5. Base de datos NoSQL: MongoDB

A pesar de que las bases de datos relacionales han satisfecho las necesidades de la inmensa mayoría de usuarios en las últimas décadas, no podríamos decir que son infalibles en todas las soluciones, y si bien gracias a la gran especialización de personal con alta formación se pueden conseguir adaptaciones que terminen dando un resultado aceptable, tal vez sea mejor partir de cero con nuevas ideas que puedan hacer frente a nuevos desafíos.

Si bien la cuota de mercado de estas soluciones NoSQL no está a la altura de las relacionales, las mayores empresas de Internet ya tienen sus propios sistemas basados en esta tecnología, entre los que caben destacar: Google con su BigTable, Amazon con su DinamoDB o Linkedin con RedIs.

El contenido de este capítulo que se inicia a continuación girará en torno a la arquitectura de MongoDB. Se profundizará en la promesa de MongoDB de una eficaz escalabilidad y se presentarán algunos ejemplos de cómo trabaja la base de datos con la información, insertando, seleccionando o actualizando datos.

5.1 Introducción

MongoDB es un sistema de base de datos NoSQL multiplataforma de licencia libre.

Está orientado a documentos de esquema libre, lo que implica que cada registro puede tener un esquema de datos distinto, (los atributos no tiene que repetirse entre los diferentes registros).

Es una solución pensada para mejorar la escalabilidad horizontal de la capa de datos, con un desarrollo más sencillo y la posibilidad de almacenar datos con órdenes de magnitud mayores. Cuando se necesita escalar con muchas máquinas, el enfoque no-relacional resulta eficiente, y MongoDB una alternativa correcta. El modelo de documento de datos (JSON/BSON) pretende ser fácil de programar, fácil de manejar y ofrece alto rendimiento mediante la agrupación de los datos relevantes entre sí, internamente.

En MongoDB, cada registro o conjunto de datos se denomina documento, que pueden ser agrupados en colecciones, (equivalente a las tablas de las bases de datos relacionales pero sin estar sometidos a un esquema fijo). Se pueden crear índices para algunos atributos de los documentos.

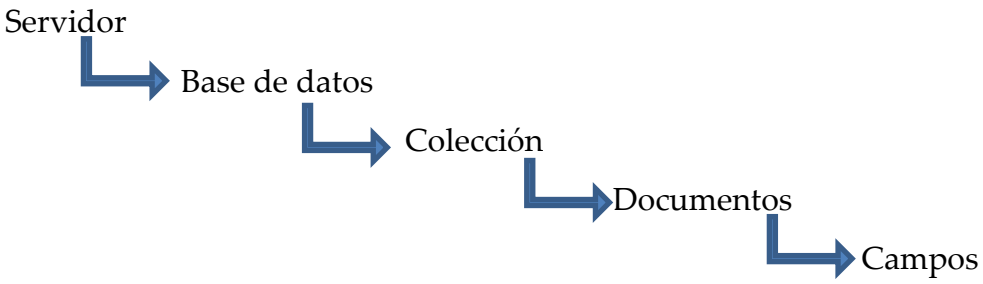

Figura 15: Arquitectura funcional MongoDB

La jerarquía principal de MongoDB viene dada por los elementos presentados arriba. La diferencia con los SGBDR, (Sistmas Gestores de Bases de Datos Relacionales), es que no utilizamos tablas, filas ni columnas como allí lo hacíamos, sino que utilizamos documentos con distintas estructuras. Un conjunto de Campos formarán un Documento, que en caso de asociarse con otros formará una Colección. Las bases de datos estará formada por Colecciones, y a su vez, cada servidor puede tener tantas bases de datos como el equipo lo permita.

En MongoDB, como ya se ha mencionado, no existe un esquema estándar para trabajar con los datos, pero eso no significa que vayamos a tener una cantidad ingente de datos incorrelados difíciles de relacionar. De hecho, la mayoría de las veces trabajaremos con documentos estructurados, sólo que no seguirán el mismo esquema todos ellos, sino que cada uno podrá tener uno propio si así resulta apropiado trabajar con ellos.

Para el intercambio de datos para almacenamiento y transferencia de documentos en MongoDB usamos el formato BSON, (Binary JavaScript Object Notation). Se trata de una representación binaria de estructuras de datos y mapas, diseñada para ser más ligero y eficiente que JSON, (JavaScript Object Notation).

