DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico II

11 de noviembre de 2015

Bases de datos

Segundo Cuatrimestre de 2015

Integrante	LU	Correo electrónico
Ignacio Truffat	837/10	el_truffa@hotmail.com
Gaston Rocca	836/97	gastonrocca@gmail.com
Agustín Godnic	689/10	agustingodnic@gmail.com
Matías Pizzagalli	257/12	matipizza@gmail.com

Facultad de Ciencias Exactas y Naturales

Universidad de Buenos Aires

Ciudad Universitaria - (Pabellón I/Planta Baja)

Intendente Güiraldes 2160 - C1428EGA

Ciudad Autónoma de Buenos Aires - Rep. Argentina

Tel/Fax: (54 11) 4576-3359

http://www.fcen.uba.ar

Índice

1. Introdución.

El trabajo práctico esta enfocado en aprender el uso de las diferenctes funcionalidad disponibles en la base de datos MongoDB. Consiste en resolver distintos ejercicios utilizando esta base de datos No relacional. Dentro de las bases No relaciones, Mongo DB es una base orientada a Documentos. Esto quiere decir que en lugar de guardar los datos en registros, guarda los datos en documentos. Estos documentos son almacenados en un formato BSON, que es una representación binaria de JSON.

Una de las diferencias más importantes con respecto a las bases de datos relacionales, es que no es necesario seguir un esquema. Los documentos de una misma colección - concepto similar a una tabla de una base de datos relacional -, pueden tener esquemas diferentes.

En el trabajo vamos a experimentar y desarrollar ejercicios sobre Desnormalización, Map-Reduce y Sharding. También haremos una comparativa con otras bases de datos No Relaciones.

2. Desnormalización.

Pequeña intro al problema...

2.1. Empleados que atendieron clientes mayores de edad.

Embebimos los clientes dentro de los empleados, con la fecha de atención, obteniendo:

```
Empleado: {
  nroLegajo: int,
  nombre: string,
  clientes: [{DNI, Nombre, Edad, Fecha}],
  ...
}
```

Luego, para responder la consulta deseada hay que correr:

```
db.empleados.aggregate(
  [
    { $unwind: "$clientes" },
    { $match: {"clientes.Edad": { $gt: 17 } }  },
```

```
    { $group: { _id: "$nombre", nombre:{$first:"$nombre" } } },
    { $project : { _id:0, nombre: 1 } }
  ]
)
```

2.1.1. Ejemplo

Para insertar registros en la base corremos:

```
db.empleados.insert( {
        nroLegajo: 003,
        nombre: "Ernestino Juanes",
        clientes: [
                {DNI: 40528343,
                Nombre: "Raul Juan Lopez",
                Edad: 14,
                Fecha: "14/03/2015"} ] } )

db.empleados.insert( {
        nroLegajo: 002,
        nombre: "Juan Paez",
        clientes:
        [
                {
                DNI: 30154820,
                Nombre: "Juana Perez",
                Edad: 23,
                Fecha: "20/04/2015"
                },
                {
                DNI: 40528753,
                Nombre: "Raul Lopez",
                Edad: 15,
                Fecha: "23/04/2015"
                }
        ] } )

db.empleados.insert( {
        nroLegajo: 001,
        nombre: "Joaquina Paez",
```

```
        clientes: [ {
                DNI: 30154820,
                Nombre: "Juan Perez",
                Edad: 25,
                Fecha: "03/04/2015"} ] } )
```

Luego, una vez insertados los registros deseados, corremos la consulta mencionada arriba y nos da:

```
{ "nombre" : "Joaquina Paez" }
{ "nombre" : "Juan Paez" }
```

2.2. Artículos mas vendidos.

Embebimos los DNIs de clientes que compraron dentro de los artículos y buscamos los máximos.

```
Articulos: {
  CobBarras: int,
  nombre: string,
  compradores: [{DNI}]
}
```

Asumimos que hay un único articulo mas vendido:

```
db.articulos.aggregate(
  [
    { $unwind : "$Compradores"},
    {$group : { _id: "$CodBarras", CodBarras:{$first:"$CodBarras"},
      Nombre:{$first:"$Nombre"}, totalVendidos: {$sum: 1} } },
    {$sort: {totalVendidos: -1}},
    {$limit : 1},
    {$project : { _id:0,CodBarras: 1, totalVendidos: 1, Nombre: 1}}
  ]
)
```

