idea
ilusión
imaginación
imparcialidad
impecable
intercambiar
ímpetu
importante
impresionante

Grupo iD

innovación
inculcar
indagar
inédito
ineludible
inestimable
informado
ingenio
iniciativa
inquietud
inspiración
intachable
integridad
intelectual
intensidad
identidad

DESAROLLA TU "i"

intuición
inversión
invencible
inventar
investigación

EMERGYA

MONGO DB

ÍNDICE

Pablo Campos

Material consulta: MONGO DB

Como consultora especializada en Formación y RRHH, en Grupo iD perseguimos **mejorar la competitividad del capital humano** de nuestros clientes. La implementación de las acciones formativas será eminentemente práctica y conectada con la realidad de la compañía.

Por tanto, EMERGYA y Grupo iD asumen los siguientes objetivos:

- Aprender qué es NoSQL y en particular MongoDB
- Saber las diferencias con otras bases de datos
- Saber realizar las operaciones más habituales
- Aprender algunas técnicas de optimización
- Aprender algunas operaciones avanzadas y la administración de usuarios y roles

Contenidos

MÓD. 1 INTRODUCCIÓN

- Introducción a NoSQL
- Introducción a MongoDB
- Diferencias con una base de datos relacional.
- Creación de nuestra primera base de datos

MÓD. 2 CRUD

- Crear documentos
- Recuperar documentos
- Filtrar documentos
- Actualizar documentos
- Eliminar documentos

MÓD. 3 DISEÑO DE ESTRUCTURA DE DATOS

- Introducción
- Referencias
- Documentos Embebidos
- Relaciones

MÓD. 4 OPTIMIZACIÓN CON INDICES

- Introducción a indices.
- Creación de Indices.
- Uso de indices.
- Filtros de indices.

MÓD. 5 USUARIOS, ROLES Y FUNCIONES AVANZADAS

- Operaciones de agregación.
- Usuarios y roles.
- Expresiones regulares.

Duración

Modalidad

Presencial (in-house)

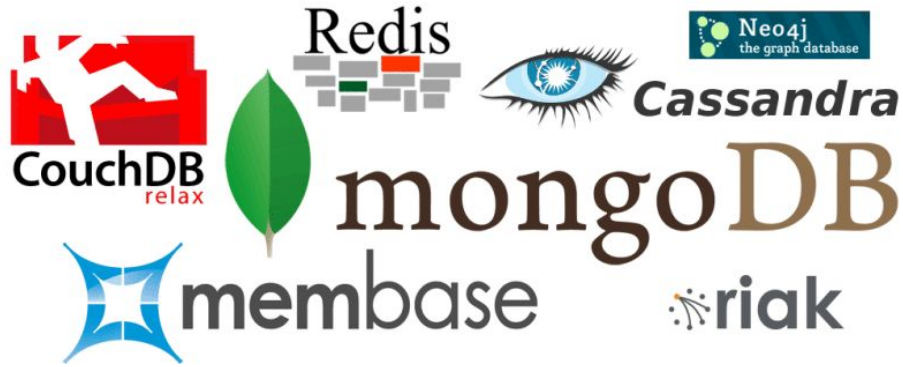
Nota: Los días y horarios definitivos se acordarán con el cliente una vez aprobada la propuesta

Material consulta: MONGO DB

MÓD. 1 INTRODUCCIÓN

NoSQL

- No solo SQL
- Escalabilidad
- Versatilidad
- Desarrollo ágil
- Velocidad
- Orientada a documentos