MongoDB cuenta con una serie de herramientas que permiten trabajar con la base de datos desde diferentes perspectivas, y tratar con ella para diferentes propósitos, destaco entre ellas:

i. ***Mongod***: Servidor de bases de datos de MongoDB
ii. ***Mongo***: Cliente para la interacción con la base de datos MongoDB
iii. ***Mongofiles***: Herramienta para trabajar con ficheros directamente sobre la base de datos MongoDB

Existen más herramientas que puede consultar en [7]

Las bases de datos NoSQL hacen gala de una alta escalabilidad horizontal, que en resumidas cuentas supone la capacidad de adaptación de un sistema al aumento de información con los que trabajar. MongoDB presenta una arquitectura que facilita la ejecución de esta importante tarea, y presenta un método para dividir los datos entre los múltiples servidores que pudiera presentar una solución escalada: el Sharding. El principal punto fuerte de este método es que se ejecuta de forma automática, permitiendo siempre una ágil distribución de datos entre todos los equipos que se encuentren trabajando. Siendo ésta una característica importante de MongoDB se profundizará en el apartado 5.3.1

5.2 Replicación

Para dotar de consistencia a nuestro sistema, y particularmente a una base de datos NoSQL como MongoDB, resulta muy aconsejable el uso de la replicación, y la propuesta estudiada aporta buenas soluciones a tal cuestión.

MongoDB, es más flexible que las bases de datos relacionales, y por ello menos restrictivo, lo que puede presentar en ocasiones problemas de volatilidad. Esto es del todo indeseable en las bases de datos.

MongoDB manda los documentos escritos a un servidor maestro, que sincronizado a otro u otros servidores mandará esta misma información replicada, a estos "esclavos". Hay muchas opciones para trabajar con esta utilidad, por lo que podremos escribir todo o parte según nuestras necesidades. De especial importancia es el hecho de que si el servidor maestro cae, uno de los esclavos puede ocupar su lugar y permitir así que el servicio continúe.

5.3 Escalabilidad horizontal

Las bases de datos relacionales comerciales con capacidad distribuida suelen tener unas licencias realmente costosas, por lo que resulta interesante buscar alternativas eficientes y de bajo coste para nuestro sistema.

La escalabilidad horizontal supone trabajar con varias máquinas de manera distribuida, almacenando en cada uno de los nodos cierta información que de una forma u otra debe estar comunicada con el resto de nodos que forman nuestro sistema. Esto dota de mayor flexibilidad al sistema, ya que facilita la agregación de equipos en función de las necesidades.

Cosas a tener en cuenta para adoptar una solución de manejo de grandes volúmenes:

1. Limites potenciales de la escalabilidad

2. Tolerancia a fallos

3. Facilidad de administración

4. Facilidad de implementación

5. Expectativa de escalabilidad de rendimiento

Cuando se trabaja con una base de datos en un solo equipo, el número de estados posibles se reduce a Up, o Down, pero cuando se tiene un sistema de bases de datos distribuido, el número de opciones aumenta, en tanto en cuanto existen dependencias entre los nodos que llevará a trabajar con un mayor número de opciones; puede ocurrir que uno de los servidores caiga, o que no haya comunicación entre algunos nodos, o que la comunicación sea muy lenta.

El hecho de meter mayor número de nodos conlleva un aumento de la complejidad de la solución; por suerte, las soluciones NoSQL han sido pensadas para reducir al mínimo este número de inconvenientes, y hacer que la tarea de distribuir las bases de datos no sea excesivamente ardua. En este contexto, MongoDB resulta ser una excelente elección, que por el hecho de ser open source ya supone un ahorro considerable en cuanto a coste de herramientas comerciales para la distribución de bases de datos.

Una vez se ha llegado a la conclusión de que necesitamos un clúster, hemos de meternos en materia para conocer cómo funciona con MongoDB y cómo llevarlo a cabo.

En primer lugar, decir que un clúster es un conjunto de servidores distribuidos para un propósito común, en nuestro caso, una base de datos.

5.3.1 Sharding

MongoDB utiliza el Sharding como método para dividir los datos a lo largo de los múltiples servidores de nuestra solución. Las bases de datos relacionales también hacen tareas similares a ésta, si bien de forma diferente. Tal vez el rasgo más destacable en MongoDB pasa porque realiza estas tareas de manera automática.

A continuación se hablará de clúster habitualmente, para lo que habría que indicar que se trata de un conjunto de equipos que trabajan conjuntamente ofreciendo una imagen de equipo único al exterior.