2.2.1. Ejemplo

```
db.articulos.insert( { CodBarras: 7231564345110546, Nombre: "1984",
  Compradores: [22222222] } )
```

```
db.articulos.insert( { CodBarras: 0231564105110546,
  Nombre: "El principito", Compradores: [333333333, 22222222] } )
```

2.3. Sectores donde trabaja exactamente 3 empleado.

```
Sector: {
CodSector: int,
Empleados: [ {NroLegajo, idTarea}]
}
```

```
db.sectores.aggregate([{ $unwind : "$Empleados"},
        {$group : {_id: "$CodSector",CodSector:{$first:"$CodSector"},
        totalEmpleados: {$sum: 1}}},{$project : {_id: 0, CodSector: 1,
        totalEmpleados:1}},{$match : {totalEmpleados :3 }} ])
```

2.3.1. Ejemplo

```
db.sectores.insert({CodSector: 1, Empleados: [{NroLegajo: 001,
        idTarea: 02},{NroLegajo: 002, idTarea: 03},{NroLegajo: 003,
        idTarea: 01}]})

db.sectores.insert({CodSector: 2,
Empleados: [
        {NroLegajo: 001, idTarea: 01},
        {NroLegajo: 002, idTarea: 07},
        {NroLegajo: 003, idTarea: 01},
        {NroLegajo: 001, idTarea: 02},
        {NroLegajo: 008, idTarea: 12}
        ]})

db.sectores.insert({CodSector: 4,
Empleados: [
                {NroLegajo: 007, idTarea: 02},
                {NroLegajo: 003, idTarea: 11}]})
```

2.4. Empleado que trabaja en más sectores..

```
Empleado: {
nroLegajo: int,
nombre: string,
clientes: [{DNI, Nombre, Edad}]
trabajos: [{CodSector, idTarea}]
}
```

```
db.empleados.aggregate([{ $unwind : "$trabajos"},
{$group : {_id: "$nroLegajo",nroLegajo:{$first:"$nroLegajo"},
totalTrabajos: {$sum: 1}}},
{$project : {_id: 0, nroLegajo: 1, totalTrabajos:1}},
{$sort : {totalTrabajos: -1}},
{$limit : 1} ])
```

2.4.1. Ejemplo

```
db.empleados.insert( { nroLegajo: 006, nombre: "Ernestino Juanes",
        clientes: [
                {DNI: 40528343, Nombre: "Raul Juan Lopez", Edad: 14} ] ,
        trabajos: [{CodSector: 07, Tarea: 10}]} )

db.empleados.insert( { nroLegajo: 005, nombre: "Juan Paez",
        clientes: [
                {DNI: 30154820, Nombre: "Juana Perez", Edad: 23},
                {DNI: 40528753, Nombre: "Raul Lopez", Edad: 15} ],
        trabajos: [
                {CodSector: 01, Tarea: 02},
                {CodSector: 04, Tarea: 03},
                {CodSector: 07, Tarea: 09}] } )

db.empleados.insert( { nroLegajo: 004, nombre: "Joaquina Paez",
        clientes: [
                {DNI: 30154820, Nombre: "Juan Perez", Edad: 25} ],
        trabajos: [
                {CodSector: 09, Tarea: 01},
                {CodSector: 03, Tarea: 05}] } )
```

2.5. Ranking de los clientes con mayor cantidad de compras.

Asumo que se refiere a ordenarlos por la cantidad de compras que hizo cada uno, porque en el DER no dice nada de ranking ni votos ni nada.

```
Cliente: {
DNI: int,
nombre: string,
edad: int,
compras: [{CodBarra}]
}
```

```
db.clientes.aggregate([{ $unwind: "$compras"},
{$group : {_id: "$DNI",DNI:{$first:"$DNI"},
                nombre:{$first:"$nombre"},totalCompras: {$sum:1}} },
{$project :{ _id:0, DNI:1, nombre:1, totalCompras:1}},
{$sort : {totalCompras : -1}} ] )
```