NoSQL

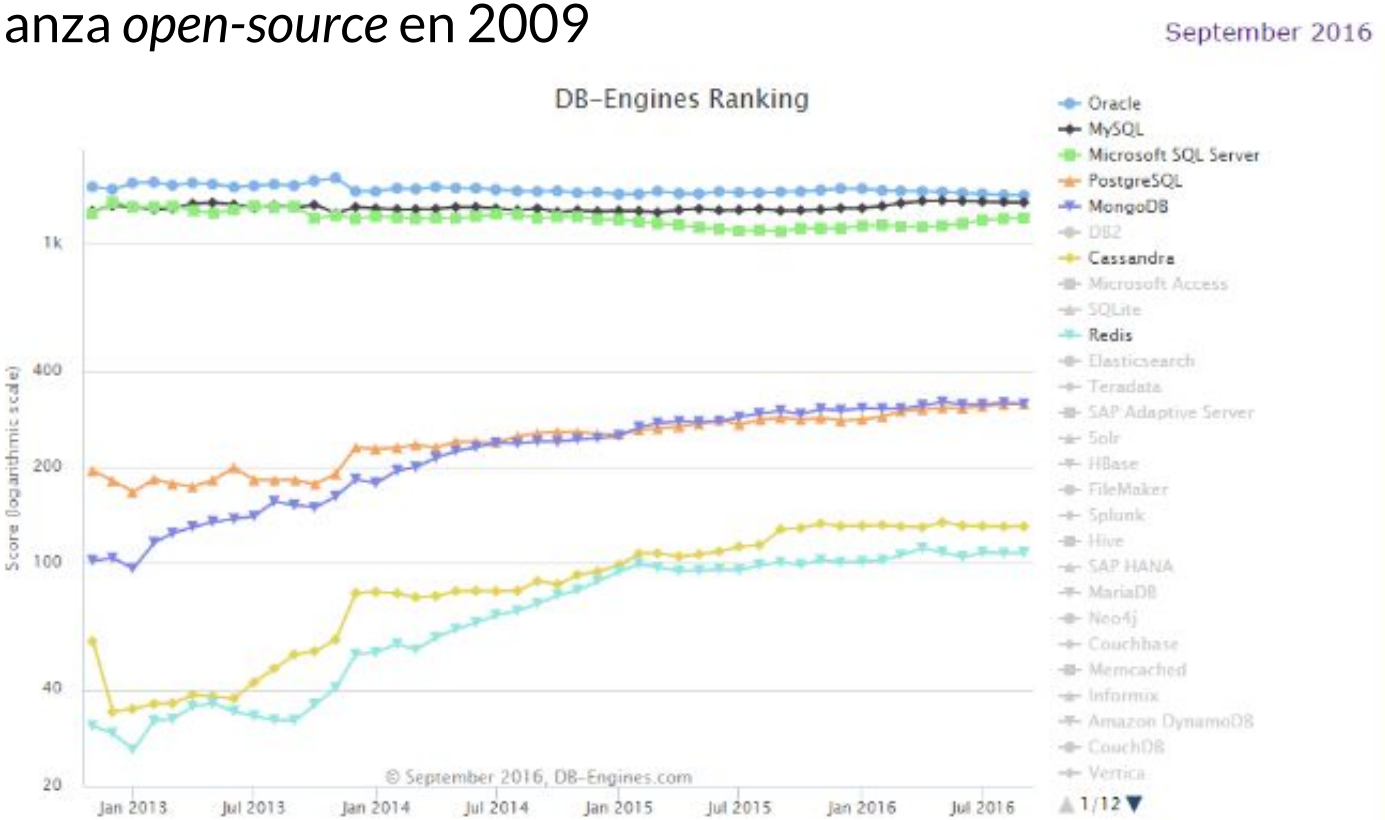
- Base de datos
- Colección
- Documento
- Campos

SQL

- Base de datos
- Tabla
- Fila
- Columna

MongoDB

- 10gen detecta la necesidad 2007
- Se lanza *open-source* en 2009



Material consulta: MONGO DB

MongoDB

- Alto rendimiento
- Lenguaje de consultas avanzado
- Alta disponibilidad
- Escalabilidad horizontal
- Consola en JS

MongoDB

```
sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv EA312927

echo "deb http://repo.mongodb.org/apt/ubuntu trusty/mongodb-org/3.2 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-3.2.list

sudo apt-get update

sudo apt-get install -y mongodb-org

sudo service mongod start
```

MongoDB

Comandos interesantes:

```
use DATABASE_NAME
show dbs
db.dropDatabase()
db.createCollection(name, options)
show collections
db.COLLECTION_NAME.drop()
```

MongoDB - Práctica

1. Instala mongodb y familiarízate con la consola
2. Comprueba las bases de datos existentes
3. Crea una base de datos para el curso
4. Comprueba que se ha creado correctamente
5. Crea dos colecciones en la base de datos

MÓD. 2 CRUD

CRUD

Material consulta: MONGO DB

Documentos

- No tienen estructura definida
- Formato JSON
- Se guardan en colecciones
- Son las “filas” de SQL
- key – value
- La “Clave primaria” se genera por defecto en la clave _id

Documentos

```
{
   _id: ObjectId(7df78ad8902c)
   title: 'Introduccion a MongoDB',
   description: 'MongoDB es una base de datos no sql',
   by: 'App Software Factory',
   url: 'http://www.www.app-softwarefactory.com',
   tags: ['mongodb', 'database', 'NoSQL'],
   likes: 100,
   comments: [
      {
         user:'usuario1',
         message: 'Mi primer comentario',
         dateCreated: new Date(2011,1,20,2,15),
         like: 0
      },
      {
         user:'usuario2',
         message: 'Mi segundo comentario',
         dateCreated: new Date(2011,1,25,7,45),
         like: 5
      }
   ]
}
```

Crear

- Los documentos se insertan en colecciones
- No tienen un modelo definido

```
db.articulos.insert({
  título: ‘Aprendiendo MongoDB',
    Contenido : ‘En este curso vamos a aprender las
funcionalidades básicas de mongoDB',
  tags : [‘eLearn' , 'MongoDB']
});
```

Leer

- Podemos filtrar y aplicar operaciones post-query

```
db.articulos.find();

db.artículos.find({
    título:‘Aprendiendo MongoDB’,
})

db.artículos.find().sort({campo:1})        *1 ó -1 definen el
orden ascendente o descendente

db.artículos.find.limit(5)
```

Leer

- Podemos usar proyecciones para que mongo no devuelva todos los campos de un resultado, si no solo aquellos que nos interesan
- _id siempre se devuelve salvo que especifiquemos lo contrario marcándolo a 0
- Se marcan a 1 los campos que queremos que nos sean devueltos

```
db.artículos.find({},{“titulo”:1,_id:0})
```

Eliminar

- Se pueden eliminar documentos o colecciones

```
db.articulos.remove({
    título:‘Aprendiendo MongoDB’,
})

db.artículos.drop()
```