Una vez se ha indicado a MongoDB que se va a distribuir la base de datos, añadiendo un nuevo nodo al clúster, éste se encarga de balancear los datos, (distribuir los datos de forma eficiente entre todos los equipos), entre los servidores de forma automática. Los objetivos del Sharding en MongoDB pasan por:

1. Un clúster transparente: Nos interesa que a la hora de presentar el trabajo sea indiferente si se trabaja con uno o varios servidores. Para satisfacer esto, MongoDB hace uso de un proceso de enrutamiento, al que llaman *mongos*. Los mongos se colocan frente al clúster, y de cara a cualquier aplicación, parece como si estuviera delante de un servidor individual MongoDB. Mongos reenvía la consulta al servidor o servidores correctos, y devuelve al cliente la respuesta que éste ha proporcionado.

2. Disponibilidad para lecturas y escrituras: Siempre que no existan causas de fuerza mayor, (como caída masiva de la luz), debemos asegurar la disponibilidad de escritura y lecturas en nuestra base de datos. Antes de que se degrade excesivamente la funcionalidad del sistema, el clúster debe permitir fallar tantos nodos como sean necesarios. MongoDB hace que algunos de los procesos del clúster tengan redundancia en otros, de forma que si cae un nodo, otro puede continuar con estos procesos.

3. Crecimiento ágil: Nuestro clúster debe añadir o reducir capacidad cuando lo necesite.

Un shard es uno o varios servidores de un clúster que son responsables de un subconjunto de datos del mismo, (un clúster con 1.000.000 de documentos, un shard por ejemplo con 200.000). En el caso de que el shard esté compuesto por más de un servidor, cada uno de estos tendrá una copia idéntica del subconjunto de datos.

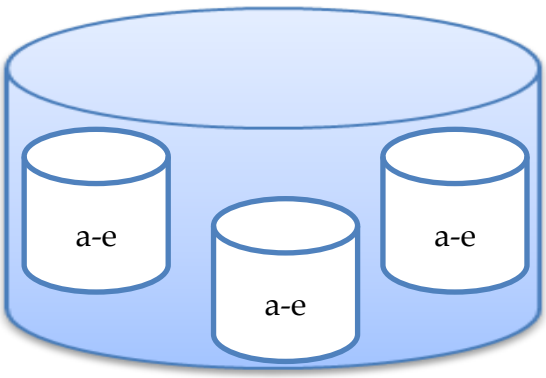

Figura 16: Shard simple

Para distribuir uniformemente los datos, MongoDB mueve los subconjuntos de shard en shard en base a una clave que debemos elegir.

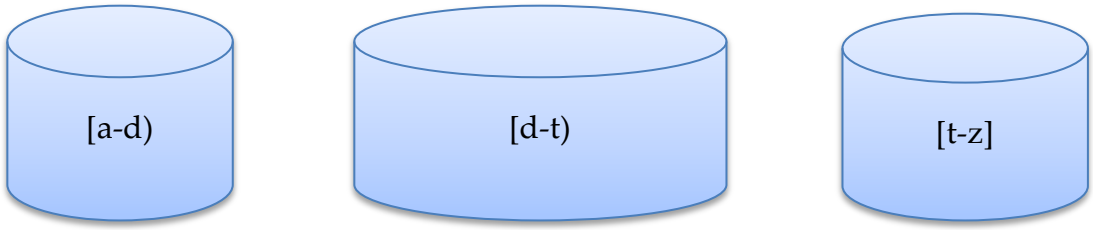

Figura 17: Distribución de los Shards en función de la letra de inicio

A continuación se presenta un ejemplo simple, basado en los presentados en [5]. En este ejemplo se van a definir una serie de nombres de personas, a los que se les va a asociar una edad determinada, para posteriormente ver cómo trabajaría el Sharding con ellos.

Sea la colección:

{"nombre" : "Juan", "edad" : 25} {"nombre" : "Paula", "edad" : 17} {"nombre" : "Tomás", "edad" : 30}

{"nombre" : "Pepe", "edad" : 75} {"nombre" : "Paqui", "edad" : 42}

Si elegimos un rango con clave edad y rango [18,42), la selección sería:

{"nombre" : "Juan", "edad" : 25}

{"nombre" : "Tomás", "edad" : 30}

Esta es la forma más fácil de trabajar, pero puede ocasionar problemas si por ejemplo, en el segundo servidor, se registran muchos nombres, haciendo que el tamaño del mismo crezca desorbitadamente. Para ajustar manual, o automáticamente estas situaciones tendríamos que mover una cantidad de datos enorme, nada eficiente.

