ULB

Couchbase

Document Store

Carlos Martínez Lorezno
Pablo Molina Mata

Content

1) Document Store

a) Chronological

In the early 70s, relational databases were introduced and for that time data schemas were quite simple, and it was suitable to think about objects as sets of relationships between them.

Structured Query Language, or SQL, was used and developed because those relationships between the different types of data were specified in the Database schema. But as time passed, the environment for data and programming has changed in different aspects:

- The emergence of cloud computing has brought deployment and storage costs down dramatically, but only if data can be spread across multiple servers easily without disruption. Executing distributed joints is a quite complex problem in relational databases.
- The need to store unstructured data, such as post in social websites or multimedia, has grown rapidly, and in SQL databases workarounds or compromises have to be made for storing and querying unstructured data.
- As demands evolve, the database schema needs to change rapidly and easily, and in SQL databases the structured needs to be specified in advanced.

As way of responding to these changes, new manners of storing data have emerged in which data can be grouped together more easily and logically. One of this new way of storing data is Document Store Databases, and it's going to be discussed in the following paragraphs.

b) Characteristics

A Document Store Database (DSDB), also named as “Document-oriented Database”, “Aggregate Database” or simply as “Document Database”, is a database that uses a document-oriented model to store data.

A DSDB is used for storing, retrieving and managing semi-structured data, but unlike traditional relational databases, the data model is no longer structured in a table format of rows and columns. In this case, the schema can vary, providing a major degree of flexibility for data modelling than relational databases.

Regarding of the way that DSDB works, it is usually used as the “document”, a block of XML and JSON, YAML or BSON among others. As it is mentioned before, instead of columns with names and data types used in relational databases, the document contains a description of the data type and the correspondent value for that description.

Furthermore, there is no need for each document to have the same structure, it can be different. For adding extra types of data to a DSDB, it is no longer needed to modify the entire database schema, as it is necessary with a relational database. It is possible to add the object directly to the database.

In DSDB, documents are grouped into “collections” or “groups”, which mission is similar to relational tables. The database provides a query mechanism to search into these groups, for documents with attributes.

c) Differences between document store databases and relational databases

Some of them are mentioned briefly in previous paragraphs, but here is a closer look at them:

Relational databases way of storing data is by using tables, columns and rows (representing each record). The only manner of establishing an association between two or more tables is by creating a relationship between them.

In the other hand, DSDBs store data on a provided entity within a single document, and any associated data is stored inside that one document

With traditional relational databases, before loading any data, it is mandatory to create a schema. But, with DSDBs it is no longer compulsory, so data can just be loaded without any predefined schema. Therefore, any two documents can contain a different structure and data type.

In terms of scalability, document databases can scale horizontally quite well, meaning data can be stored over several thousand of computers and the performance of the systems is good.

Meanwhile, relational databases are more capable of scaling vertically. Although, there is a limit to how many resources can be stored inside one machine, there would be a point where horizontal scaling has become the only option.

Traditional relational databases establish relationships between tables by using foreign keys. DSDBs, on the other side, don't have implemented this option so if a relationship needs to be done, it would be compulsory to be implemented at the application level. But, as the whole idea in document model is that any data associated with a record is stored within the same document, the need of establishing a relationship should not be as prevalent.

Document store databases are quite like key-value databases, but the main difference lies within the "value" of the DSDB. The "value" contains structured or semi-structured data, and can be encoded using a widely number of methods, including XML, JSON, BSON, YAML, etc. And the main benefit is that, with the key-value database you can't query within the value, meaning you get the whole value no matter how big it might be. In contrast, with DSDB, you can query against the structure of the document as well as the elements within that structure. Therefore, you can return only those parts of the document that you required.

d) Main uses for document store databases

Thanks to their characteristics, document-oriented databases are well suited for a wide variety of use situations:

They are used for web applications, such as content management system, blogging platforms, Web analytics, user preferences data, etc, due to their capability of making changes easily through different sources, at the same time without duplicating the content.

Also used in Used generated content, like chat sessions, tweets, blog posts, ratings, comments, etc, thanks to their schema-less design, their highly flexible data model and their quickness.

In Catalog Data aspects, for example user accounts, product catalogues, devices registries for Internet of Things, it's necessary to access a region of the information (so you have to query the value of the document) and you need to do it quickly. Due to these reasons, document store database is the correct choice.

Very used in Gaming aspects, like in-game stats, social media integration, high score leader boards, in-game chat messages, challenges completed, etc, thanks again to their quickness in getting the information and the availability of the information itself (by using more servers to scale out).

e) Examples

MongoDB development begun in 2007 and was launched in 2009. The main features of this database are: Ad hoc queries, indexing, replication, load balancing, file storage, aggregation, server-side JavaScript execution, Capped collections and Transactions.

Amazon DynamoDB was released in 2012 and it has a particularity that Dynamo had a multi-master design requiring the client to resolve version conflicts and uses synchronous replication across multiple datacenters for high durability and availability.

Couchbase is the database chosen for this particular project and in the next paragraphs it is going to be explained more deeply.

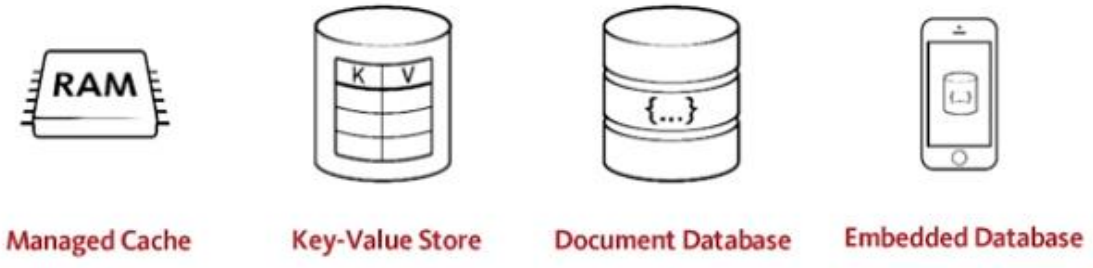
2) Couchbase

a) What is Couchbase?

Couchbase is one of the most powerful NoSQL technology, it is the first NoSQL Database to enable you to develop with agility and operate at any scale. Couchbase database emerges from the merge of two products; several leaders of the Memcached project (a distributed memory caching system) founded Membase, and some of these leaders merge with people of CouchDB (a distributed memory caching system of Apache) in order to create Couchbase Server.

Couchbase is a memory first solution (distributed cache and the backend are the same), each time that you interact with Couchbase you write or read directly from memory. From CouchDB, Couchbase has inherited the key-value architecture and each value has a maximum size of 20 MB, so you can use Couchbase like a Key-Value store. However, the most typical way is to use Couchbase like a document store (a document is a JSON file), in this way you can access to the documents with key-value, with index access or with N1QL (It is an implementation of SQL in JSON).

Couchbase has two main product, Couchbase server and Couchbase mobile (an implementation of Couchbase server for embedded systems). Moreover, Couchbase can work as a distributed database as well, but in this topic, we are going to be focus on Couchbase Server and this behaviour as a document store.



[1] Picture 1. Main Couchbase Characteristics.

