The Little MongoDB Book

by Karl Seguin

Acerca de este libro

Licencia

Acerca del autor

Karl Seguin es un desarrollador con experiencia en varios campos y tecnologías. Es un experto en .NET y desarrollador en ruby. Es un colaborador semi-activo de proyectos OSS, escritor técnico y presentador ocasional. Con respecto a MongoDB, fue un colaborador principal de la librería C# NoRM, escribió el tutorial interactivo mongly así como el administrador web Mongo Mongo Web Admin. Su servicio gratuito para desarrolladores de juegos casuales mogade.com, esta soportado con MongoDB.

Su blog esta en: http://openmymind.net y sus tweets vía @karlseguin

Agradecimientos

Gracias especiales a Perry Neal por prestarme sus ojos, mente y pasión. Me proporciono una ayuda invaluable. Gracias.

Ultima Versión

La ultima versión de este libro esta disponible en:

http://github.com/karlseguin/the-little-mongodb-book. (ENG) https://github.com/uokesita/the-little-mongodb-book (ESP)

Introducción

No es mi culpa que los capítulos sean cortos, MongoDB solo es fácil de aprender.

A menudo se dice que la tecnología avanza a rápidamente. Es cierto que hay una creciente lista de nuevas tecnologías y técnicas que son liberadas. Sin embargo, desde hace mucho tiempo siempre he sido de la opinión de que las tecnologías fundamentales utilizadas por los programadores se mueven a un ritmo más bien lento. Uno podría pasar años aprendiendo poco pero seguir siendo relevantes.

Lo que es sorprendente es la velocidad a la que las tecnologías tradicionales son reemplazadas. Al parecer de la noche a la mañana, las tecnologías desde hace tiempo se ven amenazados por cambios en el enfoque de los desarrolladores.

Nada puede ser mas representativo de este cambio repentino que el progreso de tecnologías NoSQL contra bases de datos relacionales bien establecidas. Pareciera que un día la web fuera llevada por algunas RDBMS y al siguiente, 5 o mas soluciones NoSQL se han establecido como soluciones dignas.

Aunque estas transiciones parecen ocurrir de la noche a la mañana, la realidad es que puede tomar años en convertirse en practicas aceptadas. El entusiasmo inicial es llevado por un grupo de desarrolladores y compañías relativamente pequeños. Las soluciones son refinadas, lecciones aprendidas y ver que una nueva tecnología esta aquí para quedarse, otros lentamente lo intentan por si mismos. De nuevo, esto es particularmente cierto en el caso de NoSQL donde muchas soluciones no remplazan a las soluciones mas tradicionales de almacenamiento de datos, pero en lugar de eso, se enfocan en necesidades especificas ademas de lo que uno puede obtener de las soluciones tradicionales.

Habiendo dicho todo eso, lo primero que debemos hacer es explicar qué se entiende por NoSQL. Es un término amplio que significa diferentes cosas para diferentes personas. Personalmente, lo uso de manera muy amplia en el sentido de un sistema que interviene en el almacenamiento de datos. Dicho de otra manera, NoSQL (de nuevo, para mí), es la creencia de que la capa de persistencia no es necesariamente la responsabilidad de un solo sistema. Donde los proveedores de bases de datos relacionales históricamente han tratado de posicionar su software como la solución única para todo, NoSQL se inclina hacia unidades más pequeñas de responsabilidad donde la mejor herramienta para un trabajo determinado se puede aprovechar. Por lo tanto, tu stack NoSQL todavía puede aprovechar una bases de datos relacionales, por ejemplo MySQL, pero también puede contener Redis como la persistencia de búsqueda de partes específicas del sistema, así como Hadoop para el procesamiento intensivo de datos. En pocas palabras, NoSQL se trata de ser abierto y consciente de alternativas existentes y adicionales patrones y herramientas para la gestión de sus datos.

Puedes estarte preguntando donde cae MongoDB en todo esto. Como una base de datos orientada a documentos, Mongo es una solución mas general de NoSQL. Debe ser vista como una alternativa a las bases de datos relacionales. Como las bases de datos relacionales, también puede beneficiarse de la relación con otras soluciones NoSQL mas especializadas. MongoDB tiene ventajas y desventajas que cubriremos en otros capítulos de este libro.

Como pudiste haber notado, usamos el termino MongoDB y Mongo indistintamente.

Primeros Pasos

La mayoría de este libro se enfoca en el núcleo de la funcionalidad de MongoDB. Por lo tanto usaremos la terminal de MongoDB. La terminal es útil tanto para aprender como para usarla como una herramienta administrativa, tu código

usara un driver MongoDB.

Esto trae a colación lo primero que debes saber de MongoDB: sus drivers. MongoDB tiene un numero de drivers oficiales para varios lenguajes. Estos drivers pueden ser vistos como los diferentes drivers para bases de datos con los cuales probablemente estarás familiarizado. Sobre estos drivers, la comunidad de desarrollo han construido otros frameworks y librerías especificas para ciertos lenguajes. Por ejemplo, NoRM es una librería en C# que implementa LINQ, y MongoMapper es una librería ruby que es ActiveRecord-friendly. Así decidas programar directamente contra el núcleo de drivers de MongoDB, o alguna librería de alto nivel, es tu opción. Hago esta acotación solo porque muchas personas que vienen a MongoDB están confundidas sobre porque hay librerías oficiales y librerías de la comunidad, las primeras se enfocan en la comunicación/conectividad con MongoDB y la segunda esta centrada en las implementaciones de lenguaje/framework especificamente.

Mientras lees este libro, te invito a que juegues con MongoDB para replicar lo que demuestro así como explorar las dudas que puedas tener. Es fácil comenzar con MongoDB, así que tomemos unos minutos para establecer las cosas.

1. Ve a la pagina de descargas oficial y toma los binarios de la primera fila (la versión estable recomendada) para tu sistema operativo. Para efectos de desarrollo puedes tomar 32-bit o 64 bit.

2. Extrae el archivo (donde lo desees) y navega a la sub-carpeta `bin`. No ejecutes nada aun, pero ten en cuenta que `mongod` es el proceso para el servidor y `mongo` es la terminal cliente - estos son dos ejecutables en los que pasaremos la mayoría del tiempo.

3. Crea un nuevo archivo de texto en la sub-carpeta `bin` llamado `mongodb.config`

4. Añade una sola linea a tu mongod.config: `dbpath=RUTA_DONDE_QUIERES_GUARDAR_TUS_ARCHIVOS_DE_BASE_DE_DATOS`. Por ejemplo, en Windows podrías hacer `dbpath=c:\mongodb\data` y en Linux `dbpath=/etc/mongodb/data`.

5. Asegúrate que la ruta `dbpath` que especificaste exista.

6. Lanza mongod con el parametro `--config /ruta/a/tu/mongodb.config`

Como ejemplo para usuarios windows, si extrajiste el archivo en `c:\mongodb\` y creaste `c:\mongodb\data\` entonces en `c:\mongodb\bin\mongodb.config` especificaras la ruta `dbpath=c:\mongodb\data\`. Puedes lanzar `mongod` desde la terminal con `c:\mongodb\bin\mongod --config c:\mongodb\bin\mongodb.config`.

Siéntete libre de agregar la carpeta `bin` a tu path para poder lanzar mongod sin estas especificaciones. Usuarios de MacOSX y Linux siguen los mismos pasos. La única cosa que debes cambiar son las rutas.

Espero que ahora tengas MongoDB corriendo. Si tienes un error, lee la salida con cuidado - el servidor es muy bueno explicando cual es el error.

Ahora puedes lanzar `mongo` (sin la *d*) que conectara tu terminal con el servidor que esta corriendo. Intenta escribir `db.version()` para asegurarte que todo funciona como debe ser. Si todo va bien veras el numero de la versión que has instalado.

Capitulo 1 - Lo Básico

Comenzamos nuestro viaje conociendo la mecánica básica de trabajar con MongoDB. Obviamente este es el núcleo para entender MongoDB, pero también nos ayuda a responder dudas técnicas acerca de MongoDB.

Para comenzar, hay seis simples conceptos que necesitamos entender.

1. MongoDB tiene el mismo concepto de 'base de datos' con el que probablemente estés familiarizado (o un schema para los amigos de Oracle). En una instancia de MongoDB puedes tener cero o mas bases de datos, cada una actuando como un contenedor de alto nivel para todo lo demás.

2. Una base de datos puede tener cero o mas 'colecciones'. Una colección comparte lo suficiente con lo que es tradicionalmente una tabla por lo que puedes pensar en ellas como la misma cosa.