Una solución mejor implementada es la de usar múltiples rangos dentro de los Shards, lo que lleva aparejada un mejor rendimiento, en tanto en cuanto la flexibilidad que tenemos va a permitir mover un menor número de datos entre servidores para el balanceo, y esto a su vez, mayores probabilidades de que no baje el rendimiento de nuestro sistema. MongoDB trabaja muy bien con este sistema, de forma que si un subconjunto de datos se hace muy grande, MongoDB lo divide en dos pedazos más pequeños de forma automática.

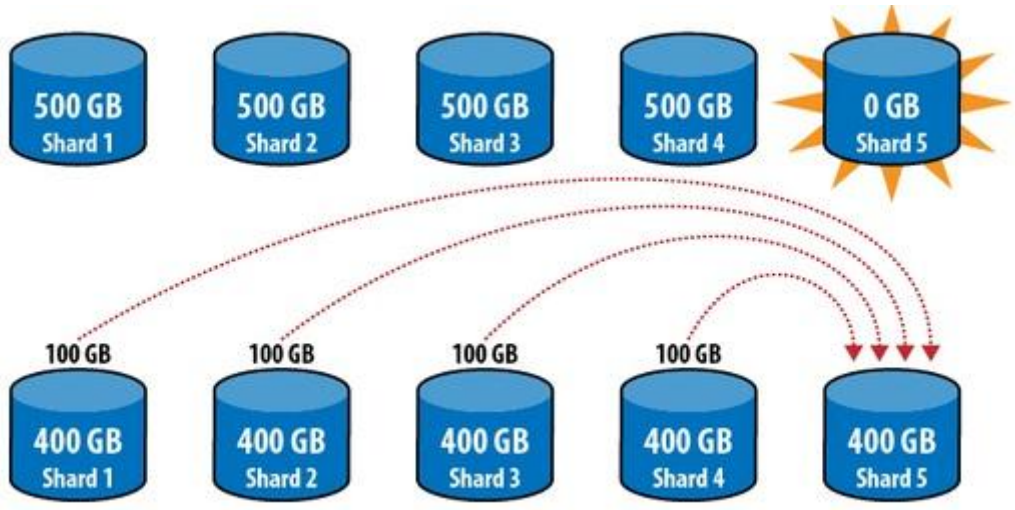

Figura 18: Sharding complejo en MongoDB [5]

Desde el primer subconjunto de datos que tenemos en la colección de documentos, MongoDB le asigna una porción, que en un primer momento será todos los datos que haya. Si el número de datos crece, puede llegar el momento en que MongoDB divida estos datos, creando nuevos subconjuntos, tantos como sea necesario, y de una forma balanceada.

Es importante resaltar que los subconjuntos no pueden estar solapados.

Una vez que tenemos un subconjunto, hay que tener en cuenta que no podemos cambiar la clave shard a un documento; si queremos hacerlo, debemos eliminar el documento, cambiarle la clave shard en el lado cliente, y reinsertarlo.

En la división de subconjuntos se pueden mezclar diferentes tipos de campos, MongoDB tratará con ellos de forma ordenada en un proceso interno.

Importante: los subconjuntos por defecto tienen 64MB de tamaño máximo. Mover grandes cantidades puede ser contraproducente, en cuanto la bajada de rendimiento sería muy notable.

5.3.2 Balanceo

El balanceador es un proceso de MongoDB para equilibrar los datos en nuestro sistema. Mueve porciones de datos de un shard a otro, de manera automática, lo que supone una ventaja apreciable. Si el administrador no quisiera que esto se llevara a cabo, habría que deshabilitar dicha opción.

Para que existan particiones debemos crear datos de al menos 2GB, aunque podemos modificar esto con el comando, chunksize N, donde N es el tamaño en MB

5.3.3 El Clúster

Un clúster MongoDB se basa de forma general en tres procesos:

1. Sharding: para el almacenamiento distribuido de datos

2. Mongos: para enrutar las peticiones hacia el dato correcto

3. Configuración de servidor: Para hacer el seguimiento del estado del clúster

En la siguiente imagen se muestran a modo de módulos los componentes de un clúster en MongoDB