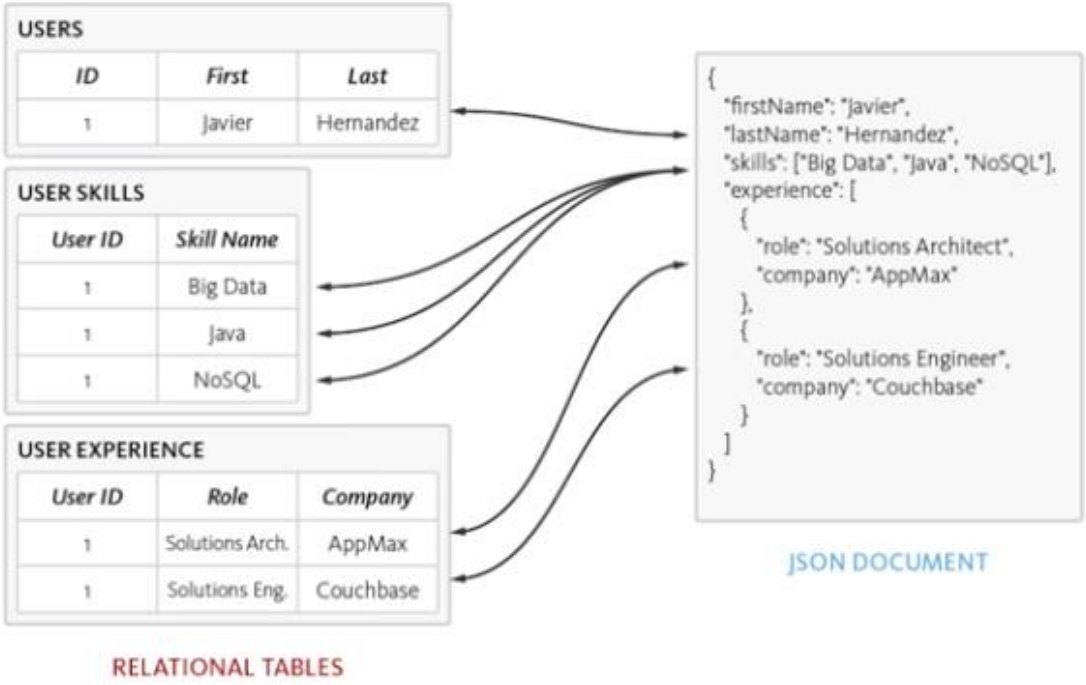
[1] Couchbase Server. Distributed Data Management. Consulted on 2/11/2018, in https://developer.couchbase.com/documentation/server/5.0/concepts/distributed-data-management.html.

b) What is a JSON document?

JSON is the de facto standard in the frontend word, therefor Couchbase stores JSON documents instead of relational table. With JSON files, you do not need to access to different table to get the information, you have all the information that you need in the same document.

In a JSON document you do not have a specific structure. For example, in a relational database you have structure specify in the tables, all the tuples from a table have the same attributes and when some tuple do not have an attribute you are going to put this attribute to null. In a document store you do not have a structure so for each tuple of one type of document you can have different attributes, you can have missing or null attributes.

Moreover, you do not have relationships, if you want to relate two different documents, you are going to create an attribute in the document to refer to the primary key or another attribute of the other document.



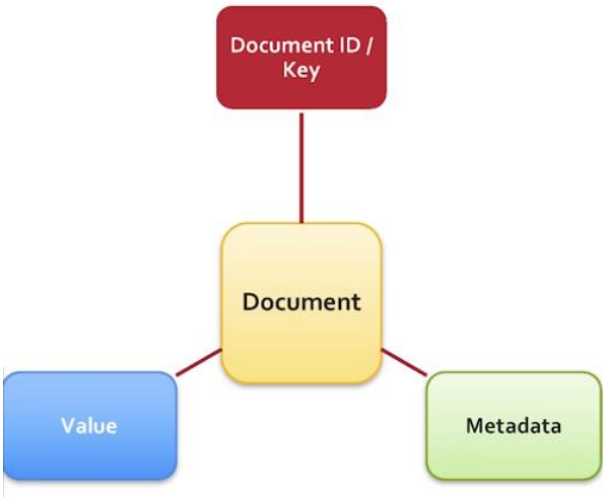
[2] Picture 2. Relational Table vs JSON Document

In the previous picture we can see how in relational tables you are going to repeat a lot of time the same information. In “USER SKILLS” you have 3 tuples to express skills of the same user and other 2 tuple to express the “USER EXPERIENCE”, while in the a JSON document you can write inside the document of the USER all this information like a simple array of strings or an array of complex object. You are not going to repeat the User ID 5 times.

[2] Couchbase Server. Couchbase. Consulted on 20/11/2018, in https://www.couchbase.com.

Each document has three important things:

- Document ID or key, that is like primary keys in relational databases. Documents are portioned based on the document ID. ID based document lookup is extremely fast and this document ID must be unique, and you can generate it automatically.
- Value is the information that you want to store.
- Metadata, this is all the metainformation relate to the document so that the database administrator system can read and understand the document.



[3]Picture 3. Parts of JSON document

JSON is the most typical way to share information in the web, currently most of the calls share JSON files, so it is better to have a document store database based on JSON files, since it is very simple to communicate with web technologies.

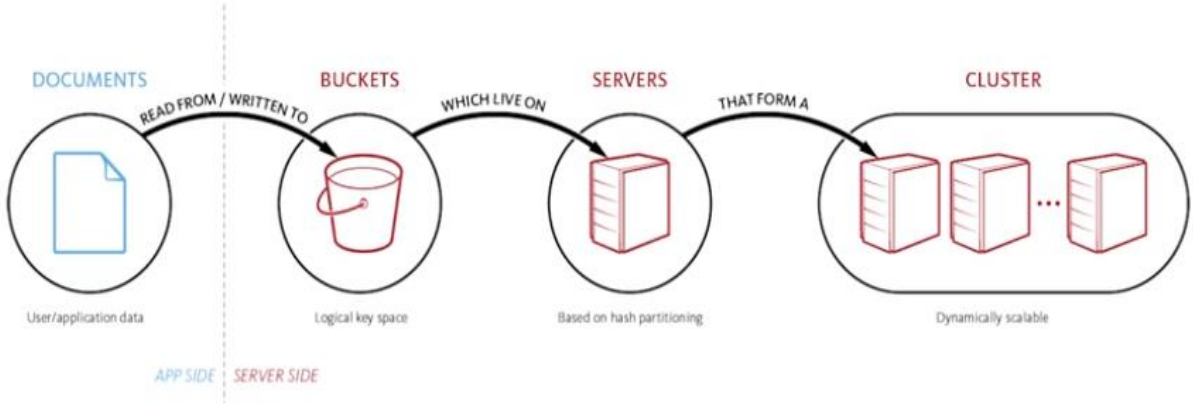
[3] Couchbase Server. Couchbase. Consulted on 20/11/2018, in https://www.couchbase.com

c) What should you know about Couchbase?

Buckets are one of the most important things in CouchDB, it is the keyspace, what would be a scheme in the relational databases. In a bucket, you are going to store all the different object that you need. All the documents that you are going to store in a bucket do not have to have the same structure, you can have a lot of different types of document. However, you might be interested in have some different buckets in order to reduce the number of objects of a bucket and improve the searching in the bucket. Buckets are very important since you can replicate all the information to other buckets with a single instruction, so from a scalability view, this is very powerful.

Then, the bucket is going to be in a specific server, that a group of these servers are going to form a cluster. With Couchbase is very simple to scale, since you only need to create a new server and if you need some bucket here, with a simple query you can replicate all you want.