3. Las colecciones están hechas de cero o mas 'documentos'. De nuevo, un documento puede ser visto como una 'fila'.

4. Un documento esta compuesto de uno o mas 'campos', probablemente puedas adivinar que son las 'columnas'.

5. 'Indices' en MongoDB funcionan muy parecido a sus equivalentes en RDBMS.

6. 'Cursores' son un concepto diferente de los otros 5 pero son tan importantes, y comúnmente pasados por alto, que pienso valdría la pena crear un tema de discusión para ellos. Lo importante a entender sobre los cursores es que cuando le pides datos a MongoDB, mongo retorna un cursor, con el que podemos hacer algunas cosas, como contar o saltar algún dato, sin descargarlos de hecho.

Para resumir, MongoDB esta hecho de `bases de datos` que contienen `colecciones`. Una `coleccion` esta hecha de `documentos`. Cada `documento` esta compuesto de `campos`. Las `colecciones` pueden ser indexadas, lo cual mejora la búsqueda y ordenamiento. Finalmente, cuando obtenemos datos de MongoDB lo hacemos a través de un cursor cuya ejecución es retrasada hasta que sea necesario.

Te preguntaras, porque usar una nueva terminología (colecciones vs tablas, documentos vs filas y campos vs columnas). Es solo para hacer las cosas mas complicadas? La verdad es que mientras estos conceptos son similares a sus contrapartes en base de datos relacionales, no son idénticos. La diferencia principal viene del hecho de que las bases de datos relacionales definen `columnas` a nivel de la `tabla` mientras que bases de datos orientadas a documentos definen sus `campos` a nivel de `documento`. Eso es, cada `documento` dentro de una `coleccion` puede tener su propio y único set de `campos`. Por lo tanto una `coleccion` es un contenedor silencioso en comparación a una `tabla`, mientras que un `documento` tiene mucha mas información que una `fila`.

A pesar de que esto es importante de entender, no te preocupes si las cosas no te quedan claras. No tomara mas que un par de inserciones para ver lo que realmente significa. Al final, el punto es que una colección no es estricta acerca de lo que contiene (sin schema). Los campos son rastreados con cada documento individual. Los beneficios e inconvenientes de esto serán explorados en un capitulo mas adelante.

Manos a la obra. Si no tienes MongoDB corriendo aun, ve y lanza el servidor `mongod` así como la terminal `mongo`. La terminal corre JavaScript. Hay algunos comandos globales que puedes ejecutar, como `help` (ayuda) o `exit` (salir). Los comandos que ejecutes contra la base de datos, son ejecutados a través de el objeto `db`, como `db.help()` (ayuda de la base de datos) o `db.stats()` (estadísticas sobre la base de datos).

Ve y escribe en la terminal `db.help()`, obtendrás una lista de los comandos que puedes ejecutar contra el objeto `db`.

Una pequeña nota. Como esta es una consola JavaScript, si ejecutas un método y omites los paréntesis (), veras el cuerpo del método en vez de la ejecución del mismo. Solo lo menciono porque la primera vez que lo hagas no te sorprendas cuando obtengas una respuesta que comience con `function (...){`. Por ejemplo, si escribes `db.help` (sin paréntesis), veras la implementación del método `help`.

Primero usaremos el método global `use` (usar) para cambiar de base de datos, inténtalo y escribe `use learn`. No importa si la base de datos aun no existe. La primera colección que crearemos de hecho creara la base de datos `learn`. Ahora que estas dentro de una base de datos puedes comenzar a utilizar comandos de bases de datos, como `db.getCollectionNames()`. Si lo haces, deberás obtener un arreglo vacio (`[ ]`). Ya que las colecciones son sin schema, no necesitamos crearlo. Podemos insertar simplemente un documento en una nueva colección. Para hacerlo, usa el comando `insert`, suministrando el documento a insertar.

```
db.unicorns.insert({name: 'Aurora', gender: 'f', weight: 450})
```

La linea anterior ejecuta `insert` contra la colección `unicorns`, pasándole un solo argumento. Internamente MongoDB usa un formato binario serializado en JSON. Externamente, esto significa que usaremos mucho JSON, como es el caso de nuestros parámetros. Si ejecutamos `db.getCollectionNames()`, veremos dos colecciones: `unicorns` y `system.indexes`. `system.indexes` es creado una vez por base de datos, y contiene la información de los indices de nuestra base de datos.

Ahora puedes usar el comando `find` contra `unicorns` para retornar una lista de documentos:

```
db.unicorns.find()
```

Fíjate que, a parte de los datos que especificas, hay un campo `_id`. Todo documento debe tener un campo `_id` único. Puedes generar uno por tu cuenta o dejar a MongoDB que genere un ObjectId por ti. La mayoría del tiempo probablemente quieras que MongoDB lo genere por ti. Por defecto, el campo `_id` es indexado - lo que explica porque la colección `system.indexes` fue creada. Puedes mirar `system.indexes`:

```
db.system.indexes.find()
```

Lo que estas viendo es el nombre del indice, la base de datos y colección contra la que fue creada y los campos incluidos en el indice.

Ahora, de vuelta a la discusión acerca de colecciones sin schema. Inserta un documento totalmente diferente en `unicorns` como:

```
db.unicorns.insert({name: 'Leto', gender: 'm', home: 'Arrakeen', worm: false})
```

Y, de nuevo usa `find` para listar los documentos. Luego que conocemos mas, discutiremos este interesante comportamiento de MongoDB, pero con suerte estarás comenzando a entender porque la terminología tradicional no es la mejor opción.

Dominando los Selectores

Ademas de los seis conceptos que hemos explorado, hay un aspecto practico de MongoDB que necesitas conocer antes de movernos a temas mas avanzados: selectores de consulta. Un selector de consulta MongoDB es como una

clausula where de una declaración SQL. Por lo tanto, la usaras cuando buscas, actualizas y borras documentos de las colecciones. Un selector es un objeto JSON, el mas simple es {} retorna todos los documentos (null funciona también). Si quisiéramos encontrar todos los unicorns hembras, usaríamos {gender:'f'}.

Antes de profundizar mas en los selectores, crearemos algo de data para jugar. Primero, borraremos lo que hemos colocado en la colección unicorns via: db.unicorns.remove() (como no le proveemos un selector, borrara todos los documentos). Ahora, haz los siguientes inserts para tener mas datos (Sugiero que copies y pegues esto):

```
db.unicorns.insert({name: 'Horny', dob: new Date(1992,2,13,7,47), loves: ['carrot
    ','papaya'], weight: 600, gender: 'm', vampires: 63});
db.unicorns.insert({name: 'Aurora', dob: new Date(1991, 0, 24, 13, 0), loves: ['
    carrot', 'grape'], weight: 450, gender: 'f', vampires: 43});
db.unicorns.insert({name: 'Unicrom', dob: new Date(1973, 1, 9, 22, 10), loves: ['
    energon', 'redbull'], weight: 984, gender: 'm', vampires: 182});
db.unicorns.insert({name: 'Roooooodles', dob: new Date(1979, 7, 18, 18, 44),
    loves: ['apple'], weight: 575, gender: 'm', vampires: 99});
db.unicorns.insert({name: 'Solnara', dob: new Date(1985, 6, 4, 2, 1), loves:['
    apple', 'carrot', 'chocolate'], weight:550, gender:'f', vampires:80});
db.unicorns.insert({name: 'Ayna', dob: new Date(1998, 2, 7, 8, 30), loves: ['
    strawberry', 'lemon'], weight: 733, gender: 'f', vampires: 40});
db.unicorns.insert({name: 'Kenny', dob: new Date(1997, 6, 1, 10, 42), loves: ['
    grape', 'lemon'], weight: 690,  gender: 'm', vampires: 39});
db.unicorns.insert({name: 'Raleigh', dob: new Date(2005, 4, 3, 0, 57), loves: ['
    apple', 'sugar'], weight: 421, gender: 'm', vampires: 2});
db.unicorns.insert({name: 'Leia', dob: new Date(2001, 9, 8, 14, 53), loves: ['
    apple', 'watermelon'], weight: 601, gender: 'f', vampires: 33});
db.unicorns.insert({name: 'Pilot', dob: new Date(1997, 2, 1, 5, 3), loves: ['
    apple', 'watermelon'], weight: 650, gender: 'm', vampires: 54});
db.unicorns.insert({name: 'Nimue', dob: new Date(1999, 11, 20, 16, 15), loves: ['
    grape', 'carrot'], weight: 540, gender: 'f'});
db.unicorns.insert({name: 'Dunx', dob: new Date(1976, 6, 18, 18, 18), loves: ['
    grape', 'watermelon'], weight: 704, gender: 'm', vampires: 165});
```