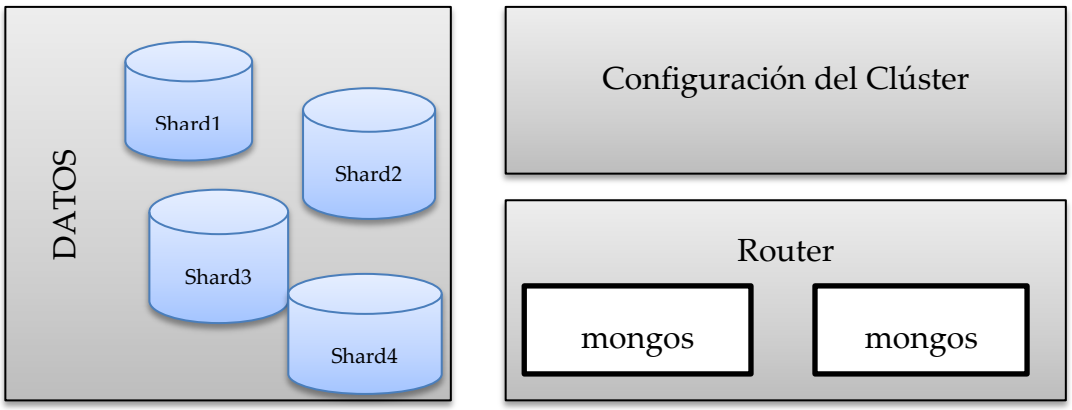

Figura 19: Arquitectura de Clúster MongoDB

Como se ha mencionado con anterioridad, los mongos son el punto de encuentro entre el usuario y el clúster. Su principal misión consiste en hacer transparente el trabajo del clúster, de forma que permita que el usuario piense que se encuentra ante un único servidor.

Cuando se usa un clúster, nos conectaremos a un mongo y le enviaremos todas las operaciones de escritura/lectura.

Cuando un usuario realiza una petición, mongos se encarga de redirigirla al shard correcto, y dentro de este, a la división adecuada.

5.4 Lista de comandos

Hay una serie de comandos que podemos ejecutar contra la base de datos sobre la que trabajamos, para acceder a ellos hemos de entrar en la herramienta ***mongo*** y posteriormente llamar al objeto *db.help()*

```
> db.help()
DB methods:
        db.addUser(username, password[, readOnly=false])
        db.auth(username, password)
        db.cloneDatabase(fromhost)
        db.commandHelp(name) returns the help for the command
        db.copyDatabase(fromdb, todb, fromhost)
        db.createCollection(name, { size : ..., capped : ..., max : ... } )
        db.currentOp() displays the current operation in the db
        db.dropDatabase()
        db.eval(func, args) run code server-side
        db.getCollection(cname) same as db['cname'] or db.cname
        db.getCollectionNames()
        db.getLastError() - just returns the err msg string
        db.getLastErrorObj() - return full status object
        db.getMongo() get the server connection object
        db.getMongo().setSlaveOk() allow this connection to read from the nonmas
ter member of a replica pair
        db.getName()
        db.getPrevError()
        db.getProfilingLevel() - deprecated
        db.getProfilingStatus() - returns if profiling is on and slow threshold

        db.getReplicationInfo()
        db.getSiblingDB(name) get the db at the same server as this one
        db.isMaster() check replica primary status
        db.killOp(opid) kills the current operation in the db
        db.listCommands() lists all the db commands
        db.logout()
        db.printCollectionStats()
        db.printReplicationInfo()
        db.printSlaveReplicationInfo()
        db.printShardingStatus()
        db.removeUser(username)
        db.repairDatabase()
        db.resetError()
        db.runCommand(cmdObj) run a database command.  if cmdObj is a string, tu
rns it into { cmdObj : 1 }
        db.serverStatus()
        db.setProfilingLevel(level,<slowms>) 0=off 1=slow 2=all
        db.shutdownServer()
        db.stats()
        db.version() current version of the server
        db.getMongo().setSlaveOk() allow queries on a replication slave server
        db.fsyncLock() flush data to disk and lock server for backups
        db.fsyncUnock() unlocks server following a db.fsyncLock()
```

Figura 20: Lista de comandos ejecutables contra la base de datos

Se introduce seguidamente un nuevo ejemplo, que va a servir de guía para insertar datos y seleccionarlos con posterioridad. En el ejemplo se asociará el nombre de una asignatura con diferentes campos, como pueden ser el nombre del profesor que la imparte, el año del que se trata, o cualquier otro campo que se quiera implementar, lo cual es posible gracias a la arquitectura libre de esquemas que presenta MongoDB.