Actualizar

- La actualización se realiza según el siguiente formato
- La actualización permite distintos operandores
- upsert: Si no hay resultado al aplicar el filtro, se creará un nuevo documento
- multi: Si el filtro devuelve más de un resultado, los cambios se aplican a todos

```
db.coleccion.update(
    filtro,
    cambio,
    {
        upsert: booleano,
        multi:  booleano
    }
);
```

Actualizar

- Modificamos el artículo que publicamos anteriormente

```
db.artículos.update({
     título:‘Aprendiendo MongoDB’,
},{
    título: ‘Aprendiendo MongoDB’,
    Contenido : ‘En este curso vamos a aprender las
funcionalidades básicas y algunas más avanzadas de mongoDB’,
    tags : [‘eLearn’ , ’MongoDB’]

});
```

Actualizar

- Operadores básicos: $inc, $rename, $set, $unset
- $inc -> Incrementa el valor de una variable numérica
- $rename -> Renombra el nombre de una clave
- $set -> Define la clave que se actualizará (si no existía previamente se creará) dentro del documento
- $unset -> Define la clave del documento que va a eliminarse

```
db.artículos.update({
     título:‘Aprendiendo MongoDB’,
},{
    $set: {comentarios:’23’}
});
```

Actualizar

- $pull -> Elimina los valores de un array que cumpla el filtro
- $pullAll -> Elimina los valores especificados de un array
- $pop -> Elimina el primer o último valor de un array
- $push -> Añade un elemento a un array
- $addToSet -> Añade elementos a un array si no existen
- $each -> Se emplea con $addToSet y $push para permitir la agregación múltiple de elementos a un array

```
db.artículos.update({
     título:‘Aprendiendo MongoDB’,
},{
    $pullAll: {tags: [‘eLearn’ , 'MongoDB’]}
});
```

Actualizar

```
db.artículos.update({
     título:‘Aprendiendo MongoDB',
},{
    $push: {tags :{$each:[‘eLearn' , 'MongoDB‘, ‘mongo’]}}
});
```

```
db.artículos.update({
     título:‘Aprendiendo MongoDB',
},{
    $push: {tags:’bases de datos’}
});
```

Método save()

- Puede servir para crear un documento si no existe o para actualizarlo si existe

```
db.artículos.save({
    título: 'Aprendiendo a usar save',
});

db.artículos.find({titulo:'Aprendiendo a usar save'})
Apuntamos el _id del documento

Db.artículos.save({
    _id: ObjectId(`Número apuntado`),
    título: 'Aprendiendo a usar save',
    Contenido : 'En este curso vamos a aprender las funcionalidades
básicas de mongoDB',
    tags : ['eLearn' , 'MongoDB']
})
```

MongoDB - Práctica

MÓD. 3 DISEÑO DE ESTRUCTURA DE DATOS

Estructuras de datos

- Se definen modelos para usar mongoDB a través de conectores desde otros frameworks (mongoose, pymongo…)
- Podemos usar referencias o embeber documentos
- Los tipos admitidos están limitados

Tipos de datos

- String
- Integer
- Boolean
- Double
- Timestamp
- Date
- Null
- Array
- Object
- ObjectId
- Binarios
- Javascript

¿Embeber o referenciar?

- Embeber -> Añadir documentos enteros dentro de documentos
- Referenciar -> Añadir _id dentro de documentos que referencien a otro documento

- Depende del tipo de relación y del crecimiento de las mismas

De SQL a NoSQL

Relaciones 1 a 1

Queremos relacionar un articulo del blog al autor del mismo:

```
Articulo = {
        _id: 'ID_DEL_ARTICULO_1',
        titulo: 'Relaciones 1 a n en SQL para NoSQL',
        Contenido: 'Vamos a hacer una relación 1 a n en mongoDB',
        tags: ['Relaciones', '1 a n'],
        autor: {
                usuario: 'pcampos',
                fechaDeRegistro: '10/10/10',
                articulosEscritos: '16'
        }
}
```

De SQL a NoSQL

Relaciones 1 a n

Queremos relacionar un articulo del blog a sus comentarios:

```
Articulo = {
        _id: 'ID_DEL_ARTICULO_1',
        titulo: 'Relaciones 1 a n en SQL para NoSQL',
        Contenido: 'Vamos a hacer una relación 1 a n en mongoDB',
        tags: ['Relaciones', '1 a n'],
        comentarios: [{
                usuario: 'pcampos',
                comentario: 'Gran blog!'
        },{
                usuario: 'smario',
                comentario: 'Mamma mia!'
        },{
                usuario: 'Paco',
                comentario: 'El link está roto!'
        }]
}
```

De SQL a NoSQL

Relaciones 1 a n

Queremos relacionar un articulo del blog a sus comentarios:

```
Articulo = {
        _id: 'ID_DEL_ARTICULO_1',
        titulo: 'Relaciones 1 a n en SQL para NoSQL',
        Contenido: 'Vamos a hacer una relación 1 a n en mongoDB',
        tags: ['Relaciones', '1 a n'],
        comentarios: [  'ID_COMENTARIO_1',
                        ' ID_COMENTARIO_2',
                        ' ID_COMENTARIO_3']
}
```

De SQL a NoSQL

Relaciones 1 a n

Queremos relacionar un articulo del blog a sus comentarios:

```
comentario = {
        _id: 'ID_DEL_COMENTARIO',
        idDelArticulo: 'ID_DEL_ARTICULO_1',
        usuario: 'pcampos',
        comentario: 'Buen articulo!'
}
```