Ahora que tenemos datos, podemos aprender los selectores. {field: value} es usado para encontrar cualquier documento donde field es igual a a value. {field1: value1, field2: value2} es como haremos una declaración and. Los selectores especiales $lt, $lte, $gt, $gte y $ne son usados por las operaciones menor que, menor o igual que, mayor que, mayor o igual que y no igual a. Por ejemplo, para obtener todos los unicorns que pesen mas de 700 libras, podemos hacer:

```
db.unicorns.find({gender: 'm', weight: {$gt: 700}})
//o (no exactamente lo mismo, pero para propósitos de demostración)
db.unicorns.find({gender: {$ne: 'f'}, weight: {$gte: 701}})
```

El operador $exists es usado para determinar la presencia o ausencia de un campo, por ejemplo:

```
db.unicorns.find({vampires: {$exists: false}})
```

Debe retornar un solo documento. Si queremos un OR en ves de AND usamos el operador $or y le asignamos a un arreglo de valores que queremos:

```
db.unicorns.find({gender: 'f', $or: [{loves: 'apple'}, {loves: 'orange'}, {weight
    : {$lt: 500}}]})
```

Lo anterior retornara todos los unicorns hembras que le gusten las manzanas o las naranjas o pesen menos de 500 libras.

Hay algo muy elegante en nuestro ejemplo anterior. Tal ves ya lo notaste, pero el campo `loves` es un arreglo. MongoDB soporta arreglos como objetos de primera clase. Esto es una característica muy útil. Una vez que lo comienzas a usar, te preguntaras como has vivido sin el. Lo que es mas interesante es lo fácil que es seleccionar basándose en un arreglo: `{loves: 'watermelon'}` retornara cualquier documento donde `watermelon` sea un valor de `loves`.

Hay algunos otros operadores disponibles aparte de los que hemos visto. El mas flexible es `$where` que nos deja pasarle JavaScript para ejecutar en el servidor. Estos están todos descritos en la sección de Selecciones Avanzadas del sitio web de MongoDB. Lo que hemos cubierto hasta ahora es lo básico para comenzar. También es lo que mas usaras la mayoría del tiempo.

Hemos visto como estos selectores pueden ser usados con el comando `find`. También pueden ser usados el comando `remove` que vimos superficialmente, el comando `count`, que no hemos visto pero probablemente has podido descifrar, y el comando `update` en el que nos centraremos mas adelante.

El objeto `ObjectId` que MongoDB genera por nosotros de nuestro campo `_id` puede ser seleccionado via:

```
db.unicorns.find({_id: ObjectId("TheObjectId")})
```

En este Capitulo

No hemos visto el comando `update` aun, o lo creativos que podemos ser con el comando `find`. Sin embargo, si logramos tener corriendo MongoDB, vimos brevemente los comandos `insert` y `remove` (no son mucho mas de lo que vimos). También vimos `find` y los que son los selectores de MongoDB. Hemos tenido un buen comienzo y sentamos una base solida para lo que veremos mas adelante. Créelo o no, ya sabes la mayoría de lo que se trata MongoDB - de verdad esta hecho para aprenderlo rápido y de fácil uso. Te recomiendo fuertemente que juegues con tu copia local antes de continuar leyendo. Insertar diferentes documentos, nuevas colecciones, y familiarizarte con los diferentes selectores. Usa `find`, `count` y `remove`. Luego de algunos intentos por tu cuenta, las cosas que te parecieron extrañas al principio, con suerte comenzaran a tomar su lugar.

Capitulo 2 - Actualizando

En el capitulo 1 nos introducimos en tres de las cuatro operaciones del CRUD (create, read, update y delete). Este capitulo esta dedicado al que saltamos update. Update tiene algunos comportamientos sorprendentes, por lo que dedicaremos un capitulo a la misma.

Actualizando: Replace Versus $set

En su forma mas simple, update tiene 2 argumentos: el selector (where) a usar y el campo a actualizar. Si Roooooodles a ganado un poco de peso, podríamos ejecutar:

```
db.unicorns.update({name: 'Roooooodles'}, {weight: 590})
```

(Si has jugado con tu colección unicorns y ya no tienes los datos originales, sigue adelante y remove todos los documentos y vuelve a insertar el código en el capitulo 1.)

Si esto fuera código verdadero, probablemente habría que actualizar sus registros por _id, pero como yo no se que _id genero MongoDB para ti, usare names. Ahora bien, si nos fijamos en el registro actualizado:

```
db.unicorns.find({name: 'Roooooodles'})
```

Descubrirás la primera sorpresa de update, No se encuentran documentos porque el segundo parámetro suministrado es utilizado para **remplazar** el original. En otras palabras, el update encontró un documento por name y remplazo el documento completo con un nuevo documento (el segundo parámetro). Esto es diferente a como trabaja el comando update en SQL. En algunas situaciones, esto es ideal y se puede aprovechar para algunas actualizaciones realmente dinámicas. Sin embargo, cuando todo lo que quiero es modificar uno o algunos de los campos, es mejor usar el modificador $set de MongoDB:

```
db.unicorns.update({weight: 590}, {$set: {name: 'Roooooodles', dob: new Date
    (1979, 7, 18, 18, 44), loves: ['apple'], gender: 'm', vampires: 99}})
```

Esto va a restablecer los campos perdidos. No se sobrescribe el nuevo weight ya que no lo especifique. Ahora si ejecutamos:

```
db.unicorns.find({name: 'Roooooodles'})
```

Obtenemos el resultado esperado. Por lo tanto, la forma correcta de haber actualizado el weight en el primer lugar es:

```
db.unicorns.update({name: 'Roooooodles'}, {$set: {weight: 590}})
```

Modificadores de Update

Adicionalmente a $set, podemos aprovechar otros modificadores para hacer algunas cosas ingeniosas. Todos estos modificadores de Update trabajan en los campos - entonces el documento completo no sera remplazado o borrado. Por ejemplo, el modificador $inc es usado para incrementar un campo en cierto monto positivo o negativo. Por ejemplo, si a Pilot se le adjudico incorrectamente la muerte de un par de vampires, se podría corregir el error ejecutando:

```
db.unicorns.update({name: 'Pilot'}, {$inc: {vampires: -2}})
```

Si Aurora de pronto desarrolla un gusto por el dulce podemos agregarlo a su campo `loves` con el modificador `$push`:

```
db.unicorns.update({name: 'Aurora'}, {$push: {loves: 'sugar'}})
```

La sección de Actualizar del sitio web de MongoDB tiene más información sobre los modificadores de actualización disponibles.

Upserts

Una de las sorpresas mas agradables de los `updates` es que es completamente compatible con `upserts`. Un `upsert` actualiza el documento si lo encuentra o lo inserta sino lo encuentra. Upserts son útiles de tener en determinadas situaciones, cuando lo ejecutes, lo sabrás. Para habilitar upserting fijamos un tercer parámetro a `true`.

Un ejemplo clásico es un contador de visitas de un sitio web. Si queremos mantener un recuento total en tiempo real, tendríamos que ver si el registro ya existía para la página, y basado en eso decidiremos si correr un update o un insert. Con el tercer parámetro omitido (o establecido en false), ejecutando lo siguiente y no hará nada:

```
db.hits.update({page: 'unicorns'}, {$inc: {hits: 1}});
db.hits.find();
```

Sin embargo, si nos permiten upserts, los resultados son muy diferentes:

```
db.hits.update({page: 'unicorns'}, {$inc: {hits: 1}}, true);
db.hits.find();
```

Dado que no existen documentos con un campo `page` igual `unicorns`, se inserta un nuevo documento. Si se ejecuta una segunda vez, el documento existente se actualiza y `hits` se incrementa a 2.

```
db.hits.update({page: 'unicorns'}, {$inc: {hits: 1}}, true);
db.hits.find();
```

Actualizaciones Multiples

La ultima sorpresa que `update` tiene para ofrecer es que, por defecto, se va a actualizar un unico documento. Hasta el momento, por los ejemplos que hemos visto, esto puede parecer lógico. Sin embargo, si se ejecuta algo como:

```
db.unicorns.update({}, {$set: {vaccinated: true }});
db.unicorns.find({vaccinated: true});
```

Probablemente esperas encontrar todos tus preciosos `unicorns` a ser `vaccinated`. Para poder obtener eso, es necesario asignar un cuarto parametro a `true`:

```
db.unicorns.update({}, {$set: {vaccinated: true }}, false, true);
db.unicorns.find({vaccinated: true});
```

En Este Capitulo

Este capítulo concluye nuestra introducción a la base de operaciones CRUD disponible para una colección. Vimos `update` en detalle y observaron tres comportamientos interesantes. En primer lugar, a diferencia de un `update` de SQL, el `unicorns` de MongoDB reemplaza el documento actual. Debido a esto el modificador `$set` es bastante útil. En segundo lugar, `update` admite un intuitivo soporte para `upsert` que es particularmente útil cuando se combina con el modificador `$inc`. Finalmente, por defecto, `update` solo actualiza el primer documento encontrado.