5.4.1 Inserciones básicas

Se puede comprobar a continuación que en una misma colección se pueden insertar diferentes documentos con distintos esquemas:

>db.asignaturas.insert({nombre: 'Química', profesor: 'Juan', año: '2012'})

>db.asignaturas.insert({nombre: 'Física', profesor: 'Juan', colegio: 'Freelance', edad: '36'})

Siempre que se quiera trabajar con los datos de una base de datos debe seguir una nomenclatura definida. Primeramente aparecerán las letras *db*

seguidas de un punto, y el nombre de la colección sobre la que se quiere trabajar, en este caso, *asignaturas*. Posteriormente debe indicarse el comando que se quiera ejecutar, ya sea *find()*, *insert()*, …

Para recuperar estos datos utilizamos *db.asignaturas.find()*, que devuelve:

```
> db.asignaturas.find()
{ "_id" : ObjectId("4f467942f4eddb2b25f2eb10"), "nombre" : "Quimica", "profesor"
 : "Juan", "year" : "2012" }
{ "_id" : ObjectId("4f467972f4eddb2b25f2eb11"), "nombre" : "Fisica", "profesor"
: "Juan", "colegio" : "Freelance", "edad" : "36" }
>
```

Figura 21: Recuperación de documentos

Todo documento debe tener un campo *_id*, que puede ser asignado por nosotros, o por la propia máquina con *ObjectId*. Por defecto, este campo será indexado en la colección.

5.4.2 Consultas

Un aspecto de suma importancia en las bases de datos son las consultas que se realizan sobre los datos de la misma. En MongoDB es especialmente importante por la naturaleza desesquematizada de sus datos. Aquí se va a hablar de selectores de consulta.

>db.asignaturas.find({nombre: 'Física' $and: [{profesor: 'Juan'}]})

Lo anterior devolverá todos los documentos que tenga como valores de nombre: física, y como profesor: Juan.

Se presenta un nuevo ejemplo a continuación para profundizar en el área de las consultas. En este ejemplo se presenta una colección, *prendas*, que ha sido almacenada en la base de datos previamente, y que contiene documentos con campos que hacen referencia al tipo de prenda, su talla, el color, o el fabricante.

En algún caso particular podemos preferir obtener sólo el valor de un campo en nuestra consulta, para ello MongoDB establece la forma de realizarlo, que es usando un segundo parámetro que se lo indica a la base de datos:

>db.prendas.find({color= 'Rosa palo'}, {nombre: 1});

>db.prendas.find({color= 'Rosa palo'}, {fabricante: 0});

En el primer caso, devolvemos como resultado el campo nombre de aquellos documentos que cuenten con color rosa palo. En la segunda opción, el resultado mostrará todos los campos de los documentos con color rosa palo, a

excepción del campo fabricante, que no será mostrado. Cabe resaltar que aunque no lo queramos, el campo _id siempre será devuelto.

De interés también resulta la posibilidad de ordenar en función de nuestras necesidades los resultados de nuestras consultas, ayudándonos del comando Sort:

```
>db.prendas.find().sort({cantidad: -1});   ///orden descendente

>db.prendas.find().sort({cantidad: 1});   ///orden ascendente
```

También nos puede interesar contar el número de documentos que cumplen con una determinada condición:

```
>db.prendas.find({fabricante: 'Kuretan'}).count();
```

Este último comando nos devuelve el número de prendas cuyo fabricante es Kuretan.

5.4.3 Actualizaciones

La forma más sencilla de actualización es a través del comando *Update*, que toma dos argumentos, uno de los cuales será el selector que indicará dónde hay que modificar, y el otro será el campo que hay que modificar, con el valor concreto.

En MongoDB la interacción es diferente a las de bases de datos SQL tradicionales. Un ejemplo de estas diferencias se presenta al utilizar la actualización de campos; usando el comando Update sin más, estamos remplazando el documento original, lo que nos va a llevar errores en caso de realizar búsquedas específicas. Para actualizar simplemente uno, o varios campos, hemos de hacer uso de la llamada a $Set.

Para el ejemplo ya introducido de la colección *asignaturas*, haríamos lo siguiente:

```
>db.asignaturas.update({nombre: 'Física'}, {$set: {profesor: 'Juan'}})
```

Aparte del modificador ya estudiado: $set, existen otros que pueden resultar de interés:

- $inc: incrementa o decrementa un campo en función del valor indicado
- $unset: elimina un campo añadido

- $push: añade un valor al campo, trabajando con arrays
- $addtoSet: añade valores a un array
- $pop: elimina el último elemento de un array
- $pull: elimina todas las concurrencias que coincidan con el valor indicado para el campo requerido
- $rename: renombrado de un campo
- $bit: actualización bit a bit de un campo