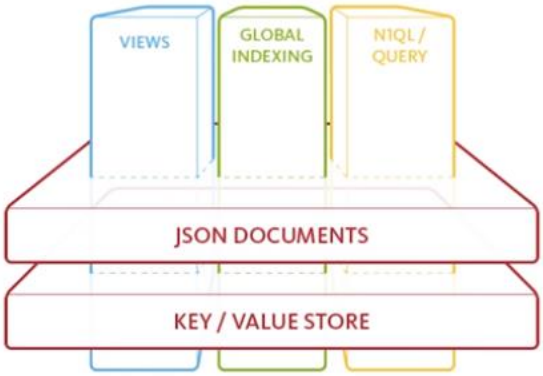
Moreover, the scalability is very good as well because Couchbase is a distributed database too, but we are not going to speak here about Distributed Databases since this report is focus on Document Store Databases, there will be other reports explaining the Distributed Databases.



[4] Picture 4. Couchbase Architecture.

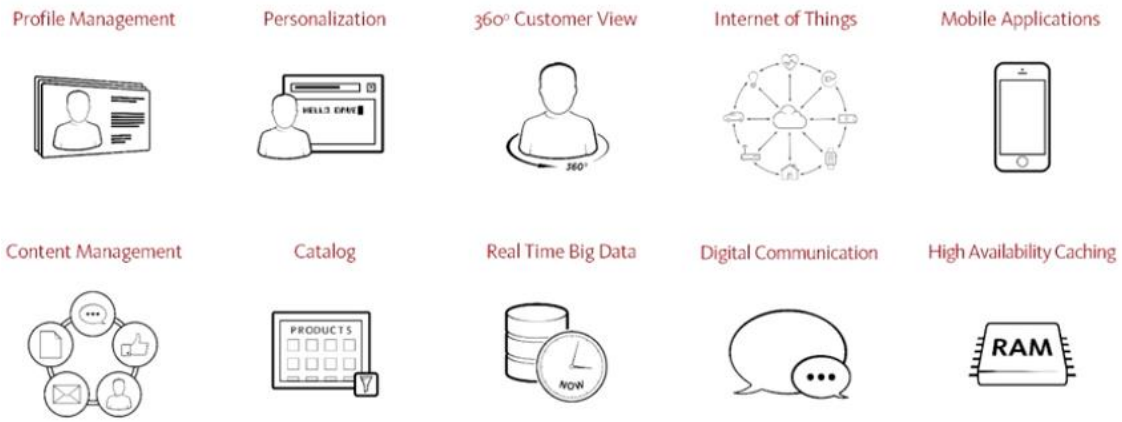
[4] Couchbase Server. Distributed Data Management. Consulted on 2/11/2018, in https://developer.couchbase.com/documentation/server/5.0/concepts/distributed-data-management.html.

Couchbase allow you to access to all their documents in some different ways. The simplest and faster way is the key-value access like in all the database systems, since a Document store is a special case of key-value. Then, there are the views, that provide aggregation and real-time analytics through incremental map-reduce. Nevertheless, the most important access ways are the Global Secondary Indexes or N1QL, this provide low latency and high throughput indexes. N1QL is a language that provides a powerful and expressive way of accessing documents, that is an SQL for native JSON. The learning curve of this language is very small since almost everyone knows SQL.



[5]Picture 5. Couchbase Searching Document

The most important use cases of Couchbase are profile management, personalization, internet of Things, mobile Applications, content management, product catalogues, real time big data, digital communication or high availability caching.



[6]Picture 6. Couchbase most typical use cases

[5] Couchbase Server. Couchbase. Consulted on 02/11/2018, in https://www.couchbase.com.
[6] Couchbase Server. Couchbase. Consulted on 24/11/2018, in https://www.couchbase.com.

Couchbase is quite good in these different use cases because is a Document Store (we have explained it why in the previous chapter). However, there are some different use cases that Couchbase is good unlike the other Document Store like MongoDB. For example, Couchbase is good in the use cases related with internet of things or Real Time Big Data because Couchbase has a Distributed Architecture and has a good performance in scalability, you only need to add new served pressing a single bottom.

Moreover, Couchbase is better than the other Document Store in mobile system, since most of the Document Store Databases do not have implement any solution for mobile device, so if you are going to create an application, you do not have a lite database in the device, you need internet connection all the time. In Couchbase, you have a mobile device solution, so you have your lite database in the mobile phone that you do not need to be all the time connected to interned and this is going to perform so much the application since the application has the data in the same device.

d) Learning N1QL

N1QL is a native SQL for JSON, so you are going to get the advantages of SQL (general query functionality and queries across relationships) with advantages of JSON document (rich structure with a flexible schema that is easy to change). So N1QL has the standard SELECT syntaxis of SQL; select, from, where, order by, group by, limit or offset are the same that in SQL.

EXAMPLE:

```
SELECT ***** FROM ***** WHERE *****
```

Moreover, N1QL allows to do:

- JOINs: this is important since there are some document store databases that not allow you to do it, for example MongoDB.
- Subqueries.
- Aggregations.
- UNION, INTERSECT or EXCEPT.
- Etc.

In the write syntaxis, there are UPDATE, DELETE, INSERT and MERGE (Same in SQL).

We are not going to explain these standard SQL things, since if you are reading this report is because you know SQL, so we are going to talk only about the particularities of N1QL. One of the particularities is that has expression only for JSON, in JSON you can get null values for one document, but you can have missing values as well, so N1SQL differentiate it with "IS MISSING" and "IS NULL". You can try all the examples in https://query-tutorial.couchbase.com/tutorial.

```
SELECT name
FROM person
WHERE children IS NULL

OR

SELECT name
FROM person
WHERE children IS MISSING
```

If you want to filter based on arrays nested inside the document you can use ANY or EVERY, depending on if you need that all the elements of an array or at least one satisfies any conditions.

```
SELECT name
FROM person
WHERE ANY/EVERY child IN person.children SATISFIES child.age > 15  END
```

Specific primary keys within a bucket can be queried using the USE KEYS clause.

```
SELECT name
FROM person
USE KEYS ["John", "Ross"]
```

Slicing an array.

```
SELECT children[0:2]
FROM person
WHERE children[0:2] IS NOT MISSING
```

Array comprehensions, if you do not want to show everything of an array.

```
SELECT
        name,
        ARRAY child.name FOR child IN person.children END AS nameOfChild
FROM person
WHERE children IS NOT NULL
```

JOINs.

```
SELECT A.accountID, A2.accountID
FROM  Account A
INNER JOIN Account A2
ON A1.ownerID=A2.ownerID;
```

INSERT.

```
INSERT INTO person (KEY, VALUE)
VALUES ("1234", {"name":"Pedrito", "age":"31"})
```

DELETE.

```
        DELETE FROM person p
        USE KEYS "1234" RETURNING p
```

UPDATE.

```
UPDATE person
USE KEYS "1234 "
SET age = 32 RETURNING person.name
```

e) Why should you use Couchbase instead of MongoDB?

Now, we are going to compare Couchbase with MongoDB since MongoDB is the most used document store database. In this way we are going to understand the advantages of using Couchbase.

Performance at scale

The performance of MongoDB quickly degrades as increase the number of user or clusters. It is due to its master-slave architecture, that limits its ability to perform a lot of concurrent users in the same node. Moreover, Increase the hardware resources is not the solution because the database is going to be more complex and it does not improve so much the performance.

With Couchbase everything is different, its distributed architecture without master is simple and deliver consistent performance at any scale, Couchbase support so many concurrent users in the same node and scales horizontally across multiple nodes. Setting up more cluster do not take so much time and you can add or remove node pressing a simple button.

Mobile

MongoDB does not have a solution to support mobile phone applications. In MongoDB you have to create your own code to synchronise data on the mobile device with the data in the remote server, so a MongoDB application has to have always a good internet con in order to works properly, this is not good for the performance. However, Couchbase has a complete mobile solution (Couchbase Mobile), this solution has an embedded JSON database and a synchronization solution, so the application does not need internet connection all the time, it can store the data in the JSON database and then, when it has connection, synchronize with the server.

