that is stored in CouchDB were copied here.

```
[ { id: '001',
```



```

key: '001',
value: { rev: '1-23cf3767e32a682c247053b16caecedb' },
doc:
{ name: 'Geeta',
age: 25,
  Designation: 'Programmer',
  _id: '001',
  _rev: '1-23cf3767e32a682c247053b16caecedb' } },
{ id: '002',
key: '002',
value: { rev: '1-d5bcfafbd4d4fae92fd7fc4fdcaa3a79' },
doc:
{ name: 'Zara Ali',
age: 24,
  Designation: 'Manager',
  _id: '002',
  _rev: '1-d5bcfafbd4d4fae92fd7fc4fdcaa3a79' } },
{ id: '003',
key: '003',
value: { rev: '1-bf4619471ac346fdde46cfa8fbf3587f' },
doc:
{ name: 'Ram',
age: 26,
  Designation: 'Programmer',
  _id: '003',
  _rev: '1-bf4619471ac346fdde46cfa8fbf3587f' } },
{ id: '004',
key: '004',
value: { rev: '1-29b8f803958c994e3eb37912a45d869c' },
doc:
{ name: 'Robert',
age: 27,
  Designation: 'Programmer',
  _id: '004',
  _rev: '1-29b8f803958c994e3eb37912a45d869c' } },
{ id: '005',
key: '005',
value: { rev: '1-0eb89f71998ffa8430a640fdb081abd2' },

```

```
doc:
{ name: 'Rahim',
age: 28,
  Designation: 'Programmer',
  _id: '005',
  _rev: '1-0eb89f71998ffa8430a640fdb081abd2' } } ]
```

You can rewrite the above program using the **sync()** method provided by PouchDB instead of the two methods **replicate.to()** and **replicate.from()** as shown below.

```
//Requiring the package
var PouchDB = require('PouchDB');

//Creating local database object
var localDB = new PouchDB('local');

//Creating remote database object
var remoteDB = new PouchDB('http://localhost:5984/remote_database');
//Synchronising Remote and local databases
localDB.sync(remoteDB, function(err, response) {
  if (err) {
    return console.log(err);
  }
  else {
    console.log(response);
  }
});
```

On executing the above program, it synchronizes the two databases displaying the following message.

```
{ push:
{ ok: true,
start_time: Fri Mar 25 2016 15:54:37 GMT+0530 (India Standard Time),
docs_read: 6,
docs_written: 6,
doc_write_failures: 0,
```

```
errors: [],
last_seq: 10,
status: 'complete',
end_time: Fri Mar 25 2016 15:54:37 GMT+0530 (India Standard Time) },
pull:
{ ok: true,
start_time: Fri Mar 25 2016 15:54:37 GMT+0530 (India Standard Time),
docs_read: 0,
docs_written: 0,
doc_write_failures: 0,
errors: [],
last_seq: 2,
status: 'complete',
end_time: Fri Mar 25 2016 15:54:37 GMT+0530 (India Standard Time) } }
```

19. PouchDB – Miscellaneous

In this chapter, we will discuss the concepts like, compaction and retrieval of bulk data from PouchDB.

Compaction

You can reduce the size of a database by removing the unused data using **compact()** method. You can compact a local database as well as remote database using this method.

Following is an example demonstrating the usage of the **compact ()** method in PouchDB.

```
//Requiring the package
var PouchDB = require('PouchDB');

//Creating the database object
var db = new PouchDB('sample_database');

db.compact(function (err, result) {
  if (err) {
    return console.log(err);
  } else {
    console.log(result);
  }
});
```

Retrieving Bulk Data

You can retrieve a set of documents in bulk using the **bulkGet()** method. To this method, you need to pass a set of id's and _rev's.

Following is an example demonstrating the usage of the **bulkGet()** method in PouchDB.

```
//Requiring the package
var PouchDB = require('PouchDB');

//Creating the database object
var db = new PouchDB('my_database');
```

```
//Preparing documents
//Inserting Document
db.bulkGet({docs: [
  { id: "001", rev: "1-5dc593eda0e215c806677df1d12d5c47"},
  { id: "002", rev: "1-2bfad8a9e66d2679b99c0cab24bd9cc8"},
  { id: "003", rev: "1-7cff4a5da1f97b077a909ff67bd5b047"} ]}, function(err,
result) {
  if (err) {
    return console.log(err);
  }else {
    console.log(result);
  }
});
```