De SQL a NoSQL

Relaciones n a n

Queremos relacionar varios artículos del blog a sus coautores:

```
Articulo = {
        _id: 'ID_DEL_ARTICULO_1',
        titulo: 'Relaciones 1 a n en SQL para NoSQL',
        Contenido: 'Vamos a hacer una relación 1 a n en mongoDB',
        tags: ['Relaciones', '1 a n'],
        autores: ['ID_DEL_AUTOR_1', 'ID_DEL_AUTOR_2', 'ID_DEL_AUTOR_3']
}
```

De SQL a NoSQL

Relaciones n a n

Queremos relacionar varios artículos del blog a sus coautores:

```
Autor = {
        _id: 'ID_DEL_AUTOR_1',
        usuario: 'pcampos,
        artículos: ['ID_DEL_ARTICULO_1', 'ID_DEL_ARTICULO_2']
}
```

Modelos de datos

- Las validaciones se definen por colección
- Pueden implementarse sobre nuevas colecciones al crearlas o sobre colecciones existentes.
- Dos opciones validationLevel y validationAction
  - Si el nivel se sitúa en estricto y ya existían documentos en la colección, todas las actualizaciones serán validadas.
  - Si el nivel se sitúa en moderado y existían documentos que no cumplían con los criterios de validación, la validación no aplicará en ellos
- La validación se da en la creación y en la actualización de documentos

Crecimiento de los documentos

- Cuando un documento tiene un array que crece, bien con subdocumentos internos o bien con otros elementos, su tamaño crece
- El crecimiento de documentos conlleva una relocalización en disco de la información
- El límite de tamaño de un documento es de 16MB
- La solución a la relocalización es la pre-localización

Pre-localización

- Cuando conocemos la estructura final que tendrá un documento cuya información va creciendo en tiempo real, podemos insertarlo en la base de datos entero con los valores a cero y simplemente ir actualizando los valores

Atomicidad

- Todas las operaciones en mongo son atómicas a nivel de documento
- Si es necesario modificar un documento y otro documento al que este haga referencia, las operaciones deben hacerse por separado
- El concepto de transacciones íntegras de SQL no existe
- Siempre que sea posible debe guardarse toda la información relevante y sujeta de ser modificada dentro de un documento, si la aplicación soporta actualizaciones no atómicas, la información puede separarse en dos documentos

Sharding

- El sharding permite el escalado horizontal
- Se balancea la carga de las operaciones y todos los procesos de lectura/escritura se realizan de manera mucho más eficiente
- La elección de la clave de shard *(shard key)* es un aspecto clave en el diseño del modelo de datos

Índices

- Cada índice necesita de al menos 8kB de espacio en disco
- La implementación o no de un índice en una colección depende en gran medida del ratio escritura-lectura
- Cada índice activo consume espacio en disco y en memoria con las consecuentes desventajas que conlleva, debemos estar seguros de que la memoria RAM de nuestra máquina soporta los índices creados

```
db.collection.totalIndexSize()
```

Número de colecciones

- El número de colecciones permitidas por mongo no está definido, lo que mongo limita es el número de namespaces (bases de datos, índices, colecciones)
- Cada colección, índice y base de datos, requiere de ciertos bytes de memoria para sus metadatos
- Solo hay que tener precaución, tener un gran número de colecciones no perjudica al rendimiento

Colección con muchos documentos pequeños

- Siempre que sea posible agrupar esos pequeños documentos debe hacerse
- Si la representación natural de los datos es un modelo pequeño, se debe mantener
- Debemos especificar explícitamente el valor de _id ya que si no, tendrá un tamaño de 12Bytes y puede ser significativo
- Debemos usar nombres cortos para los campos

Colecciones volátiles

- Índices TTL -> se explicarán en el módulo de índices
- Capped collections -> Colecciones con un tamaño máximo definido
  - Mantienen el orden de inserción
  - Funciona como una cola FIFO
  - Si van a actualizarse documentos en una capped collection, es necesario definir un índice para no escanear la colección entera
  - No pueden "*shardearse*"

```
db.createCollection("log", { capped : true, size : 5242880, max : 5000 } )
```

MongoDB - Práctica

MÓD. 4 OPTIMIZACIÓN CON ÍNDICES

¿Qué es un índice?

- Los índices son una estructura especial de datos que almacenan una pequeña porción de los datos de una manera fácil y rápida de recorrer.
- Un índice almacena el valor de un campo específico de cualquier colección.
- Los índices se almacenan en un espacio de memoria de rápido acceso.

Los índices apoyan la resolución eficiente de las queries.

¿Cómo usar los índices?

- Los índices son capaces de aumentar el rendimiento de resolución de queries en gran medida, pero abusar de estos mermará el rendimiento en memoria de la máquina que albergue los procesos
- Es necesario encontrar una solución de compromiso, definir nuestra **estrategia de indexamiento**
- La mejor estrategia es definir un índice para cada campo de nuestras colecciones que usemos recurrentemente en nuestra aplicación o servicio
- Si un índice no se usa a menudo, es mejor eliminarlo

¿Qué tipos de índices existen?