2.5.1. Ejemplo

```
db.clientes.insert({
        DNI: 32012932,
        nombre: "Guillermo Rodriguez",
        edad: 23,
        compras: [
                {CodBarra: 321},
                {CodBarra: 023},
                {CodBarra: 231},
                {CodBarra: 123}
                ]})

db.clientes.insert({
        DNI: 33002654,
        nombre: "Pedro Juanes",
        edad: 28,
        compras: [
                {CodBarra: 023},
                {CodBarra: 231}
```

```
                ]})

db.clientes.insert({
        DNI: 38165687,
        nombre: "Carolina Hernandez",
        edad: 20,
        compras: [
                {CodBarra: 123}
                ]})
```

2.6. Cantidad de compras realizadas por clientes de misma edad.

```
db.clientes.aggregate([{ $unwind: "$compras"},
{$group : {_id: "$edad", edad: {$first:"$edad"},totalCompras:{$sum:1}}},
{$project: {_id: 0, edad: 1, totalCompras: 1}}])
```

2.6.1. Ejemplo

```
db.clientes.insert({
        DNI: 33002654,
        nombre: "Juan Martinez",
        edad: 28,
        compras: [
                {CodBarra: 007},
                {CodBarra: 109},
                {CodBarra: 182}]
                })
```

3. Map-Reduce.

Para la resolución de los Map-Reduce tuvimos que cargar diferentes archivos .json. Esto se realizo con las siguientes instrucciones:

```
mongoimport --db DB --collection COLLECTION --file
disposiciones_201*.json --jsonArray

Por ejemplo:
mongoimport --db tp2 --collection disposiciones --file
disposiciones_2014.json --jsonArray
```

Para cargar el código desde un .js se puede hacer load("codigo.js")

3.1. Disposiciones de tipo resolución en Abril de 2013.

Map: Si el registro dado es de tipo resolución y tiene fecha abril del 2013, Emitir("resolucion",cant=1)

Reduce: Sumar los cant q nos pasan y emitir lo mismo ("resolucion",suma)

```
var map1 = function(){
  var date = this["FechaBOJA"].split('/')
  if(this["Tipo"] == "Resoluciones" && date[1]==4 && date[2]==2013){
    emit(this["Tipo"],1)
  }
}
```

```
var reduce1 = function(key,values){
  return Array.sum(values)
}
```

Luego llamar a la función de map-reduce de la forma:

```
db.disposiciones.mapReduce(map1,reduce1,{out: parte2a})
```

Lo que nos devuelve:

```
{ "_id" : "Resoluciones", "value" : 607 }
```

3.2. Disposiciones de cada tipo.

Map: Emitir (tipo,cant=1)

```
var map2 = function(){
        emit(this["Tipo"],1)
}
```

Reduce: Sumar los cant que nos pasan y emitir lo mismo (tipo,suma)

```
var reduce2 = function(key,values){
        return Array.sum(values)
}
```

Luego correr:

```
db.disposiciones.mapReduce(map2,reduce2,{out: parte2b})
```

3.3. Fecha mas citada.

Map: Parsear la fechaBOJA y la fecha disposicion, matchearles el formato, luego: emitir(fechaBOJA,cant=1) emitir(fechaDisposicion,cant=1)

```
var map3 = function(){
        var date = (this["FechaDisposicion"].split('T'))[0].
                split('-').reverse().join('/');
        emit(date,1);
        emit(this["FechaBOJA"],1);
}
```

Reduce: Sumar los cant y emitir (fecha,cant)

```
var reduce3 = function(key,values){
        return Array.sum(values)
}
```

```
db.disposiciones.mapReduce(map3,reduce3,{out: parte2c})
```

Luego, de ese resultado nos quedamos con el máximo.

```
db.parte2c.find().sort({value : -1}).limit(1)
```

3.4. Mayor cantidad de páginas por cada tipo.

Map: emitir(tipo,cantPaginas)

```
var map4 = function(){
        var cantPags = this["PaginaFinal"] - this["PaginaInicial"] + 1
        emit(this["Tipo"],cantPags)
}
```

Reduce: Buscar el máximo entre todos los cantPaginas que nos llega y emitir (tipo,maxCantPaginas)

```
var reduce4 = function(key,values){
        var cantPagsMax = 0;
        for (var i=0; i < values.length; i++){
                if(values[i] > cantPagsMax){
                        cantPagsMax = values[i];
                }
        }
        return(cantPagsMax);
}
```

4. Sharding.

MongoDB utiliza esta técnica para gestionar la carga de los servidores. Distribuye los datos entre distintos shards (conjuntos de servidores que almacenan parte de los datos), para que la carga a la hora de realizar consultas e inserciones se reparta.