High availability and disaster recovery

with MongoDB, if you are developing an application with multiple data centers, MongoDB is only going to perform writes in the main data center, other data centers have to perform writes to remote locations. Therefore, MongoDB cannot perform locally all writes and can suffer from data loss and inconsistency.

On the other hand, Couchbase has a distributed architecture without master, you can perform locally all writes, so it is going to reduce the latency and to improve the performance of the application. Moreover, Couchbase has different replication routines that reduce data loss and inconsistency problems.

Queries

We have explained in this report N1QL, the database query language of Couchbase, so if we compare Couchbase queries with MongoDB queries, we are going to see that Couchbase queries are simplest and faster to do.

In this part of the comparison with MongoDB, we prefer to show the differences with an example. Database schema:

Orders:

- Name: String
- Category: String
- Value: Integer
- Quantity: Integer

N1QL

```
SELECT category, SUM(value * quantity) AS total,
FROM   orders
WHERE  category IN ( "1", "2" "3","4") AND value > 100
GROUP  BY category
ORDER  BY category ASC, SUM(value * quantity) DESC
```

MongoDB

```
database.order.aggregate([
        { "$match": {
                "$and": [
                    {"category": {
                            "$in": [
                                "1",
                                "2"]}},
                    { "value": {
                            "$gt": 0 }}]}},
        { "$group": {
                "_id": {
                    "symbol": "$category" },
                "sum(value * quantity)": {
                    "$sum": {
                        "$multiply": [
                            "$value",
                            "$quantity"]}}}},
        { "$project": {
                "_id": 0,
                "sum(value * quantity)":  "$sum(value * quantity)",
                "symbol": "$_id.symbol"}}
        { "$sort": {
                "category": 1,
                "sum(value * quantity)": -1 }}])
```

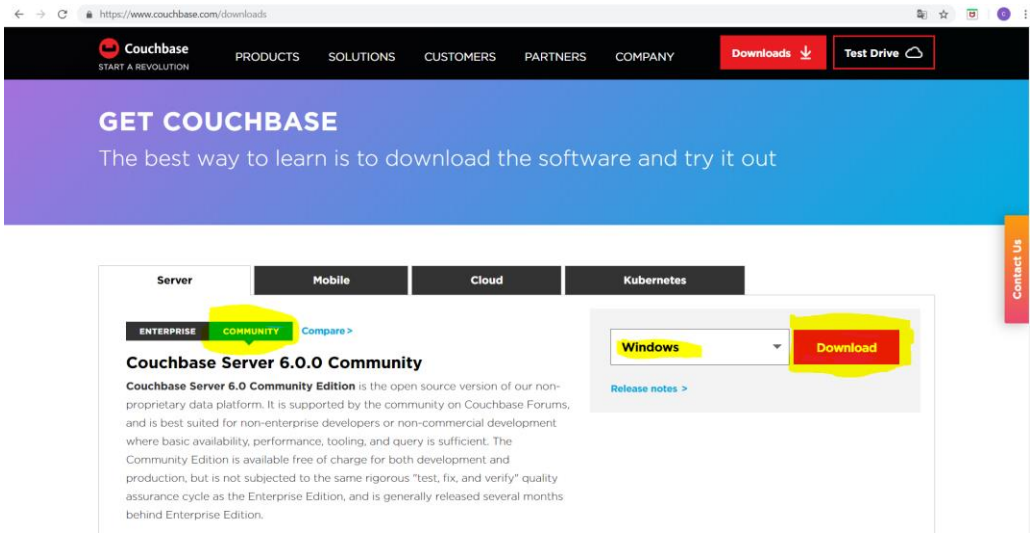
From our point of view this is the best advantage of using Couchbase in front of MongoDB, the learning curve of Couchbase is very soft in comparison with MongoDB learning curve. You can start to do queries without learning anything since it is very similar to SQL. Moreover, with Couchbase you can some SQL thing that you cannot do in MongoDB, like JOINs.

Overall, with Couchbase you are going to avoid scale problems, you do not need to be always connected to the server, you are not going to lose data and the most important thing, you can use N1QL.

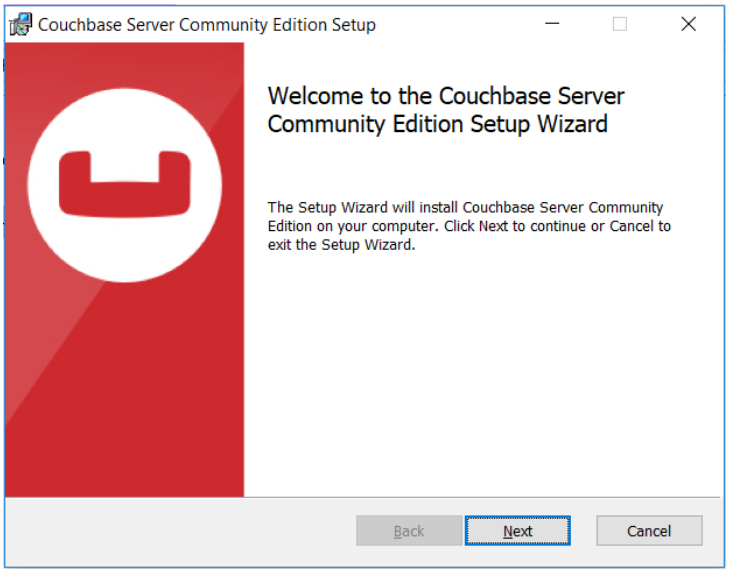
f) How install Couchbase

Installing Couchbase is very simple and you do not need to install anything more since Couchbase has integrated their own management console. So, you only must follow the next steps.

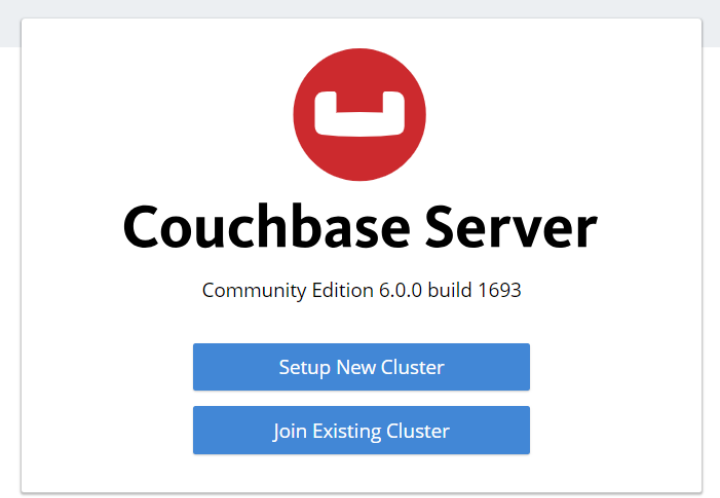
1. Download the latest version of Couchbase from https://www.couchbase.com/downloads. We have installed Couchbase Server 6.0.0 Community for Windows.