- _id -> Se genera automáticamente por parte de MongoDB para cualquier documento
- Sencillo -> Se definen sobre un único campo de cualquier documento
- Compuesto -> Se definen sobre varios campos de un documento
- Multillave -> Se emplean cuando el campo a indexar pertenece a un subdocumento dentro de un array del documento padre
- Geoespacial -> Se definen para indexar campos relativos a coordenadas de tipo GeoJSON
- Texto -> Se definen para indexar el contenido de los campos y poder buscar ágilmente dentro del contenido de la base de datos
- Hash -> Se emplean para almacenar los valores hasheados del campo a indexar

¿Qué propiedades tienen los índices?

- Pueden definirse como únicos para evitar duplicados con la propiedad *unique*
- Pueden ser dispersos (*sparse*) para permitir filtrar los resultados que se obtienen buscando por un índice y no mostrando aquellos que no tienen el campo
- Puede definirse una “fecha de caducidad” para datos de carácter volátil en nuestra base de datos con la propiedad *expireAfterSeconds*
  - Puede ponerse a 0 ó a un valor específico en segundos

Unique

- No puede aplicarse sobre una colección que ya contenga datos que violen la norma
- Pueden crearse índices únicos simples o compuestos
- Podrán insertarse documentos en la colección si no contienen el campo al que el índice se refiera

```
db.usuarios.createIndex({ “email” : 1 },{ unique : true })
```

```
db.usuarios.createIndex({ nombre : 1, apellidos : 1 },{ unique : true })
```

Sparse

```
{ “_id” : “1”, “user_id” : “antonio”, “horas” : 16}
{ “_id” : “2”, “user_id” : “juan”, “horas” : 12}
{ “_id” : “3”, “user_id” : “pablo”}

db.developers.ensureIndex( { horas : 1 }, { sparse : true } )

db.developers.find( { horas : { $gt : 13 } } )
```

¿Y si hacemos un find sin filtro?

¿Y si usamos hint()?

Índices parciales

- Se define una condición y solo se indexa la parte de la muestra que la cumpla
- Es incompatible con *sparse*

```
db.delanteros.createIndex({
    nombre : 1, equipo : 1 },{
    partialFilterExpression : { goles : { $gt : 10 } }})
```

¿Qué ocurre si filtramos para delanteros con más de 12 goles?
¿Qué ocurre si filtramos para delanteros con menos de 15 goles?
¿Cómo podríamos imitar el comportamiento de sparse?
¿Si buscamos delanteros del Betis con mas de 12 goles usamos el índice?
¿Si buscamos a Messi del Barcelona usamos el índice?

Índices parciales y únicos

- La condición de ser único solo debe cumplirse para aquellos documentos que cumplan con el filtro parcial

```
db.delanteros.createIndex({
    nombre : 1 },{
    unique : true,
    partialFilterExpression : { goles : { $gt : 10 } }})
```

```
{ “_id” : “1”, “nombre” : “Rubén”, “goles” : 16}
{ “_id” : “2”, “nombre” : “Lionel”, “goles” : 12}
{ “_id” : “3”, “nombre” : “Cristiano”, “goles” : 11}
```

¿Qué ocurre si introducimos a Rubén con 19 goles?
¿Qué ocurre si introducimos a Cristiano con 10 goles?
¿Qué ocurre si introducimos a Rubén sin campos goles?

TTL

```
db.eventsLog.ensureIndex( { “createdAt” : 1 }, { expireAfterSeconds
: 18000 } )

db.eventsLog.insert({
    “createdAt” : new Date(),
    “evento” : 4
})


db.eventsLog.ensureIndex( { “expireAt” : 1 }, { expireAfterSeconds :
0 } )

db.eventsLog.insert({
    “expireAt” : new Date(‘December 25, 2016 15:00:00’),
    “evento” : 4
})
```

db.runCommand({"collMod":"log_events","index":{keyPattern:{createdAt:1},expireAfterSeconds:5}})

Índices simples

- Los índices simples se definen sobre un único campo de un documento

```
Articulo = {
     _id: ‘ID_DEL_ARTICULO_1’,
     titulo: ‘Relaciones 1 a n en SQL para NoSQL’,
     Contenido: ‘Vamos a hacer una relación 1 a n en mongoDB’,
     tags: [‘Relaciones’, ‘1 a n’],
     comentarios: [{
          usuario: ‘pcampos’,
          comentario: ‘Gran blog!’
     },{
          usuario: ‘smario’,
          comentario: ‘Mamma mia!’
     },{
          usuario: ‘Paco’,
          comentario: ‘El link está roto!’
     }],
     shares: 19
}
```

Índices simples

- Para crear un índice usamos el comando *ensureIndex*

```
db.artículos.ensureIndex({ shares : 1 })

db.artículos.find({ shares : {$gt:10} })
```

¿Y si queremos indexar un campo embebido?
¿Y si queremos indexar un documento embebido?
¿Y si queremos ordenar (sort) usando índices?