4.1. Pasos para la generación de resultados:

Levantamos cinco shards siguiendo las instrucciones del archivo tutorial_sharding.txt.

Luego creamos un indice simple sobre el atributo codigo_postal.

Luego importamos el código de insert_data.js donde tenemos funciones que nos permiten ingresar datos de a 20k y pedir las estadísticas. Finalmente, nos guardamos las estadísticas en archivos .txt. Los mismos se encuentran en la carpeta mediciones.

Generacion de Datos (insert_data.js)

```
var insertData = function(dbName, colName, num)
{
        var col = db.getSiblingDB(dbName).getCollection(colName);

        for (i = 0; i < num; i++)
        {
                x = Math.floor(Math.random() * 1000000);
                doc =
                {
                        nombre: 'Martin Juarez',
                        password: 'asdasd' ,
                        codigo_postal: x,
                        genero: 'masculino',
                        edad: 29,
                        fecha_creacion: '30/02/2015'
                }
                col.insert(doc);
        }

        print(col.count());
        print(col.getShardDistribution());
        print("################################################");
```

```
}

var insertDataTotal = function(dbName, colName, step, total)
{
        var col = db.getSiblingDB(dbName).getCollection(colName);

        while(col.count() < total)
        {
                for (i = 0; i < step; i++)
                {
                        x = Math.floor(Math.random() * 1000000);
                        doc = {
                                nombre: 'Martin Juarez',
                                password: 'asdasd' ,
                                codigo_postal: x,
                                genero: 'masculino',
                                edad: 29,
                                fecha_creacion: '30/02/2015'
                        }
                        col.insert(doc);
                }

                print(col.count());
                print(col.getShardDistribution());
                print("########################################");
        }
}
```