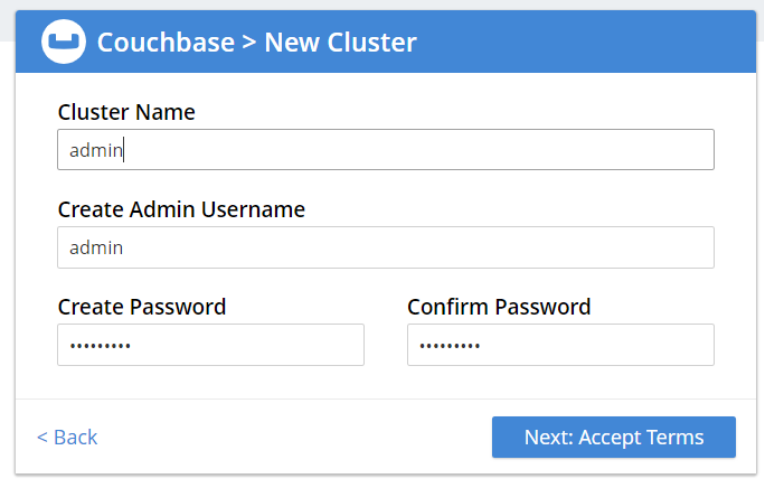
2. Execute the installer (accept all the terms).



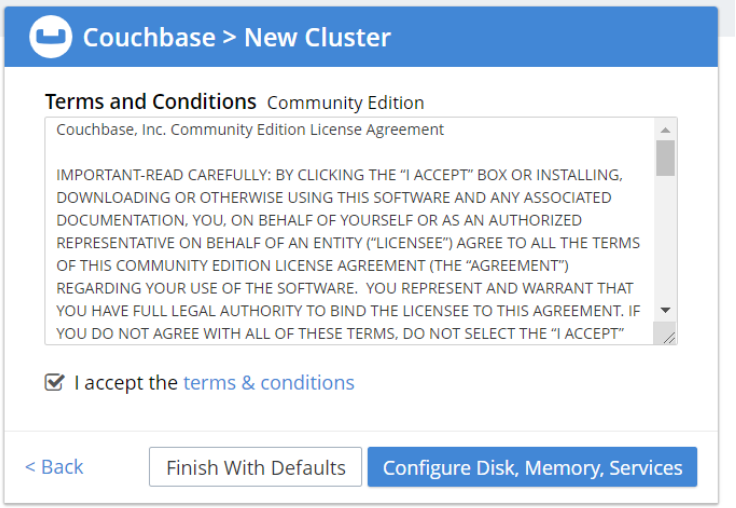
3. Open the management console in your browser, http://localhost:8091, and setup a new cluster.



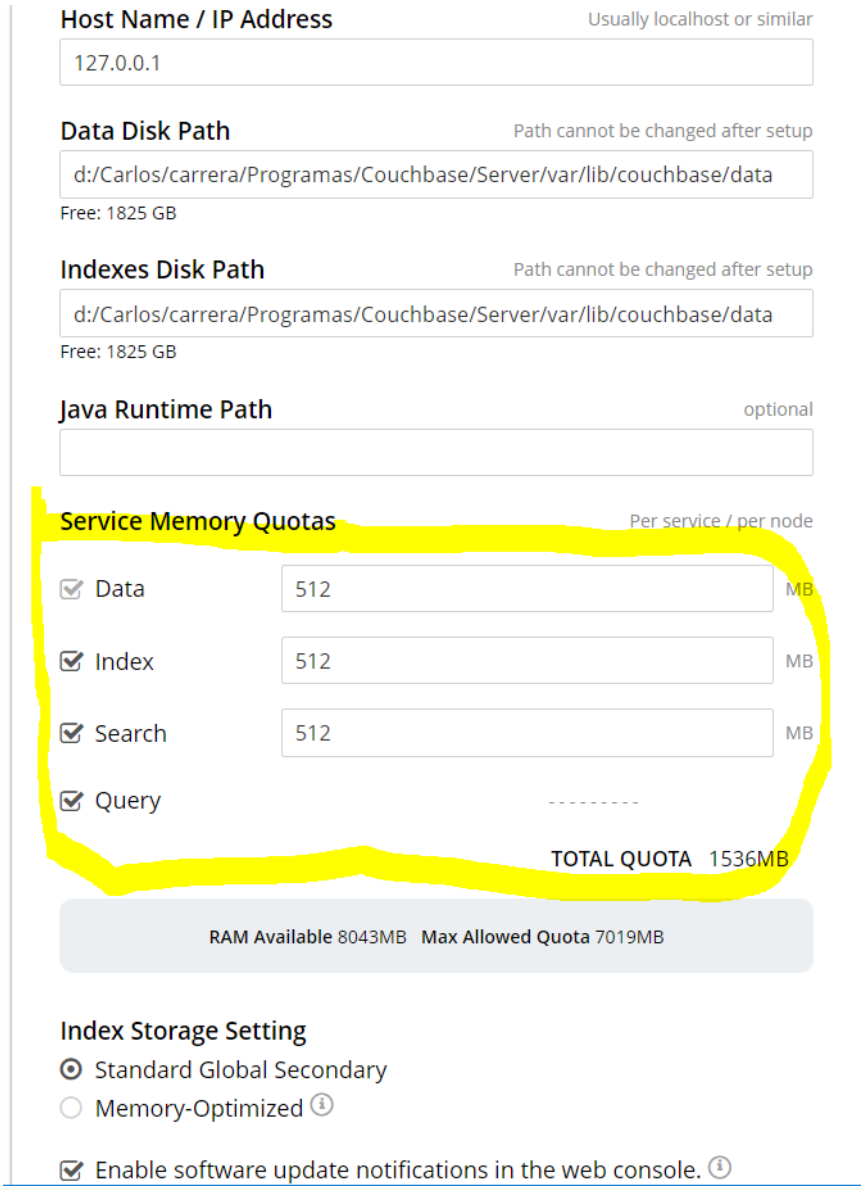
4. Create a new cluster and an admin user.



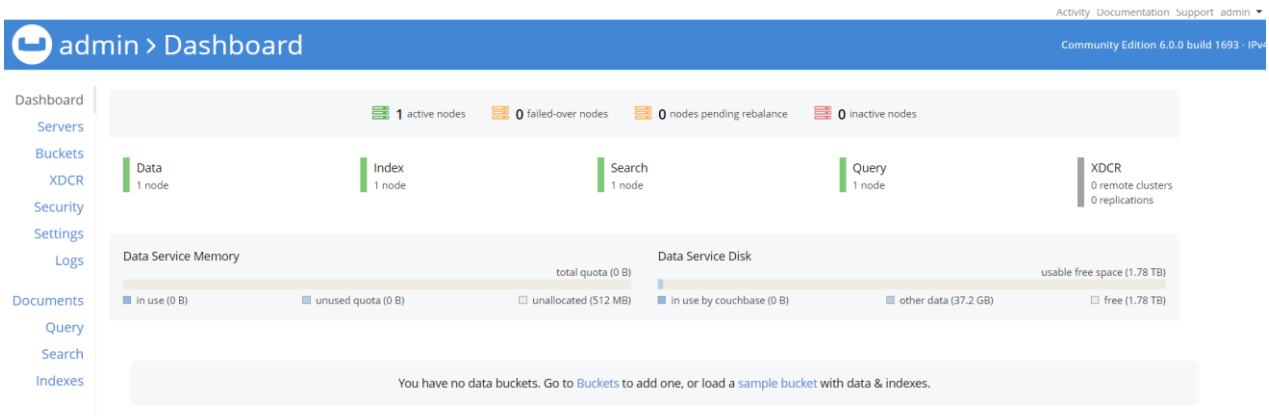
5. Accept terms without reading anything.