Índices compuestos

- Para crear un índice usamos el comando *ensureIndex*
- Está limitado por mongoDB a 31 campos
- Para buscar y ordenar, el orden de los campos altera el rendimiento de la query

```
db.artículos.ensureIndex({ titulo : 1, contenido : 1 })

db.artículos.find({ titulo : “LOQUESEA” })
db.artículos.find({ titulo : “LOQUESEA”, contenido : “ELQUESEA” })
db.artículos.find({ contenido : “LOQUESEA”, titulo : “ELQUESEA” })
```

Índices compuestos

- Los prefijos se definen como partes del todo de un índice compuesto completo
- El orden en el que se defina el índice es relevante y de gran importancia para el rendimiento de la base de datos

```
db.artículos.ensureIndex({ stock : 1, udsVendidas : 1, precio : 1 })
Prefijo 1 -> { stock : 1}
Prefijo 2 -> { stock : 1, udsVendidas : 1}
```

Índices compuestos

- Usando índices compuestos pueden realizarse operaciones de ordenación usando prefijos completos o partes de prefijos
- Si se usa el prefijo completo, debe especificarse en el orden que se declaró para que sea usado

```
db.artículos.find().sort({ stock:1, udsVendidas:1 })
```

- Si se usa parte de un prefijo, será necesario realizar alguna operación de igualdad para las claves que precedan a la parte del prefijo

```
db.artículos.find({stock:20}).sort({udsVendidas:1})
```

Índices multillave

- Para crear un índice usamos el comando *ensureIndex*
- Puede construirse sobre arrays que contengan cadenas, números u otros documentos
- No es posible construir un índice compuesto de índices multillave, pero si un índice compuesto de uno simple y uno multillave

```
db.artículos.ensureIndex({ tags : 1 })

db.artículos.find({tags:'eLearn'})
```

Índices multillave

- Pueden realizarse queries para buscar un array específico pero el índice multillave no será usado en su totalidad

```
db.películas.find( { ratings: [ 5, 9 ] } )
```

- Pueden definirse índices multillave para campos contenidos en un objeto dentro de un array

```
{
  _id: ID,
  item: “NOMBRE”,
  stock: [
    { size: “TALLA”, color: “COLOR”, quantity: CANTIDAD },
    { size: “TALLA”, color: “COLOR”, quantity: CANTIDAD },
    { size: “TALLA”, color: “COLOR”, quantity: CANTIDAD },  ]
}
```

Índices multillave

¿Cómo definiríamos los índices para el ejemplo anterior por talla y cantidad?
¿Si buscamos por talla, se usarán los índices?
¿Si buscamos por talla y cantidad, se usarán los índices?
¿Si ordenamos por talla y calidad, se usarán los índices?
¿Si buscamos por cantidad y ordenamos por talla, se usarán los índices?
¿Si buscamos por talla y ordenamos por cantidad, se usarán los índices?

Índices de texto

- Facilitan y optimizan las búsquedas basadas en texto en gran medida
- Se declaran igual que el resto de índices con la particularidad de no ponerse a uno el valor, si no a *"text"*
- Pueden indexarse varios campos con un índice de texto
- **Solo puede declararse un índice text por colección**

```
db.artículos.createIndex({ titulo : "text" })

db.artículos.createIndex({ titulo : "text", contenido : "text" })
```

Índices de texto

- Para búsquedas de texto podemos definir un peso relativo a cada uno de los campos de texto que se indexan, de esta forma se ordenará la búsqueda según nuestro criterio.

```
db.artículos.createIndex({
    titulo : “text”, contenido : “text”
},{
    weights: { tirulo : 10, contenido : 5 },
    name: “TextIndex”
})
```

Índices de texto

- Si se quiere indexar todo el contenido de texto de una colección se emplea el *wildcard specifier*.
- Pueden declararse pesos para cualquier campo indexado

¿Por qué no declara siempre el wildcard specifier e indexar todo el texto?

```
db.collection.createIndex( { "$**": "text" }
```

Índices de texto

- La *tokenización* de las palabras está resuelta incluso con símbolos (desde la versión 3 de mongo)
- Mongo cuenta con varios lenguajes predefinidos, así como *stop words*
- Los índices de texto son *sparse* por defecto

```
db.artículos.createIndex({
    contenido : “text”
},{
    default_language: “spanish”
})

db.artículos.createIndex({
    contenido : “text”
},{
    language_override: “idioma”
})
```

Índices de texto

- La *tokenización* de las palabras está resuelta incluso con símbolos (desde la versión 3 de mongo)
- Mongo cuenta con varios lenguajes predefinidos, así como *stop words*
- Los índices de texto son *sparse* por defecto

```
{
  $text:
    {
      $search: <string>,
      $language: <string>,
      $caseSensitive: <boolean>,
      $diacriticSensitive: <boolean>
    }
}
```

Índices de texto

- Buscar una palabra
- Buscar por varias palabras
- Buscar una frase
- Buscar por una palabra excluyendo otra
- Buscar por idioma
- Buscar con sensibilidad a mayúsculas y minúsculas
- Buscar y recuperar la puntuación del resultado

Índices de texto

Si son tan útiles, ¿por qué no empleamos siempre la wildcard e indexamos todo?

- Son extensos
- Son lentos de construir
- Son costosos en memoria
- La búsqueda de frases es más efectiva sin índices

Algo más sobre índices

- Para no dejar sin servicio a la aplicación, cuando construir un índice vaya a ser muy costoso, hacerlo en background

```
db.COLECCION.createIndex( { CAMPO: 1}, {background: true} )
```

- Es posible analizar el rendimiento de nuestras queries y compararlas empleando el comando explain() con la opción *executionStats*

```
db.inventory.find(
   { quantity: { $gte: 100, $lte: 200 } }
).explain("executionStats")
```

- Debemos intentar usar las *covered queries* siempre que sea posible. Son queries dónde todos los campos de filtro son un índice y donde solo se proyecta en el resultado dichos campos.