Recuerda que estamos viendo MongoDB desde el punto de vista de su terminal. El driver y la librería que utilices puede modificar estos comportamientos predeterminados o exponer una API diferente. Por ejemplo, el driver de Ruby combina los dos últimos parámetros en un hash único: `{:upsert => false, :multi => false}`.

Capitulo 3 - Find

El capitulo 1 pudimos ver superficialmente el comando `find`. Hay mucho mas de `find` que entender los `selectores`. Ya mencionamos que el resultado del comando `find` es un `cursor`. Ahora veremos exactamente que significa esto en detalle.

Selección de Campos

Antes de que saltemos a los `cursors`, debes saber que `find` toma un segundo parámetro opcional. Este parámetro es la lista de los campos que quieres obtener. Por ejemplo, podemos obtener todos los nombres de unicorns ejecutando:

```
db.unicorns.find(null, {name: 1});
```

Por defecto, el campo `_id` siempre es retornado. Podemos excluirlo explicitamente especificando `{name:1, _id: 0}`.

A parte del campo `_id`, no puedes mezclar inclusión y exclusión de campos. Si piensas sobre eso, tiene sentido. Tu deseas seleccionar o excluir uno o más campos explícitamente.

Ordenamiento

He mencionado anteriormente que `find` retorna un cursor cuya ejecución es retardada hasta que se necesite. Sin embargo, lo que has observado sin duda desde la terminal es que `find` se ejecuta inmediatamente. Este es un comportamiento solo de la terminal. Podemos observar el verdadero comportamiento de los `cursors` viendo uno de los métodos que podemos unir a `find`. El primero que veremos es `sort` (ordenar). `sort` se comporta como la selección de campo de la sección anterior. Nosotros especificamos los campos por los cuales queremos ordenar, usando 1 para ascendente y -1 para descendente. Por ejemplo:

```
//los unicorns mas pesados primero
db.unicorns.find().sort({weight: -1})

//por nombre y luego por el numero de vampiros matados
db.unicorns.find().sort({name: 1, vampires: -1})
```

Como en una base de datos relacional, MongoDB puede usar un indice para ordenar. Veremos los indices en detalle mas adelante. Sin embargo, debes saber que MongoDB limita el tamaño de tu ordenamiento sin un indice. Esto es, si intentas ordenar un resultset grande que no puede usar un indice, tendrás un error. Algunas personas ven esto como una limitante. Realmente, desearía que mas bases de datos tuvieran la capacidad de negar queries no optimizadas (No voy a convertir cada inconveniente MongoDB en algo positivo, pero he visto bases de datos tan mal optimizadas que deseo sinceramente que había un modo-estricto.)

Paginación

Paginar los resultados puede ser logrado a través de los métodos de cursores `limit` y `skip`. Para obtener el segundo y tercer unicorn mas pesados, podemos hacer:

```
db.unicorns.find().sort({weight: -1}).limit(2).skip(1)
```

Usando `limit` en conjunto con `sort`, es una buena forma de evitar encontrarnos con problemas cuando hacemos ordenamiento en campo sin indice.

Cuentas

La terminal de Mongo hace posible ejecutar `count` directamente en una colección, así:

```
db.unicorns.count({vampires: {$gt: 50}})
```

En realidad, `count` es un metodo `cursor`, la terminal solo provee un atajo. Los drivers que no proveen ese atajo necesitan ser ejecutado así (que también funcionara en la terminal):

```
db.unicorns.find({vampires: {$gt: 50}}).count()
```

En este capitulo

El uso de `find` y `cursors` es algo sencillo. Hay algunos comandos adicionales que cubriremos en capítulos siguientes los cuales serán casos extremos, pero hasta ahora, debes estar cómodo trabajando con la terminal mongo y entendiendo los fundamentos de MongoDB.

Capitulo 4 - Modelado de datos

Vamos a cambiar maquinas y tener una conversación mas abstracta de MongoDB. Explicar nuevos términos y nueva sintaxis es una tarea facil. Tener una conversación sobre modelado con un nuevo paradigma no lo es. La verdad es que la mayoría de nosotros aun estamos encontrando lo que funciona y lo que no con respecto al modelado con estas nuevas tecnologias. Es una conversación que podemos comenzar a tener, pero al final tendras que practicar y aprender con código real.

Comparado a la mayoría de soluciones NoSQL, las bases de dato orientadas a objetos son probablemente las menos diferentes, comparadas a las bases de datos relacionales, cuando se trata de modelado. Las diferencias que existen son sutiles pero no quiere decir que no sean importantes.

No Joins

La primera diferencia y la mas fundamental con la que tendras que acostumbrarte es la falta de joins en MongoDB. No se la razón especifica por la que algunos tipos de joins no están soportados en MongoDB, pero si se, que los joins son vistos generalmente como no escalables. Esto es, una vez que empiezas a separar tu datos horizontalmente, terminas usando joins en el lado del cliente (la aplicación en enl servidor). Cuales quiera sean las razones, el hecho es que los datos *son* relacionales, y MongoDB no soporta joins.

Sin saber mas nada, para vivir en un mundo sin joins, debemos hacer joins nosotros mismo en nuestro código de aplicación. Esencialmente debemos hacer un query para encontrar los datos relevantes. Configurar nuestros datos no es nada diferente a declarar una clave foranea en una base de datos relacional. Vamos a quitarle el foco a nuestros `unicorns` y usaremos nuestros `employees` (empleados). Lo primero que haremos es crear nuestro empleados (Le proveo un `_id` explicito para tener ejemplos coherentes).

```
db.employees.insert({_id: ObjectId("4d85c7039ab0fd70a117d730"), name: 'Leto'})
```

Ahora agregaremos algunos empleados, y colocamos como su manager a `Leto`:

```
db.employees.insert({_id: ObjectId("4d85c7039ab0fd70a117d731"), name: 'Duncan',
    manager: ObjectId("4d85c7039ab0fd70a117d730")});
db.employees.insert({_id: ObjectId("4d85c7039ab0fd70a117d732"), name: 'Moneo',
    manager: ObjectId("4d85c7039ab0fd70a117d730")});
```

(Vale la pena recalcar que el `_id` puede ser cualquier valor unico. Como casi siempre usaras `ObjectId` en la vida real, lo usaremos tambien)

Por supuesto, para encontrar todos los empleados de Leto, simplemente ejecutas:

```
db.employees.find({manager: ObjectId("4d85c7039ab0fd70a117d730")})
```

No hay nada magico en esto. En el peor escenario, la mayoría del tiempo, la falta de joins solo requerira de un query extra (casi siempre indexado)

Arreglos y Documentos Embebidos Solo porque MongoDB no tenga joins no quiere decir que no tenga algunos trucos bajo su manga. Recuerdas cuando vimos que MongoDB soporta arreglos como objetos primarios de un documento? Resulta que esto es increíblemente bueno cuando lidiamos con relaciones muchos-a-uno o muchos-a-muchos. Como un ejemplo simple, si un empleado puede tener dos managers, podemos almacenarlos en un arreglo:

```
db.employees.insert({_id: ObjectId("4d85c7039ab0fd70a117d733"), name: 'Siona',
    manager: [ObjectId("4d85c7039ab0fd70a117d730"), ObjectId("4
    d85c7039ab0fd70a117d732")] })
```

Es de particular interes que, para algunos documentos, `manager` puede ser un valor escalar, mientras que para otros puede ser un arreglo. Nuestro query original `find` funcionara para ambos:

```
db.employees.find({manager: ObjectId("4d85c7039ab0fd70a117d730")})
```

You'll quickly find that arrays of values are much more convenient to deal with than many-to-many join-tables.