Otro de los comandos útiles para actualizaciones es Upsert, que actualiza un documento si ya existía, o lo crea en caso contrario. Además, otra utilidad muy interesante es la posibilidad de realizar actualizaciones múltiples de los documentos de interés, y como ejemplo presentamos una posible solución para el ejemplo de la colección *prendas*:

```
>db.prendas.update({}, {$set: {TallaLdisponible: 'true'}}, false, true);
```

Los últimos dos parámetros son de control, para poder realizar correctamente la actualización múltiple.

```
>db.prendas.find({TallaLdisponible: 'true'})
```

En la primera línea hemos establecido que todas las prendas de nuestra base de datos tienen disponible la talla L, mientras que en el segundo hacemos una consulta que nos muestre cuáles son las prendas con talla L disponibles; como hemos actualizado todas las referencias en una actualización múltiple, el resultado será, evidentemente, una respuesta con todas las referencias posibles.

5.5 MapReduce

MapReduce es una utilidad para el procesamiento de datos por lotes y operaciones de agregación. Está escrita en JavaScript y se ejecuta en servidor. Permite utilizar funciones de agregación de datos, que de forma original sería complicado de tratar debido a la ausencia de esquemas típicos de estas soluciones.

Una de los beneficios que puede ofrecer MapReduce es el aumento de rendimiento; se puede trabajar en paralelo, permitiendo así tratar con gran cantidad de datos sobre distintas máquinas.

El proceso consta de dos partes, por un lado el mapeo, y por otro la reducción. El mapeo transforma los documentos y emite un par (clave/valor). La función reduce obtiene una clave y el array de valores emitidos para esa clave y produce el resultado final.

Lo bueno de este enfoque analítico es que almacenando las salidas, los informes son más rápidos de generar y el crecimiento de datos es controlado.

Se presenta un nuevo ejemplo, que contiene la base de datos de una tienda de venta de vinos. En esta base de datos interesa saber, entre otros, el nombre del vino, su referencia, precio y fecha de adquisición.

Los valores emitidos se agrupan juntos, como arrays, con la llave como referencia común.

>db.tienda.insert({ítem: vino Mio, referencia: 1002, precio: 2.30, date: new Date(2012, 1, 2) });

>db.tienda.insert({ítem: vino Mio crianza, referencia: 1003, precio: 9.50, date: new Date(2012, 1, 2) });

>db.tienda.insert({ítem: vino Salvador, referencia: 0058, precio: 18.95, date: new Date(2012, 1, 2) });

>db.tienda.insert({ítem: vino Mio, referencia: 1002, precio: 2.30, date: new Date(2012, 1, 2) });

>db.tienda.insert({ítem: vino Torres, referencia: 3105, precio: 4.65, date: new Date(2012, 1, 2) });

>db.tienda.insert({ítem: vino Mio, referencia: 1002, precio: 2.30, date: new Date(2012, 1, 5) });

>db.tienda.insert({ítem: vino Torres, referencia: 1002, precio: 4.65, date: new Date(2012, 1, 5)});

Con lo anterior se han insertado todos estos documentos en la colección *tienda*. Un código de ejemplo para uso de MapReduce con la colección arriba definida sería:

```
var map = function() {
var key = {item: this.item, referencia: this.referencia, precio: this.precio year:
this.date.getFullYear(), month: this.date.getMonth(), day: this.date.getDate()};
emit(key, {count: 1});
};

var reduce = function(key, values) {
var sum = 0;
values.forEach(function(value) {
sum += value['count'];});

return {count: sum};
};

db.hits.mapReduce(map, reduce, {out: {inline:1}});
```

En el código mostrado, el objeto *this* hace referencia al documento actual con el que estamos trabajando, el nombre que le sigue muestra el campo con el que se quiere trabajar. La intención del código es agrupar las referencias con

valores comunes, para que aparezcan por pantalla el número de registros totales presentes en la base de datos que coinciden en todos sus campos.

El tercer parámetro de la función *MapReduce* puede variar en función de nuestras necesidades; para este caso concreto, nos devolverá en pantalla inmediatamente el resultado.

5.6 Modelado de datos

De forma general, las operaciones con Join, (operación de manejo de datos que implica tablas relacionadas entre sí), típicas de las bases de datos relacionales pueden resultar no muy escalables, y puesto que éste es uno de los principales propósitos de MongoDB, parece lógico no trabajar exactamente con Join.

A pesar de lo anterior, se debe buscar una forma de trabajar que nos proporcione una funcionalidad parecida a los Joins, en tanto en cuanto necesitamos hacer consultas con cierta complejidad. Una forma intuitiva de tratar este asunto es utilizar enlaces indirectos dentro de la declaración de documentos, a modo de clave indirecta utilizada en bases de datos relacionales.