Algo más sobre índices

- Es posible analizar el rendimiento de nuestras queries y compararlas empleando el comando explain() con la opción *executionStats*

```
{ "_id" : 1, "item" : "f1", type: "food", quantity: 500 }
{ "_id" : 2, "item" : "f2", type: "food", quantity: 100 }
{ "_id" : 3, "item" : "p1", type: "paper", quantity: 200 }
{ "_id" : 4, "item" : "p2", type: "paper", quantity: 150 }
{ "_id" : 5, "item" : "f3", type: "food", quantity: 300 }
{ "_id" : 6, "item" : "t1", type: "toys", quantity: 500 }
{ "_id" : 7, "item" : "a1", type: "apparel", quantity: 250 }
{ "_id" : 8, "item" : "a2", type: "apparel", quantity: 400 }
{ "_id" : 9, "item" : "t2", type: "toys", quantity: 50 }
{ "_id" : 10, "item" : "f4", type: "food", quantity: 75 }
```

¿Qué índices usarías para buscar tipo food de entre 100 y 300 de calidad?¿Por qué?¿Estabas en lo cierto?

Material consulta: MONGO DB

MongoDB - Práctica

MÓD. 5 USUARIOS, ROLES Y FUNCIONES AVANZADAS

Material consulta: MONGO DB

Usuarios y roles

- MongoDB es una **base de datos insegura por defecto**
- Debemos activar el control de accesos
  - Flag --auth
  - security.authorization
- Pueden definirse roles a los que se les permita realizar determinadas acciones de determinados recursos

Usuarios y roles

- Los privilegios de cada rol se definen explícitamente o se heredan
- Un privilegio se define como las acciones permitidas sobre un determinado recurso
- Un recurso es una base de datos, colección, grupo de colecciones o un cluster
- La definición de un rol puede incluir a uno o más roles distintos para especificar que se heredan sus privilegios
- Los privilegios de un rol pueden verse con el comando rolesInfo

Usuarios y roles

Definición de recursos para un rol:

1. Colección de una base de datos

   ```
   { db: “dispositivos”, collection: “wearable” }
   ```

2. Base de datos

   ```
   { db: “dispositivos”, collection: “” }
   ```

3. Colección recurrente en bases de datos

   ```
   { db: “”, collection: “users” }
   ```

4. Todas las colecciones y bases de datos (que no sean propias del sistema)

   ```
   { db: “”, collection: “” }
   ```

5. Cluster

   ```
   { cluster : true }
   ```

Usuarios y roles

Roles pre-definidos más empleados:

1. Usuario db
   `read - readWrite`
2. Administrador db
   `dbAdmin – userAdmin - dbOwner`
3. Cluster admin
   `clusterAdmin - clusterManager`
4. Todas las bases de datos
   `readWriteAnyDatabase – userAdminAnyDatabase - dbAdminAnyDatabase`
5. Root

Usuarios y roles – Creación de user

1. Creamos un usuario

```
use admin
db.createUser(
  {
    user: "myUserAdmin",
    pwd: "abc123",
    roles: [ { role: "userAdminAnyDatabase", db: "admin" } ]
  }
)
```

Usuarios y roles – Creación de user

2. Levantamos el servicio con la flag --auth

```
mongod --auth --port 27017 --dbpath /data/db1
```

3. Nos conectamos con el user/pass (desde fuera o dentro)

```
mongo --port 27017 -u "myUserAdmin" -p "abc123"
--authenticationDatabase "admin“
```

```
use admin
db.auth("myUserAdmin", "abc123" )
```

Usuarios y roles – Creación de rol

```
use admin
db.createRole(
   {
     role: "manageOpRole",
     privileges: [
         { resource: { cluster: true }, actions: [ "killop",
"inprog" ] },
         { resource: { db: "", collection: "" }, actions: [
"killCursors" ] }
     ],
     roles: []
   }
)
```

Usuarios y roles – Modificar permisos

```
db.revokeRolesFromUser(
    "reportsUser",
    [
      { role: "readWrite", db: "accounts" }
    ]
)

db.grantRolesToUser(
    "reportsUser",
    [
      { role: "read", db: "accounts" }
    ]
)

db.changeUserPassword("reporting", "SOh3TbYhxuLiW8ypJPxmt1oOfL")
```

Usuarios y roles – Seguridad

- KISS (Keep It Simple Stupid!)
- La seguridad es tan buena como lo sea el eslabón más débil del sistema
- Mantener los mínimos permisos posibles para cada usuario

Agregación

- La agregación permite realizar queries “anidadas” que incluyan operaciones
- Se desarrollan como la *tunelación* el Linux, siguiendo el orden especificado
- Si se usa MongoDB como motor de base de datos, es más eficiente emplear los métodos de agregación por *tunelación* que mapReduce
- Map-Reduce se ejecuta partido en dos fases
  - Mapeo de todos los documentos para *emitir* los objetos deseados a la siguiente fase
  - Fase de “*reducción*” que se encarga de operar con los datos recibidos de la primera fase
  - En ocasiones puede tener más fases