Besides arrays, MongoDB also supports embedded documents. Go ahead and try inserting a document with a nested document, such as:

```
db.employees.insert({_id: ObjectId("4d85c7039ab0fd70a117d734"), name: 'Ghanima',
    family: {mother: 'Chani', father: 'Paul', brother: ObjectId("4
    d85c7039ab0fd70a117d730")}})
```

En el caso de que te pregunte, los documentos embebidos puedes ser accedidos usando la notación de punto:

```
db.employees.find({'family.mother': 'Chani'})
```

Hablaremos brevemente de donde se posicionan los documentos embebidos y como usarlos.

DBRef MongoDB soporta algo conocido como `DBRef` que es una convención soportada por muchos drivers. Cuando un driver encuentra un `DBRef` puede traer automaticamente el documento. Un `DBRef` incluye la colección y el id que referencia al documento. Generalmete sirve para un proposito especifico: cuando documentos de la misma colección pueden referenciar documentos de una colecciona diferente entre si. Esto es, el `DBRef` del documento1 puede apuntar a un documento en `managers` y el `DBRef` del documento2 puede apuntar a un documento es `employees`.

Desnormalización Otra alternativa a el uso de joins es desnormalizar tus datos. Historicamente, la desnormalización estaba reservada para código sensible a rendimiento, o cuando los datos debían ser guardados en instantáneas (como en un registro de auditoria). Sin embargo, con la popularidad creciente de NoSQL, que en la mayoria no tiene joins, la desnormalización como parte de un modelado normal se esta convirtiendo en lago muy comun. Esto no significa que debes duplicar cada pieza de información en cada documento. Son embargo, en vez de dejar que el miedo de datos duplicados maneje tus decisioes, considera modelar tus datos en base a que información pertenece a que documento.

Por ejemplo, digamos que estas desarrollando una aplicación de foro. La manera tradicional de asociar un `user` especifico a un `post` es via la columna `userid` en `posts`. Con ese modelo no puedes mostrar los `posts` sin traerte (con join) `users`. Una alternativa posible es simplemente guardar `name` y `userid` en cada `post`. Tambien podrías hacerlo con documentos embebidos, `user: {id: ObjectId('Something'), name: 'Leto'}`. Si, si dejas que tus usuarios cambien su nombre, tendras que actualizar cada documento (lo que significa 1 query extra).

Ajustarse a esta forma de hacer las cosas no sera facil para algunos. La mayoría de la veces no tendra sentido hacer esto. No temas experimentar con este enfoque. No sólo es adecuado en algunas circunstancias, pero también puede ser la manera correcta de hacerlo.

Cual debes escoger? Arrays of ids are always a useful strategy when dealing with one-to-many or many-to-many scenarios. It's probably safe to say that `DBRef` aren't use very often, though you can certainly experiment and play with them. That generally leaves new developers unsure about using embedded documents versus doing manual referencing.

First, you should know that an individual document is currently limited to 4 megabytes in size. Knowing that documents have a size limit, though quite generous, gives you some idea of how they are intended to be used. At this point, it seems like most developers lean heavily on manual references for most of their relationships. Embedded documents are frequently leveraged, but mostly for small pieces of data which we want to always pull with the parent document. A real world example I've used is to store an `accounts` document with each user, something like:

```
db.users.insert({name: 'leto', email: 'leto@dune.gov', account: {allowed_gholas:
    5, spice_ration: 10}})
```

That doesn't mean you should underestimate the power of embedded documents or write them off as something of minor utility. Having your data model map directly to your objects makes things a lot simpler and often does remove the need to join. This is especially true when you consider that MongoDB lets you query and index fields of an embedded document.

Pocas o Muchas Colecciones

Dado que las colecciones no se rigen por un schema, es posible construir un sistema completo en una sola colección con documentos disparejos. Por lo que he visto, la mayoría de los sistemas MongoDB están formados similarmente a lo que puedes encontrar en un sistema relacional. En otras palabras, si existe una tabla en una base de datos relacionales, podría haber una collection en MongoDB (tablas joins de muchos-a-muchos es una excepción importante)

La conversación se vuelve mas interesante cuando consideras embeber documentos. El ejemplo que se ve frecuentemente es un blog. Deberías tener una colección `posts` y una colección `comments`, o cada `post` debería tener un arreglo de `comments` embebido en el. Dejando a un lado el limite de 4MB por el momento (todo el Hamlet es menos de 200KB, así que que tan popular es tu blog), la mayoría de los desarrolladores prefieren separar las cosas. Es simplemente mas limpio y explicito.

No hay ninguna regla (excepto por los 4MB). Juega con las diferentes opciones y tendras una idea de lo que se siente bien o no.

En Este Capitulo

Nuestro objetivo en este capitulo fue proveer alguna ayuda para guiarte a modelar tus datos en MongoDB. Un punto de partida. El modelado en un sistema orientado a documentos es diferente, pero no tan diferente al mundo relacional.

Tienes un poco mas de flexibilidad y una limitación, pero para un nuevo sistema, las cosas tienden a tomar su lugar. La unica forma de algo vaya mal es no intentando.

Capitulo 5 - Cuando usar MongoDB

A estas alturas ya debes tener un conocimiento suficiente de MongoDB para tener una idea de dónde y cómo podría encajar en tu sistema existente. Ya hay tantas tecnologías de almacenamiento nuevas que es fácil sentirse abrumado por todas las opciones.

Para mí, la lección más importante, que no tiene nada que ver con MongoDB, es que ya no tienes que depender de una única solución para hacer frente a tus datos. Sin duda, una solución única tiene ventajas obvias y para un montón de proyectos, tal vez incluso la mayoría, una solución única es el enfoque mas sensato. La idea no es que se deben utilizar diferentes tecnologías, sino que se puede. Sólo tu sabes si los beneficios de la introducción de una nueva solución son superiores a los costos.

Dicho esto, tengo la esperanza de que lo que hemos visto hasta ahora te ha hecho ver MongoDB como una solución general. Se ha mencionado un par de veces que las bases de datos orientada a documentos comparten mucho en común con las bases de datos relacionales. Por lo tanto, en lugar de lanzar puntillas a su alrededor, vamos a dejar en claro que MongoDB debe ser visto como una alternativa directa a las bases de datos relacionales. Cuando uno puede ver Lucene como la mejora de una base de datos relacional con indexación de texto completo, o Redis como un almacén persistente de clave y valor, MongoDB es un repositorio central de datos.

Ten en cuenta que no llame MongoDB un *reemplazo* de bases de datos relacionales, sino más bien una **alternativa**. Es una herramienta que puede hacer lo que un montón de otras herramientas puede hacer. Parte de ellas MongoDB lo hace mejor, algunas de ellas MongoDB lo hace peor. Vamos a analizar las cosas un poco más.

Sin schema

Un beneficio que no hemos discutido de las bases de datos orientadas a documentos es que son sin schema. Esto hace que sean mucho más flexibles que las tablas de bases de datos tradicionales. Estoy de acuerdo en ese Schema-less es una buena característica, pero no la razón principal que la mayoría de la gente menciona.

La gente habla de schema-less como si de pronto comenzara a almacenar datos locos o sin relación. Hay dominios y sets de datos que pueden ser un dolor de cabeza modelar usando bases de datos relacionales, pero esos son casos particulares. Schema-less es cool, pero la mayoría de tus datos estarán estructurados. Es cierto que tener un desfase de vez en cuando puede ser útil, especialmente cuando se introducen nuevas características, pero en realidad no es nada que una columna null no pueda resolver igual de bien.

Para mí, el beneficio real del schema-less es la falta de la configuración y la reducción de la fricción con la OOP. Esto es particularmente cierto cuando se trabaja con un lenguaje estático. He trabajado con MongoDB en C# y Ruby, y la diferencia es notable. El dinamismo de Ruby y su popular implementación de ActiveRecord ya reduce gran parte de la falta de concordancia objeto-relacional. Eso no quiere decir MongoDB no es un buen partido para Ruby, lo es realmente. Más bien, creo que la mayoría de los desarrolladores de Ruby verían MongoDB como una mejora incremental, mientras que C# o los desarrolladores de Java verán un cambio fundamental en cómo interactúan con sus datos. Piensa en ello desde la perspectiva de un desarrollador de controladores. Desea guardar un objeto? Serializa a JSON (técnicamente BSON, pero lo suficientemente cerca) y envialo a MongoDB. No hay asignación de propiedad o la asignación de tipos. Esta sencillez definitivamente te beneficia a ti, el desarrollador final.