6. Then configure all the parameters and think carefully how much memory you want to use!



7. Everything is installed!

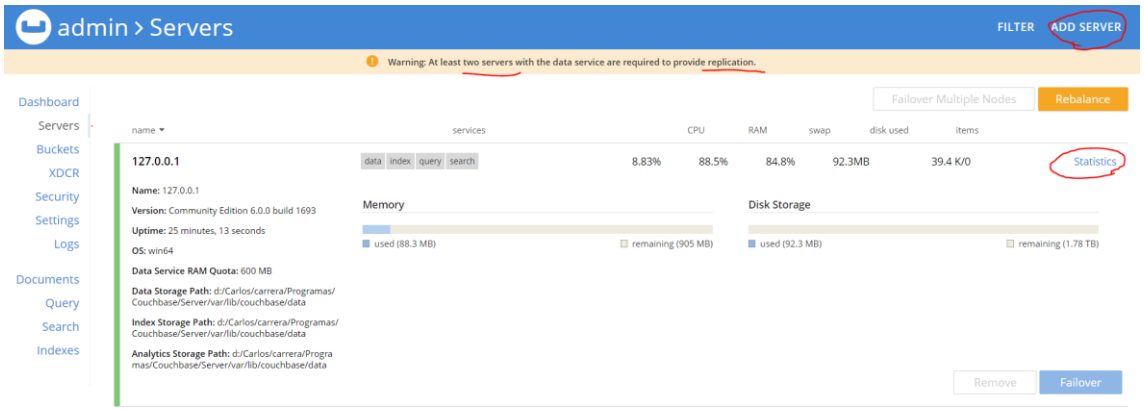


3) Playing with Couchbase

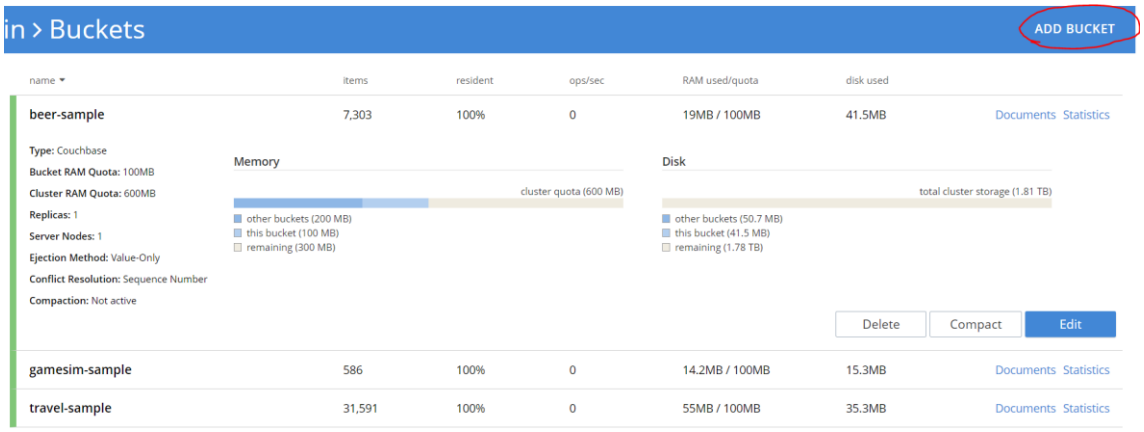
a) Couchbase console

Now we are going to learn to use Couchbase with its console. When you access and log to the console (http://localhost:8091) you are in the dash board (it is the last picture of the previous page), here you can see the general information of the database like the total quota that occupied in memory or in the disk. In the left, there is a control panel.

After dashboard, the second option is “Servers”, here you can create new server and to see a lot of different statistics. Moreover, if you want to replicate the server, you can do it here as well.

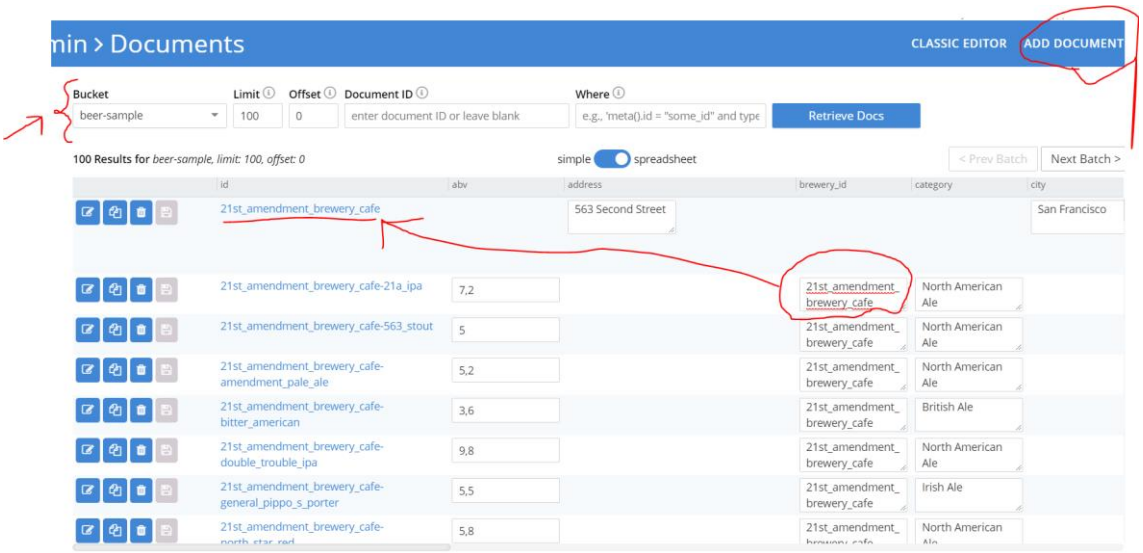


Then, in Buckets you can manage all your different bucket in your database, you can add a new one and delete or edit your current buckets.

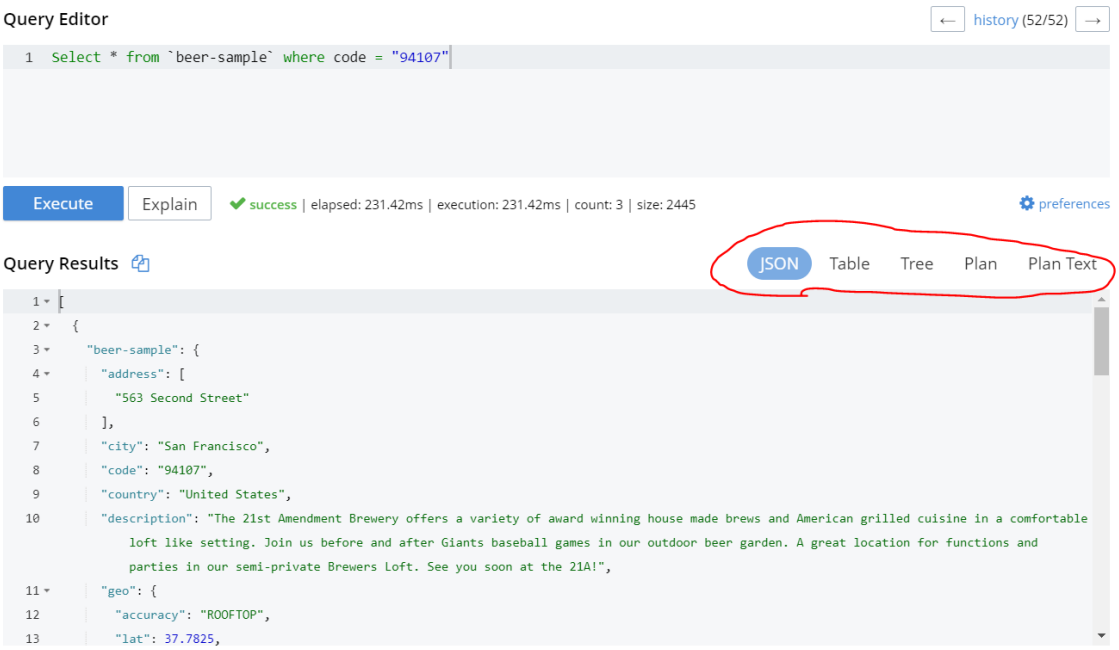


Other important tab is log, where you can see all the different actions that happen in the database.