4.2. Resultados

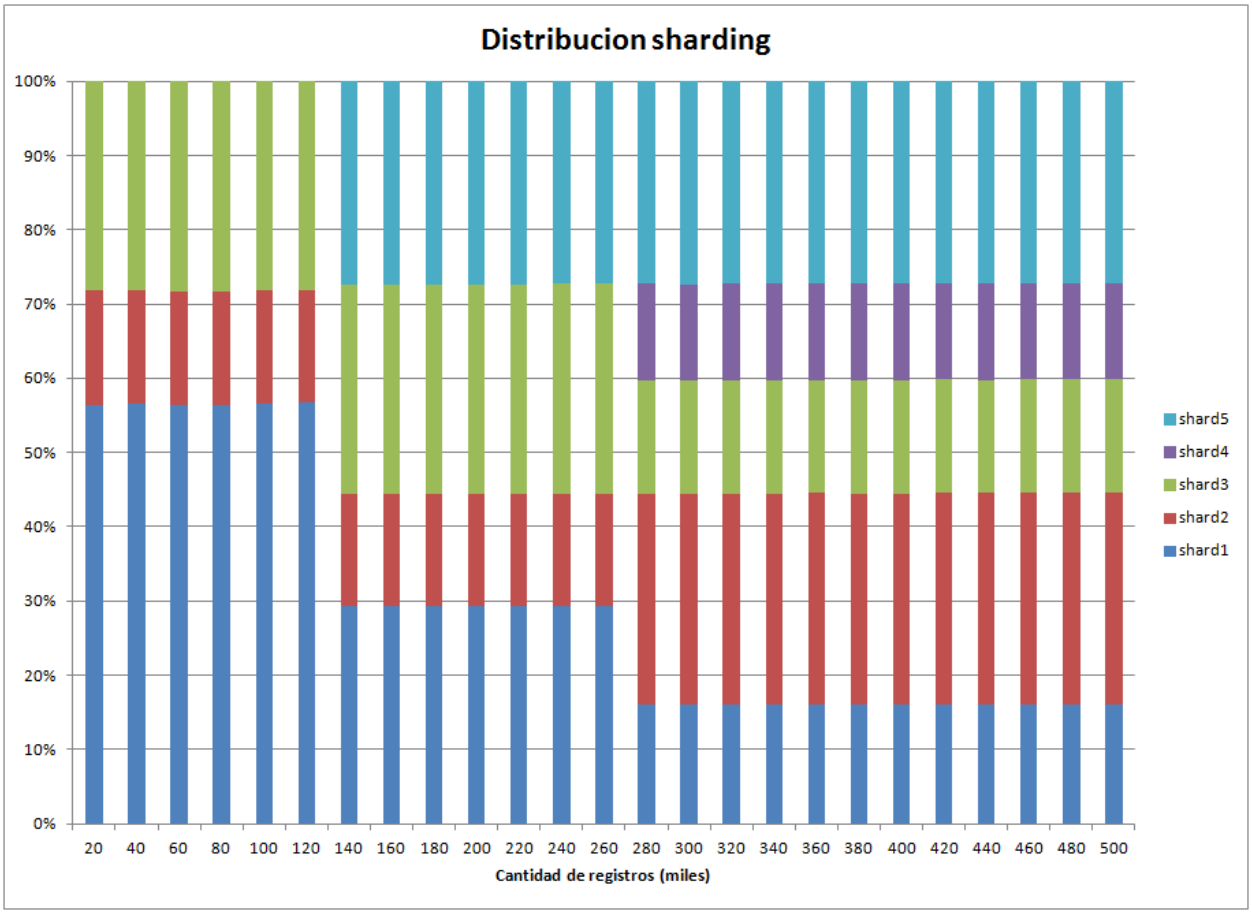

Figura 1: Porcentaje de distribucion entre los shards usando un indice simple en base al codigo postal.

El primer resultado obtenido corresponde a la distribución de carga con el indice simple sobre código postal. En este caso la distribución no es uniforme, al comienzo gran parte de la carga se la lleva el Shard1 y no intervienen los Shard4 y Shard5. Esto es asi, porque se van llenando los primeros Shards, cuando esto ocurre luego de insertar unos 140.000 registros, vemos que comienzan a distribuirse en el Shard5, y el peso del Shard1 baja a un 30 %. Al insertar unos 280.000 registros, y de ahí en adelante, el Shard4 comienza a registrar carga. Y en general, todos los shards tienen documentos.

Ahora veremos como influye la generación de un indice Hash en base al _id.