Escrituras

Un area en la que se especializa MongoDB es logging. Las inserciones en MongoDB son, por defecto, asíncronas. Escrituras en MongoDB son bastante rápidas, y hacerlas asíncronas solo las hace mas rápidas. Adicionalmente, log datos es algo que puede tomar ventaja de colecciones sin schema. Finalmente, MongoDB tiene algo llamado colecciones con tope. Hasta ahora, la simplicidad de las colecciones que hemos creado son colecciones normales. Podemos crear colecciones con tope usando el comando `db.createCollection` e indicando que tendra un tope:

```
    //limitar nuestra colección a 1 megabyte
db.createCollection('logs', {capped: true, size: 1048576})
```

Cuando la colección con tope llegue a su limite de 1MB, los documentos viejos son borrados automáticamente. Un limite en el numero de documentos, en vez de el tamaño, puede ser habilitado usando `max`. Las colecciones con tope tienen unas propiedades interesantes. Por ejemplo, puedes actualizar un documento pero no puede crecer en tamaño. Ademas, el orden de inserción es preservado, así no tienes que añadir un indice extra para obtener el ordenamiento basado en tiempo.

Este es un buen momento para decir que si no deseas que tus escrituras sean asíncronas simplemente ejecutas el siguiente comando: `db.getLastError()`. La mayoría de los drivers encapsulan esto como una "escritura segura", especificando `{:safe => true}` como segundo parametro en el `insert`.

Durabilidad

Antes de la versión 1.8, MongoDB no tenia la durabilidad de un solo servidor. Es decir, una caída del servidor podría resultar en la pérdida de datos. La solución ha sido siempre ejecutar MongoDB en una configuración multi-servidor (MongoDB admite la replicación). Una de las principales características se agregaron a la version 1.8 fue `journaling`. Para habilitarla anade una nueva línea con `journal = true` al archivo `mongodb.config` archivo que creamos cuando configuramos MongoDB (y reiniciar el servidor si deseas usarlo inmediato). Probablemente quieras `journaling` habilitado (estará habilitado por defecto en una futura version). Aunque, en algunas circunstancias, el rendimiento extra que se obtiene de deshabitar el `journaling` podría ser un riesgo que está dispuesto a tomar. (Vale la pena señalar que algunos tipos de aplicaciones pueden darse el lujo de perder datos).

La Durabilidad solo es mencionada aqui porque se ha hecho mucho en MongoDB alrededor de la falta de la durabilidad de un solo servidor. La información que encuentres acerca de esta característica faltante es simplemente por estar desactualizado.

Búsqueda de texto completo

La verdadera capacidad de búsqueda de texto es algo que se espera llegará a MongoDB en una versión futura. Con su apoyo a las matrices, de búsqueda en base a texto completo es bastante fácil de implementar. Para algo más potente, tendrás que confiar en una solución, como Lucene / SOLR. Por supuesto, esto también es cierto de muchas bases de datos relacionales.

Transacciones

MongoDB no tiene transacciones, para eso tiene dos soluciones, una que es grandiosa, pero con un uso limitado, y la otra que es pesada pero flexible.

La primera son sus muchas operaciones atómicas. Esto es bueno, siempre y cuando en resuelvan su problema. Ya hemos visto algunos de los más simples, como `$incy` `$ set`. También hay comandos como `findAndModify`, que pueden actualizar o eliminar un documento y retornarlo de forma atómica.

La segunda, cuando las operaciones atómicas no son suficientes, es caer de nuevo a una confirmación en dos fases. Una confirmación de dos fases es lo que eliminación de referencias manuales es a los joins. Es una solución de almacenamiento independiente que haces en código. Las confirmaciones de dos pasos son en realidad muy populares en el mundo relacional como una manera de poner en práctica las transacciones a través de múltiples bases de datos. El sitio web MongoDB [tiene un ejemplo] (http://www.mongodb.org/display/DOCS/two-phase+commit) que ilustra el escenario más común (una transferencia de fondos). La idea general es que se guarde el estado de la transacción en el documento actual que se esta actualizando y seguir los pasos init-pending-commit/rollback manualmente.

El soporte MongoDB para documentos anidados y sin schema hacen las confirmación en dos fases un poco menos dolorosas, pero todavía no es un gran proceso, especialmente cuando se esta comenzando con él.

Procesamiento de Datos

MongoDB se basa en MapReduce para la mayoría del trabajo de procesamiento de datos. Tiene capacidades básicas de agregación, pero para cualquier cosa seria, querrás usar MapReduce. En el proximo capitulo veremos MapReduce en detalle. Por ahora puedes pensar en ello como una forma poderosa y diferente de `group by` (que esta subestimado). Uno de los puntos fuertes de MapReduce es que puede ser paralelizado para trabajar con grandes cantidades de datos. Sin embargo, la implementación de MongoDB se basa en JavaScript que es de un solo hilo. El punto? para procesar grandes cantidades de datos, necesitaras basarte en otra cosa como Hadoop. Afortunadamente como los dos sistemas se complementan hay un adaptador de MongoDB para Hadoop.

Por supuesto, paralelización de procesamiento de datos no es algo en que las bases de datos relacionales sobresalgan. Hay planes para las futuras versiones de MongoDB para mejorar en el manejo de conjuntos muy grandes de datos.

Coordenadas Geoespaciales

Una poderosa caracteristica de MongoDB es su soporte para indeces geoespaciales. Esto te permite almacenar coordenadas `x` y `y` en los documentos y luego encontrar documentos que esten cerca de un set de coordenadas o `$within` una caja o un circulo. Esta es una caracteristica que es explicada mejor via algun aditamiento visual, asi que te invito a probar el tutorial interactivo de 5 minutos en coordenadas geoespaciales si quieres saber mas.

Herramientas y Madurez

Probablemente ya sabes la respuesta a esto, pero MongoDB es obviamente más joven que la mayoría de los sistemas de bases de datos relacionales. Esto es algo que absolutamente deberías considerar. Cuanto peso tiene depende de

lo que estas haciendo y como lo estas haciendo. Sin embargo, honestamente no se puede ignorar el hecho de que MongoDB es más joven y las herramientas disponibles no son muy grandes (aunque las herramientas en torno a una gran cantidad de bases de datos relacionales maduras es bastante horrible también!). A modo de ejemplo, la falta de apoyo para la base-10 en números de punto flotante, obviamente, será una preocupación (aunque no necesariamente una traba) para los sistemas de manejo de dinero.

En el lado positivo, los drivers existen para un gran número de idiomas, el protocolo es moderno y simple, y el desarrollo está sucediendo a una velocidad rapida. MongoDB esta en producción en suficientes compañías que, preocupaciones por madurez, aunque validas, están convirtiendo rapidamente en algo del pasado.

En este Capitulo

El mensaje de este capítulo es que MongoDB, en la mayoría de los casos, puede sustituir a una base de datos relacional. Es mucho más simple y directo, es más rápido y por lo general impone menos restricciones a los desarrolladores de aplicaciones. La falta de transacciones puede ser una preocupación legítima y seria. Sin embargo, cuando la gente pregunta * ¿dónde esta MongoDB con respecto al nuevo paisaje de almacenamiento de datos? * la respuesta es simple: **justo en el centro **.

Capitulo 6 - MapReduce

MapReduce es un enfoque al procesamiento de datos que tiene dos grandes beneficios sobre los modelos tradicionales. La primera y principal razón es que fue desarrollado enfocado en el rendimiento. En teoria MapReduce puede correr en paralelo permitiendonos así procesar largos conjuntos de datos utilizando muchos procesadores o maquinas. Como ya mencionamos, esto no es algo que MongoDB puede hacer actualmente. El segundo beneficio de MapReduce es que uno escribe código para procesar la información. Comparado con lo que se hiciera en SQL, MapReduce es mucho mas rico y te deja aprovechar las ventajas del lenguaje de programación que estes usando.

MapReduce es un patron que ha crecido en popularidad y que puedes usar en casi todos los lenguajes como: C#, Ruby, Java, Phyton, etc. Quiero advertirte que esto sonara muy complicado y diferente la primera vez que lo uses pero no te preocupes. Toma tu tiempo y juega con MapReduce. Vale la pena aprenderlo así estes usando MongoDB o no.