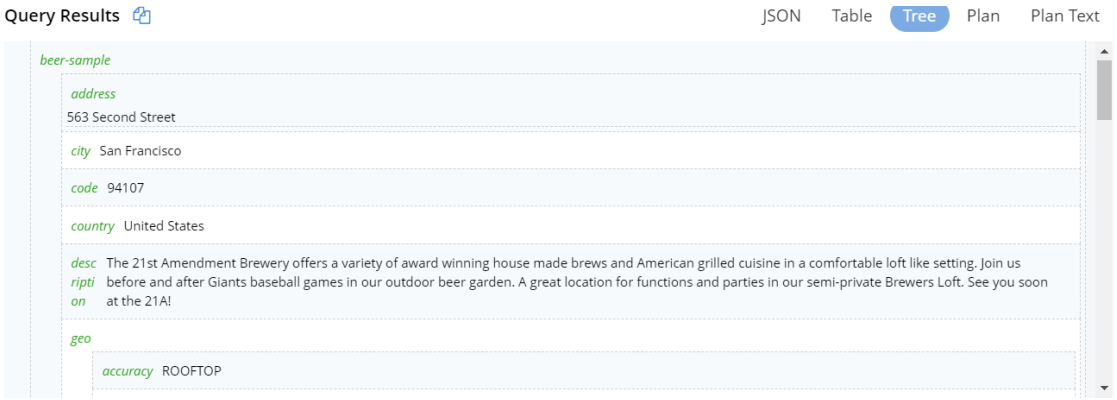
In Document you are going to find all the different document in your buckets. So, you must select a specific bucket and retrieve all their documents. You have different option here since you can limit the number of documents that you want to show per page, the offset is the number of the first page that you want to skip and the you can filter by document ID that it is fast or filter by the typical SQL where. Moreover, you can add new documents and edit in a very simple way all the documents without using any complex query.



However, you can execute queries as well in Query tab, here you are going to write the query with N1QL, if you do not know N1QL you must read the part d of the chapter 2 of this report or if you want to know more complex things you can access to https://query-tutorial.couchbase.com/tutorial/#1. The console is going to show you the query result as a JSON file, but you can change this view.



The console is going to show you the query result as a JSON file, but you can change this view. You can show the information in tables or in a tree as well.



b) Working with Couchbase and Nodejs

a. Connection to Couchbase

You can use Couchbase with a lot of different technologies like Java or Nodejs and the connection is always very simple to do. In this repost we are going to suppose that you have knowledge of JavaScript. We are going to explain how connect Nodejs with Couchbase, if you need to know how connect with other technology, you can find all the information in the official web site of Couchbase. If you want to connect to Couchbase you must connect to the cluster and authenticate yourself.

```
var couchbaseConnection = require('couchbase');
var cluster = new couchbaseConnection.Cluster('couchbase://10.0.0.1');
cluster.authenticate('admin', 'admin');
```

After that, you have to connect to the bucket that have your documents using the function openBucket of the bucket.

```
var beersBucket = cluster.openBucket('beers-sample', function(err) {
    if (err) {
      console.error('Got error: %j', err);
    }
  });
```

If you want to disconnect to bucket you have to use only the function disconnect.

```
beersBucket.disconnect();
```

b. Crud

For inserting a new document in the bucket, you only must use insert function of the bucket and write the name of the document and then you have to write in JSON the document.

```
  beersBucket.insert('kaiser', {'avg': '7.4'}, function(err, result) {
    if (!err) {
        console.log("Stored");
    } else {
        console.error("Error: %j", err);
    }
});
```

You can use the function “get()” of bucket to get the document using its primary key. If you want to delete the document you are going to use Bucket.remove() function with the primary key of the document.

```
  beersBucket.get('key-of-the-documment', function(err, result) {
    if (err) {
        console.log('Error: %j', err);

    } else {
        console.log(result.value);
    }
});
```

In Couchbase you can use N1QL language, that we have explained it before, you can create the queries in a simple way:

```
var N1qlQuery = couchbase.N1qlQuery;
  query = N1qlQuery.fromString('SELECT * FROM beers-sample');

  beersBucket.query(query, function(err, rows, meta) {
    for (row in rows) {
      console.log(row);
    }
  });
```

The result of the query is inside the variable rows, and inside the function you can use this row, in the previous example, we are showing all the document of the beer-sample bucket.

c) Executing Queries

In a previous chapter we have shown the syntax of N1QL, now we are going to execute different queries and analyse the results. Couchbase gives us 3 different set of data which we can use to learn N1QL and to analyse the performance of Couchbase. Beer-sample is a data set of 7.303 items, gamesim-sample is a data set of 586 items and travel-sample is a data set of 31.591 items.

name	items
beer-sample	7,303
gamesim-sample	586
travel-sample	31,591

So first of all, we are going to execute some queries of beers-sample and analyse it and the we are going to execute some queries for the other data set in order to compare the performance with different number of items. The beer-sample only have two different type of items, the beers and the breweries.

We are going to start with a simple query, that have to find the brewery named “Aass Brewery”.

```
SELECT address, city, country
FROM `beer-sample`
WHERE type="brewery" and name = 'Aass Brewery';
```

This query has spent 204.54 milliseconds and the result is:

```
[
  {
    "address": [
      "Ole Steensgt. 10 Postboks 1530"
    ],
    "city": "Drammen",
    "country": "Norway"
  }
]
```

The next query we are going to filter with a range condition and with a string condition.

```
SELECT name, address, city, country
FROM `beer-sample`
WHERE type="brewery"
  AND geo.lat > 59.0
  AND name LIKE '%Brew%'
ORDER BY name;
```

It has found 8 items and it has spent 204.51 milliseconds, we are going to show only the first result.

```
{
    "address": [
      "Ole Steensgt. 10 Postboks 1530"
    ],
    "city": "Drammen",
    "country": "Norway",
    "name": "Aass Brewery"
  },
```

In the next query we are going to do a join between beers document with their breweries. It is very simple since beers documents keep the id of the brewery document, so you only have to do the join in brewery_id. You do not need to specify what type of document is it because only the beer documents have the attribute brewery_id.

```
SELECT bw.name AS brewer, br.name AS beer
FROM `beer-sample` br join `beer-sample` bw
    on keys br.brewery_id
ORDER BY bw.name, br.name
LIMIT 3;
```

The execution time is 240 milliseconds.

```
  {
    "beer": "(512) ALT",
    "brewer": "(512) Brewing Company"
  },
  {
    "beer": "(512) Bruin",
    "brewer": "(512) Brewing Company"
  },
  {
    "beer": "(512) IPA",
    "brewer": "(512) Brewing Company"
  }
]
```

Now we are going to use a subquery, we want to know the number of different beers that are made in whatever City of Belgium. So, we are going to select all the name of breweries in Belgium and then filter by these names.

```
SELECT count(DISTINCT br.name) AS numberOfBeers FROM `beer-sample` br
INNER JOIN `beer-sample` bw  ON KEYS br.brewery_id
WHERE bw.name WITHIN
    (SELECT name FROM `beer-sample`
    WHERE type = "brewery" AND
    country="Belgium");
```

This is not the best way to do it, since is faster filtering in the whereby “brewery.country = Belgium”, but we want to analyse here the execution time with a subquery.