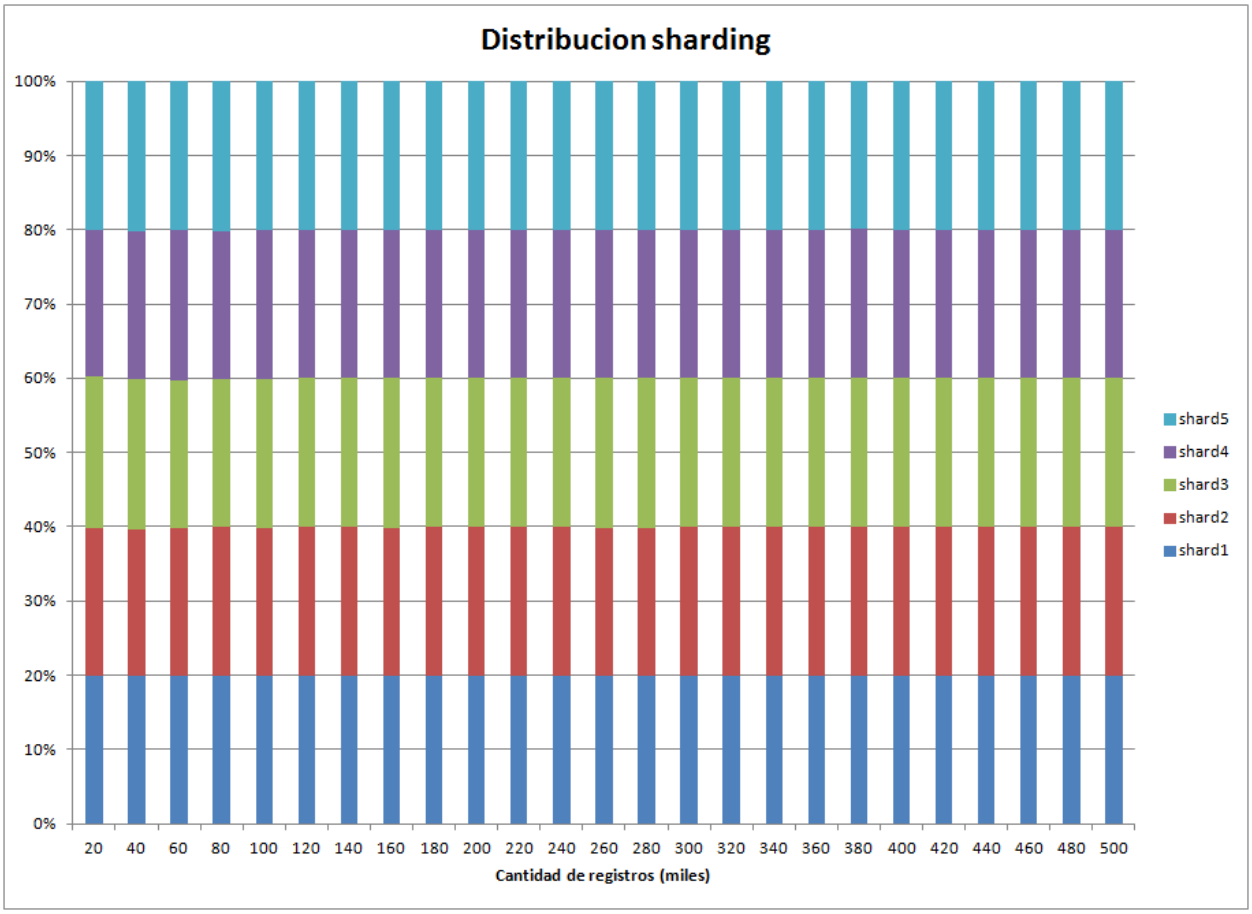

Figura 2: Porcentaje de distribución entre los shards usando un indice hasheado en base al _id

En este caso, el indice hash hace que la distribución de los documentos sea muy equitativa entre los distintos Shards. Desde el comienzo de la carga de registros, se puede ver que todos los Shards tienen aproximadamente un 20 % de carga.

Esto nos hace reflexionar sobre la importancia de la elección de los indices y su influencia en la distribución de la carga.

5. Otras base de datos NoSql.

Base de datos key-value, usando el motor redis. Redis permite el uso de namespaces (tener varios "diccionarios", la terminología de redis para esto sería "multiple databases"), lo cual hace el diseño más prolijo y simple.

Usar map-reduce en redis no es algo built-in ni estandar, pero investigamos y es posible, por ejemplo, integrarlo con Hadoop.

Parte1: 1a: En redis hay un comando SCAN que permite iterar las claves. Con esto podemos iterar un diccionario de empleados, donde en cada value hay una lista de datos de cada cliente que atendió. Entonces se puede iterar las claves una por una y quedarse con las que tienen algun cliente mayor de edad. 1b: Usando SCAN se puede iterar las claves de un diccionario articulos -> ventas. Mediante codigo se buscan las claves que tengan |ventas| maximo.

1c: Idem usando SCAN. Si tenemos un diccionario sector -> [empleados], es cuestion de iterar y mediante código buscar cuando |empleados|==3

1d: Un diccionario empleado -> [sectores]. Idem 1c usando SCAN.

1e: Diccionario cliente -> [compras]. Idem 1c, pero ordenando mediante |compras|.

1f: Esta consulta ya es un poco más compleja, hay varios enfoques posibles. Uno es mantener las cosas simples y directamente usar un diccionario edad -> cantidadDeCompras. Por ser tan específico, es un diccionario que sólo sirve para esta consulta, con lo cual estamos agregando un costo de mantenimiento extra a la DB sólo por una query. Otra opción es valerse de map reduce y usar alguno de los otros diccionarios, como el de 1a. Como ventaja, la consulta es simple y no hace falta crear un diccionario extra solo por esta consulta.

Parte 2: Asumimos que se crea un id único para las disposiciones, podría ser un hash de sus datos o la combinación (numeroBoja, paginaInicial, PaginaFinal) que asumimos que identifica univocamente a la disposicion. Entonces se tiene un diccionario id -> disposicion.

Las resoluciones por map-reduce son prácticamente iguales a las versiones hechas en mongodb ya que la entrada de la función map es practicamente la misma.

Parte 3: TODO: primero habría que hacerlo en mongodb jaja.

6. Conclusiones.

El sharding es una herramienta muy útil para balancear la carga de datos entre servidores. MongoDB nos proporciona una forma sencilla de hacerlo, pero que hay que configurar correctamente. La elección de la clave por la que se realizará el sharding (shard key) es muy importante. Esta elección no se puede cambiar una vez se ha establecido.