Mezclando un poco de Teoria y Practica

MapReduce se divide en dos pasos. El primero lo llamamos "MAP" o Mapa en español y el segundo "Reduce" o reducir en español. El proceso de crear un mapa transforma los documentos que le enviamos y emite un par de una clave y un valor o (key => value pair). El paso de reducir toma las claves y el arreglo de valores emitidos para esas claves y produce el resultado final. Expliquemos un poco mas:

El ejemplo que usaremos sera generar un reporte del numero de hits por dia que tenemos en una pagina. Esto es el "Hola Mundo" del MapReduce. Para esto vamos a usar una colección de 'hits' con dos campos: recursos y fecha. El resultado que esperamos esta dividido por 'recurso', 'año', 'mes', 'dia', 'veces'

Supongamos que tenemos la siguiente información:

```
recurso      fecha
index        Jan 20 2010 4:30
index        Jan 20 2010 5:30
about        Jan 20 2010 6:00
index        Jan 20 2010 7:00
about        Jan 21 2010 8:00
about        Jan 21 2010 8:30
index        Jan 21 2010 8:30
about        Jan 21 2010 9:00
index        Jan 21 2010 9:30
index        Jan 22 2010 5:00
```

Esperaríamos el siguiente resultado:

```
recurso   año   mes      dia   veces
index     2010   1       20    3
about     2010   1       20    1
about     2010   1       21    3
index     2010   1       21    2
index     2010   1       22    1
```

(Lo bueno acerca este tipo de enfoque es que al ordenar el resultado, los reportes son rápidos de generar y el crecimiento de la información esta controlado ya que solo usamos un documento de mongo por cada recurso que queramos rastrear)

Por ahora solo nos enfocaremos en entender los conceptos. Al final de este capitulo te proporcionaremos toda la data que usamos y el código para que juegues por ti mismo.

Lo primero que haremos es ver la funciona 'Map' o Mapa. El objetivo de esta función es que retorne un valor que pueda ser reducido. Es posible que emita 0 o mas veces. En nuestro caso, siempre emitira una sola vez (muy común). Imagina que Map es un bucle a través de todos los hits. Por cada documento queremos generar, una clave con un recurso, año, mes, dia y valor.

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

`this` se refiere al documento que estamos procesando actualmente. Vamos a ver el resultado de esta función para entender un poco mas esto. Estamos usando la información que tenemos dos párrafos mas arriba. El resultado de la función Map seria:

```
{resource: 'index', year: 2010, month: 0, day: 20} => [{count: 1}, {count: 1}, {
    count:1}]
{resource: 'about', year: 2010, month: 0, day: 20} => [{count: 1}]
{resource: 'about', year: 2010, month: 0, day: 21} => [{count: 1}, {count: 1}, {
    count:1}]
{resource: 'index', year: 2010, month: 0, day: 21} => [{count: 1}, {count: 1}]
{resource: 'index', year: 2010, month: 0, day: 22} => [{count: 1}]
```

El entendimiento de este paso intermediario es esencial para el entendimiento de MapReduce. Los valores que emite están agrupados como arreglos por clave. Los desarrollares de .NET y Java pueden pensar de esto como un: `IDictionary<object, IList<object>>` (.NET) o `HashMap<Object, ArrayList>` (Java).

Cambiemos la funcion de MAP un poco mas:

```
function() {
    var key = {resource: this.resource, year: this.date.getFullYear(), month:
        this.date.getMonth(), day: this.date.getDate()};
    if (this.resource == 'index' && this.date.getHours() == 4) {
        emit(key, {count: 5});
    } else {
        emit(key, {count: 1});
    }
```

```
}
```

El primer resultado cambiaria a:

```
{resource: 'index', year: 2010, month: 0, day: 20} => [{count: 5}, {count: 1}, {
    count:1}]
```

Observa como cada `emit` genera un nuevo valor agrupado por nueva clave.

La función reduce o Reducir en español toma cada uno de estos valores intermedios y emite un resultado final. Nuestra función para reducir es:

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

El resultado de esta es:

```
{resource: 'index', year: 2010, month: 0, day: 20} => {count: 3}
{resource: 'about', year: 2010, month: 0, day: 20} => {count: 1}
{resource: 'about', year: 2010, month: 0, day: 21} => {count: 3}
{resource: 'index', year: 2010, month: 0, day: 21} => {count: 2}
{resource: 'index', year: 2010, month: 0, day: 22} => {count: 1}
```

Tecnicamente el resultado en MongoDB es:

```
_id: {resource: 'home', year: 2010, month: 0, day: 20}, value: {count: 3}
```

Como puedes notar este es el resultado que esperábamos.

Si has estado prestando atención te preguntaras: *¿Por que no simplemente usamos* `sum = values.lenght`*?* Esto parece un enfoque eficiente cuando estas sumando arreglos de 1. La verdad es que en pocas ocasiones llamamos a la función Reduce con un set perfecto. Por ejemplo en vez de ser llamado con:

```
{resource: 'home', year: 2010, month: 0, day: 20} => [{count: 1}, {count: 1}, {
    count:1}]
```

Reduce puede ser llamado con:

```
{resource: 'home', year: 2010, month: 0, day: 20} => [{count: 1}, {count: 1}]
{resource: 'home', year: 2010, month: 0, day: 20} => [{count: 2}, {count: 1}]
```

El resultado final es el mismo (3) pero el camino tomado para llegar a el es diferente. Por lo tanto la función reduce siempre debe ser indempotente. Esto es que la función debería dar el mismo resultado así se llame una sola vez o varias veces.

Es muy común que encadenemos varias funciones Reduce cuando hacemos análisis complejos.

Pura Practica

Con MongoDB usamos el comando `mapReduce` en una colección. `mapReduce` toma una función para realizar el mapeo, una funciona para realizar la reducción y una directiva para el resultado. En nuestro terminal podemos pasarle una función de Javascript. En muchas librerías le daremos un `string` con nuestras funciones. Primero vamos a crear nuestro set de data para analizar:

```
db.hits.insert({resource: 'index', date: new Date(2010, 0, 20, 4, 30)});
db.hits.insert({resource: 'index', date: new Date(2010, 0, 20, 5, 30)});
db.hits.insert({resource: 'about', date: new Date(2010, 0, 20, 6, 0)});
db.hits.insert({resource: 'index', date: new Date(2010, 0, 20, 7, 0)});
db.hits.insert({resource: 'about', date: new Date(2010, 0, 21, 8, 0)});
db.hits.insert({resource: 'about', date: new Date(2010, 0, 21, 8, 30)});
db.hits.insert({resource: 'index', date: new Date(2010, 0, 21, 8, 30)});
db.hits.insert({resource: 'about', date: new Date(2010, 0, 21, 9, 0)});
db.hits.insert({resource: 'index', date: new Date(2010, 0, 21, 9, 30)});
db.hits.insert({resource: 'index', date: new Date(2010, 0, 22, 5, 0)});
```

Ahora podemos crear nuestras funciones Map y Reduce. El terminal de MongoDB acepta comandos multi linea y veras `..` después de pulsar enter para indicar que esperamos mas texto.

```
var map = function() {
    var key = {resource: this.resource, 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};
};
```

Ahora podemos usar el comando `mapReduce` sobre la colección de `hits`:

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

Corriendo este comando podremos ver el resultado deseado. Cambiando `out` a `inline` significa que el resultado de `mapReduce` es inmediatamente transmitido a nosotros. Esto esta actualmente limitado a resultados que son menores de 16 megabytes. Pudiéramos especificar `{out: 'hit_stats'}` y tener el resultado guardado en la colección `hit_stats`

```
db.hits.mapReduce(map, reduce, {out: 'hit_stats'});
db.hit_stats.find();
```

Cuando realizamos esta acción perdemos toda la información almacenada en `hit_stats`. Si hiciéramos `{out: {merge: 'hi_stats'}}` las claves existentes serán remplazadas cuando los nuevos valores y lass nuevas llaves sera insertadas como documentos. Finalmente podemos `out` usando una función `reduce` para manejar casos mas avanzados.

El tercer parametro toma algunas opciones adicionales. Pudiéramos filtrar, ordenar y limitar los documentos que queremos analizar. Tambien podemos especificar una función `finalize` que sera ejecutada contra el resultado después de la función `reduce`

En Este Capitulo

Este es el primer capitulo donde cubrimos algo diferente. Si te hizo sentir incomodo, recuerda que siempre puedes usar otras funciones de MongoDB como la agregación, para escenarios mas sencillos. Al final ten en cuanta que MapReduce es una de las mejores funcionalidad de MongoDB. La clave para entender como escribir las funciones de Map y Reduce es visualizar y entender la manera como la data intermedia lucirá al salir del `map` y entrando al `reduce`

Capitulo 7 - Rendimiento y Herramientas

En este capitulo, veremos algunos tópicos sobre rendimiento así como algunas de las herramientas disponibles para los desarrolladores de MongoDB. No nos adentraremos mucho en estos tópicos, pero examinaremos los aspectos mas importantes de cada uno.