The execution time is 1.31 seconds.

```
[
  {
    "NumberOfBeers": 321
  }
]
```

Now we are going to insert a new document in the database. We are going to introduce a new Spanish beer, “Alhambra verde” that is made by cervezas_alhambra.

```
INSERT INTO `beer-sample` ( KEY, VALUE )
  VALUES
  (
    "Cerveza_alhambra-verde",
    {"name": "Victoria", "abv": 4.0, "ibu": 0.0, "srm": 0.0, "upc": 0,
     "type": "beer", "brewery_id": "cervezas_alhambra",
     "updated": "2010-07-22 20:00:20", "description": "Delicious Spanish
beer",
      "style": "Light Beer", "category": "Spanish Pilsner"}
  )
RETURNING META().id as docid, *;
```

Execution time is 125 milliseconds.

```
[
  {
    "beer-sample": {
      "abv": 4,
      "brewery_id": "cervezas_alhambra",
      "category": "Spanish Pilsner",
      "description": "Delicious Spanish beer",
      "ibu": 0,
      "name": "Victoria",
      "srm": 0,
```

```
      "style": "Light Beer",
      "type": "beer",
      "upc": 0,
      "updated": "2010-07-22 20:00:20"
    },
    "docid": "Cerveza_alhambra-verde"
  }
]
```

Update a document is very easy, it is the same that in SQL. The execution time is 768 milliseconds.

```
UPDATE `beer-sample`
SET country = "Spain"
WHERE website="http://www.3fonteinen.be/index.htm";
```

Delete is like update, it is like SQL. The execution time is 690 milliseconds.

```
DELETE
FROM `beer-sample`
WHERE website="http://www.3fonteinen.be/index.htm";
```

d) Performance of Couchbase

Now, we are going to analyse how is the performance of Couchbase with data set of different size. We have “beer-sample” data set with 7.303 items, “gamesim-sample” data set with 586 items and “travel-sample” data set with 31.591 items.

All data set are different, but we are going to is similar queries for all of them, the query is going to select 3 values of each item and then filter.

“Gamesim-sample”

```
Select name, itemProbability, experienceWhenKilled
from `gamesim-sample`
where jsonType="monster" and itemProbability> 0.5
```

Number of items: 586.

Execution time: 85 milliseconds.

“Beer-sample”

```
Select name, category, brewery_id
from `beer-sample`
where type="beer"  and abv>4.9
```

Number of items: 7303.

Execution time: 679 milliseconds.

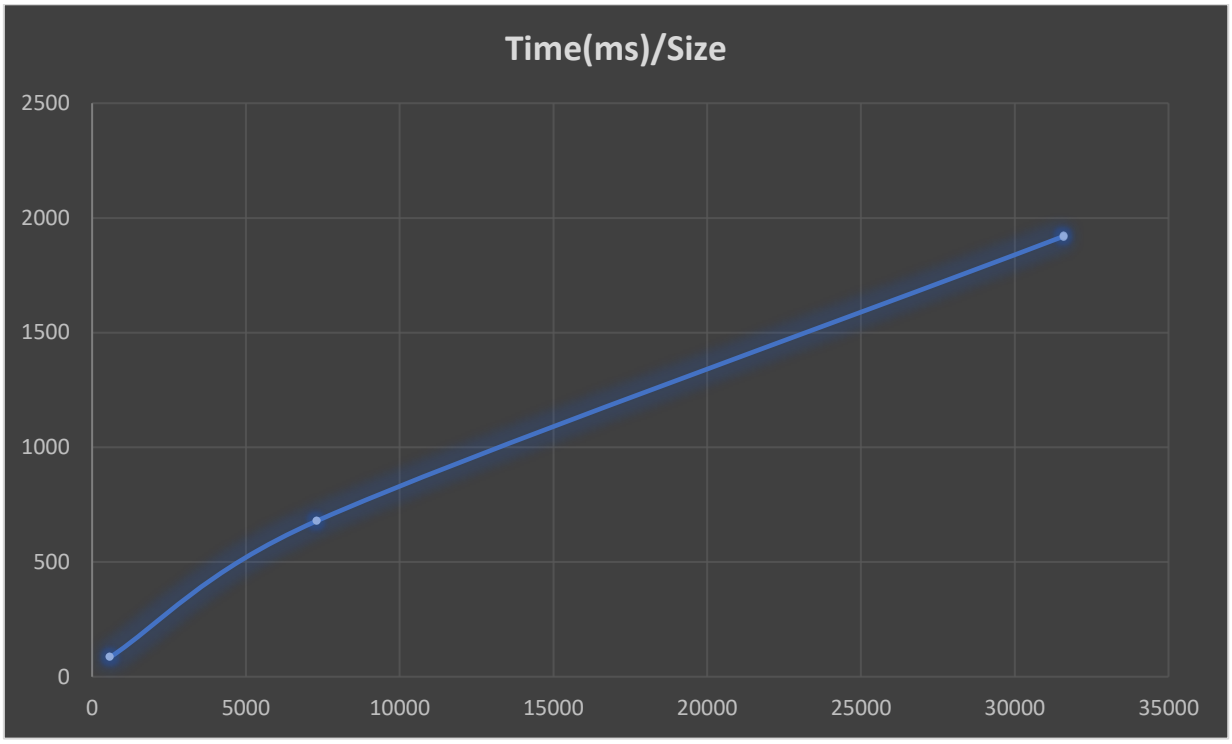
“Travel-sample”

```
Select airline, type, distance
from `travel-sample`
where type="route" and distance > 2881
```

Number of items: 31591.

Execution time: 1920 milliseconds.

In the following graph we can see the result. There is a linear correlation between the size and the execution time, so if you increment the size of the data set, the execution time is going to increment linearly.



Picture 7. Time/Size performance of Couchbase

4) Bibliography

- Couchbase Server. *Key-value or document database? Couchbase 2.0 bridges the gap.* Consulted on 5/11/2018, in https://blog.couchbase.com/key-value-or-document-database-couchbase-2-dot-0-bridges-gap/.
- DB-ENGINES. *Document Stores.* Consulted on 5/11/2018, in https://db-engines.com/en/article/Document+Stores.
- Basho. *Document Databases Explained.* Consulted on 25/11/2018, in http://basho.com/resources/document-databases/.
- IAN. *What is a Document Store Database?.* Consulted on 23/11/2018, in https://database.guide/what-is-a-document-store-database/.
- MongoDB. Document Databases. Consulted on 25/11/2018, in https://www.mongodb.com/document-databases.
- Couchbase Server. *N1QL Language Reference.* Consulted on 5/11/2018, in https://docs.couchbase.com/server/5.0/n1ql/n1ql-language-reference/index.html.
- Couchbase Server. *Distributed Data Management.* Consulted on 2/11/2018, in https://developer.couchbase.com/documentation/server/5.0/concepts/distributed-data-management.html.
- Couchbase Server. *Couchbase.* Consulted on 2/11/2018, in https://www.couchbase.com/.
- Couchbase Server. *Why Couchbase?*. Consulted on 5/11/2018, in https://docs.couchbase.com/server/6.0/introduction/intro.html.
- Couchbase Server. *Do a Quick Install*. Consulted on 5/11/2018, in https://docs.couchbase.com/server/6.0/getting-started/do-a-quick-install.html.
- Couchbase Server. *Run Your First N1QL Query*. Consulted on 5/11/2018, in https://docs.couchbase.com/server/5.0/getting-started/try-a-query.html.