El siguiente ejemplo muestra la base de datos de una cadena de tiendas, pertenecientes a la colección *tienda*, que tienen asociados un identificador único, un nombre y la central a la que pertenecen, en su caso. Primeramente se muestra cómo se insertaría en la base de datos, para continuar con una selección de información.

```
>db.tienda.insert({_id: ObjectId("XXXX"), nombre: 'Gran Plaza'});

>db.tienda.insert({_id: ObjectId("YYYY"), nombre: 'Dos Hermanas', central: ObjectId("XXXX")});

>db.tienda.insert({_id: ObjectId("ZZZZ"), nombre: 'Carlos V', central: ObjectId("XXXX)});
```

En estos ejemplos hemos relacionado las tiendas de Carlos V, y Dos Hermanas con una sede central de la cual dependen: Gran Plaza.

Una rápida consulta nos daría las tiendas que dependen de Gran Plaza:

```
>db.tienda.find({central: ObjectId("Gran Plaza")});
```

El resultado de lo anterior sería la aparición por pantalla de los documentos con nombre Dos Hermanas, y CarlosV.

Por supuesto, la potencialidad de esta función se incrementa con el uso de arrays, siendo capaces entonces de crear dependencias múltiples:

>db.tienda.insert({_id: ObjectId("HHHH", nombre: 'Avd Ciencias', central: [ObjectId: ("XXXX"), ObjectId("ZZZZ")]});

Puede observarse que la nueva tienda creada, Avd Ciencias tiene dos dependencias, una con Carlos V y otra con Gran Plaza. Esto se ha conseguido asociando el iD de las dos tiendas, a la central de la que depende la recientemente creada.

De la misma forma, hay otras acciones que otorgan mayor potencial a MongoDB, como es el anidamiento de documentos. En MongoDB se puede trabajar con documentos embebidos, facilitando la tarea de tratar con los datos en algunos casos específicos:

>db.tienda.insert({_id: ObjectId("AAAA"), nombre: 'Capuchinos', servicios: {frescos: 'Abart', congelados: 'Sisco', pedidos: ObjectId("XXXX")}});

Hemos utilizado el documento *servicios* como parte del documento *tienda*. El documento servicios contiene los proveedores asociados a la tipología de productos que pueden encontrarse en la tienda, como pueden ser la gama de productos frescos, congelados, dietéticos, o de ocio. Para hacer una consulta en la que aparezcan documentos embebidos, la solución pasa por utilizar la siguiente nomenclatura:

>db.tienda.find({'servicios.frescos': 'Abart'})

Este último comando encontrará las tiendas que tienen como proveedor de servicios de productos frescos aquel con nombre Abart.

Este capítulo ha profundizado en la base de datos NoSQL MongoDB, que se presenta como la más popular de entre las que existen en el mercado. Se ha destacado su particular arquitectura, muy diferente de las bases de datos clásicas; también se han visto ejemplos que han permitido comprender cómo funciona su modelo de datos, y como interactúan las herramientas que provee MongoDb con los documentos almacenados.

Una vez concluido el bloque 2, se ha podido comprobar las muy diferentes perspectivas que presentan las tres tecnologías estudiadas, centrándose cada una de ellas en aspectos importantes en su concepción: Robustez en el caso de MySQL, integración en el caso de Marklogic, y rapidez en el caso de MongoDB. Estudiadas sus características puede concluirse que cumplen sobradamente con el propósito anterior.

El siguiente bloque, va a presentar las aportaciones propias que se han hecho en este Proyecto Final de Carrera, que incluyen un test de rendimiento en el que se han comparado las bases de datos de las tres tecnologías a las que se han hecho referencia desde el comienzo, y que consiste en un código escrito en lenguaje C# que va a comprobar las estadísticas temporales que arrojan las inserciones, selecciones y actualizaciones de múltiples registros.

También forma parte del bloque 3 la herramienta de e-Assessment desarrollada, y que ya fue introducida en los objetivos del proyecto. Esta herramienta va a servir para la evaluación de los alumnos de Programación Orientada a Objetos del Grado de Ingeniería de Sistemas de Telecomunicación, y va a consistir en la comunicación bidireccional de dos aplicaciones, una que actuará como cliente y otra como servidor, que compilará y ejecutará remotamente el código Java escrito por los alumnos, pasando una serie de pruebas que determinarán la evaluación de los usuarios. El código de estas aplicaciones está escrito en lenguaje C.