Indices

Al principio vimos los indices especiales de colecciones `system.indexes` que contienen información de todos los indices de nuestra base de datos. Los indices en MongoDB trabajan muy parecido a los indices en una base de datos relacional: ayudan a mejorar el rendimiento de las queries y ordenado. Los indices son creados via `ensureIndex`

```
db.unicorns.ensureIndex({name: 1});
```

Y borrados via `dropIndex`:

```
db.unicorns.dropIndex({name: 1});
```

Un indice unico puede ser creado pasandole un segundo parametro y configurando `unique` a `true`:

```
db.unicorns.ensureIndex({name: 1}, {unique: true});
```

Los indices pueden ser creados en campos embebidos (de nuevo, usando la notación de punto) y en campos de arreglos. Tambien podemos crear indices compuestos:

```
db.unicorns.ensureIndex({name: 1, vampires: -1});
```

El orden de los indices (1 para ascendiente, -1 para descendiente) no importa para un unico indice, pero puede tener un impacto por composición de indices cuando estas ordenando o usando condiciones de rango.

La pagina de indices tiene información adicional sobre los indices.

Explain

Para ver si tus queries están usando indices o no, puedes usar el metodo `explain` en un cursor:

```
db.unicorns.find().explain()
```

La salida nos dice que un `BasicCursor` fue usado (lo que implica no indexado), 12 objetos fueron escaneados, cuanto tiempo tomo, que indice, si alguno fue usado así como algunas otras piezas utiles de información.

Si cambiamos nuestro query para que use un indice, veremos que `BtreeCursor` fue usado, así como el index usado para realizar el request:

```
db.unicorns.find({name: 'Pilot'}).explain()
```

Escrituras Asíncronas

Anteriormente mencionamos que, por defecto, las escrituras en MongoDB son asíncronas. Esto puede resultar en ganancias de rendimiento al costo de perder datos durante un crash. Un efecto secundario de las escrituras asíncronas es que un error no es retornado cuando una inserción o una actualización viola una restricción unica. Para ser notificado acerca de una escritura fallida, debes llamar `db.getLastError()` después del ultimo error. Muchos drivers abstraen este detalle y proveen una manera de hacer una escritura *segura* - usualmente via un parametro extra.

Desafortunadamente, la terminal hace automaticamente Escrituras seguras, para que veamos este comportamiento facilmente en acción.

Sharding

MongoDB soporta auto-sharding. Sharding es un acercamiento hacia la escalabilidad, que separa tu datos a traves de multiples servidores. Una implementación nativa podría colocar todos los datos para usuarios con un nombre que comienza con A-M en el servidor 1 y el resto en el servidor 2. Las capacidades de MongoDB de sharding exceden tan simple algoritmo. el sharding es un topico que esta fuera del alcance de este libro, pero debes saber que existe y que debes considerarlo si tus necesidades se aplian hasta usar mas de un servidor.

Replicaicones

La replicaron en MongoDB trabajaba similarmente a como o hacen las bases de datos relacionales. Las escrituras son enviadas a un solo servidor, el servidor master, el cual luego sincroniza con sigo o con los demás servers, los esclavos. Puedes controlar si las lecturas suceden en el server esclavo o no, lo que puede ayudar a distribuir la carga al momento de leer los datos. Si el servidor master deja de responder, un servidor esclavo puede ser promovido para actuar como el nuevo master. De nuevo, la replicacion en MongoDB esta fuera del alcance de este libro.

Mientras la replicacion puede mejorar el desempeño (distribuyendo escrituras), su proposito principal es incrementar la confiabilidad. Combinando la replicacion con el sharding es un proposito comun. Por ejemplo, cada shard puede estar hecho de un servidor master y un esclavo. (Tecnicamente tambien necesitaras un arbitro en el caso de que dos servidores quieran ser masters. Pero un arbitro requiere solo pocos recursos y puede ser usado y puede ser usado por multiples shards)

Estadísticas

Puedes obtener estadísticas en una base de datos escribiendo `db.stats()`. La mayoría de la información trata con el tamaño de tu base de datos. Tambien puedes obtener estadísticas de una colección, digamos `unicorns`, escribiendo `db.unicorns.stats()`. De nuevo, la mayoría de esta información se relaciona al tamaño de la colección.

Interfaz Web

Incluida en la información mostrada al iniciarse MongoDB estaba un link a una herramienta de administración web (podrás verla si haces scroll en tu terminal hasta el punto donde iniciaste `mongodb`). Puedes acceder a esto, abriendo

tu navegador en http://localhost:28017/. Para obtener el mayor resultado de el, deberás agregar `rest=true` a tu configuración y reiniciar el proceso `mongod`. La interfaz web te da una gran cantidad de conocimiento acerca del estado actual del server.

Profiler

Puedes habilitar el profiler de MongoDB ejecutando:

```
db.setProfilingLevel(2);
```

Con el activado, puedes correr el comando:

```
db.unicorns.find({weight: {$gt: 600}});
```

Y luego examinados el profiler:

```
db.system.profile.find()
```

La salida nos dice que fue lo que se ejecuto y cuando, cuantos documentos fueron escaneados, y cuantos fueron retornados.

Puedes desabilitar el profiler LLAMANDO `setProfileLevel` de nuevo per cambiando el argumento a `0`. Otra opción es especificar 1 que solo le hará profile a los queries que tomen mas de 100 milisegundos. O, puedes especificar el tiempo minimo, en milisegundos, con un segundo parametro:

```
    // profile cualquier cosa que tome mas de 1 segundo
db.setProfilingLevel(1, 1000);
```

Respaldos y Restauraciones

En la carpeta `bin` de `MongoDB` se encuentra un ejecutable `mongodump`. Simplemente ejecutando `mongodump` se conectara a la base de datos y hará un respaldo de todas tus bases de datos a una sub carpeta `dump`. Puedes ejecutar `mongodump --help` para ver opciones adicionales. Las opciones comunes son `--db DBNAME` para respaldar una base de datos especifica y `--collection COLLECTIONAME` para respaldar una colección especifica.. Luego puedes usar el ejecutable `mongorestore`, localizado en la carpeta `bin`, para restaurar un respaldo hecho previamente. De nuevo, los tags `--db` y `--collection` pueden ser especificados para restaurar una base de datos o una colección especifica.

Por ejemplo, para respaldar nuestra colección `learn` a una carpeta `backup`, ejecutamos (esto es su mismo ejecutable que puedes correr en una ventana de tu terminal, no dentro de la consola misma de mongo):

```
mongodump --db learn --out backup
```

Para restaurar solo la colección `unicorns`, podemos hacer:

```
mongorestore --collection unicorns backup/learn/unicorns.bson
```

Vale resaltar que `mongoexport` y `mongoimport` son otros dos ejecutables que pueden ser usados para exportar e importar datos desde JSON o CSV. Por ejemplo, podemos obtener una salida en JSON haciendo:

```
mongoexport --db learn -collection unicorns
```

Y una salida CSV haciendo:

```
mongoexport --db learn -collection unicorns --csv -fields name,weight,vampires
```

Nota que `mongoexport` y `mongoimport` no siempre podrá representar tus datos. Solo `mongodump` y `mongorestore` debería ser usado para respaldos en si.

En este capitulo

En este capitulo vimos varios comandos, herramientas y detalles de rendimiento al usar MongoDB. No hemos tocado todo, pero hemos visto los mas comunes. El indexado en MongoDB es similar al indexado en las bases de datos relacionales, así como muchas de las herramientas. Sin embargo, con MongoDB, muchas de estas son justas y simples de usar.

Conclusion

Debes tener suficiente información para comenzar a usar MongoDB en un proyecto real. Hay mas sobre MongoDB de lo que se ha cubierto, pero tu próxima prioridad debe ser juntar lo que has aprendido, y familiarizarte con el driver que usaras. El sitio de MongoDB tiene mucha información útil. El grupo oficial de MongoDB es un buen lugar para aclarar tus dudas.

NoSQL nació no solo como una necesidad, si no tambien como un interes de probar nuevas formas de acercamiento. Es de conocimiento que nuestro campo esta avanzando y que si no probamos cosas nuevas, y fallamos, nunca triunfaremos. Esto es, a mi parecer, una buena forma de llevar nuestras vidas profesionales.