Agregación

- Operaciones de agregación simple
  - Count -> Devuelve un entero que indica el numero de documentos que cumplesn con la query realizada
  - Group -> Agrupa todos los documentos de una colección según el campo que se indique… Igual que GROUP BY
  - Distinct -> Devuelve los distintos campos únicos que existan para el que se especfique

```
db.COLECCION.count(QUERY)
db.COLECCION.distinct("CAMPO")
db.COLECCION.group{key, reduce, initial} ()
```

Agregación

- Mongo aprovecha los índices durante las operaciones de agregación para los operadores $match y $sort
- MongoDB optimiza solo las operaciones de agregación para que sean más eficientes cuando detecta un error común

Agregación

- $project -> Proyecta como salida los campos especificados de todos los obtenidos en la query anterior
- $match -> Filtra documentos según el criterio que se especifique, como en un find()
- $limit -> limita los resultado al numero de documentos especificado
- $skip -> Se salta el número de documentos indicados del resultado de la operación anterior
- $unwind -> Deconstruye un el array de un documento y construye muchos documentos que tienen un solo valor en el campo en el que había un array
- $group -> Agrupa el resultado en base al campo que se indique
- $sort -> Ordena los resultados

Agregación

- $lookup ->Permitie realizar operaciones JOIN SQL en base de datos mongoDB
- $out -> Debe ser el último operador y permite especificar una colección en la que guardar el resultado obtenido
- $and, $or y $not -> Operadores booleanos
- $cmp -> 0 si los dos valores son iguales y 1 en caso contrario
- $eq -> true si los valores son equivalentes
- $gt, $gte, $lt, $lte, $ne -> Mayor que, mayor que o igual…
- $in, $nin -> contenido en, no contenido en

https://docs.mongodb.com/manual/reference/operator/aggregation/#aggregation-pipeline-operator-reference

Agregación

- $sum -> Suma los valores numéricos del campo especificado (usarlo en group)
- $avg -> Valor medio del campo especificado (group)
- $first, $last, $max, $min
- Operadores aritméticos: $divide, $sqrt, $exp…
- Operadores de fechas: $week, $hour, $year…

https://docs.mongodb.com/manual/reference/operator/aggregation/#aggregation-pipeline-operator-reference

Agregación - Optimización

- $match siempre antes que $sort para minimizar el número de objetos que ordenar
- Cuando ponemos $skip antes de $limit, Mongo los reordena al revés (cambiando los valores) para permitir la fusión de los comandos $sort y $limit
- Cuando Mongo detecta que dos comandos pueden fusionarse, lo hace automáticamente de manera interna y transparente
- Cuando ponemos $project seguido de $skip o $limit, mongo adelanta a los segundos para que sea una proyección de menos resultados o para que si hay un $sort delante pueda fusionarse
- Mongo fusiona dos $limit seguidos
- Mongo fusiona dos $skip seguidos

Agregación - Optimización

- Cuando ponemos dos $match seguidos, es más eficaz realizar la misma operación en uno solo contando con el operador lógico $and
- Cuando ejecutemos un $lookup seguido de un $unwind podemos optimizarlo poniendo el $unwind en el propio $lookup

```
{
  $lookup: {
    from: "otherCollection",
    as: "resultingArray",
    localField: "x",
    foreignField: "y"
  }
},
{ $unwind: "$resultingArray"}
```

```
{
  $lookup: {
    from: "otherCollection",
    as: "resultingArray",
    localField: "x",
    foreignField: "y",
    unwinding: {
preserveNullAndEmptyArrays: false }
  }
}
```

Agregación – Map-Reduce

- Menos eficiente que *aggregate()*
- Se define la fase de mapero, la de reducción, la posible query y la posible colección de salida para el resultado

```
db.pedidos.mapReduce(
      function(){ emit( this._id, this.precio ) },
      function(key, values ){ return Array.sum( value ) },
      {
            query: { categoría : “limpieza” },
            out: “totales”
      }
)
```

Expresiones regulares

- Pueden emplearse para filtrar en cualquier campo que sea de texto
- Se usa el operador $regex

$regex : “/^ABC/i”

db.products.find({ sku: { $regex: /^ABC/i } })

- Opciones:
  - i -> Sensible a mayúsculas y minúsculas
  - m -> Para patrones que incluyan *anchors* (Inicio y final de línea)
  - x -> Para ignorar los espacios en blanco del patrón salvo que se escape el mismo específicamente
  - s -> Permite que el carácter “.” *matchee* con cualquier carácter

db.products.find({ description: { $regex: /^S/, $options: 'm' } })

MongoDB - Práctica

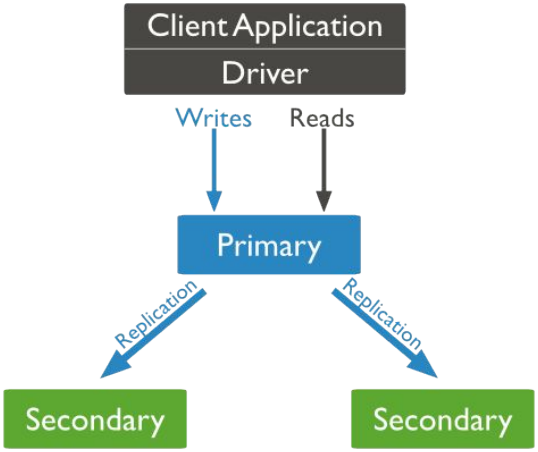
MÓD. 6 CONTENIDO EXTRA

Índice

1. Replica sets
2. Sharding con MongoDB
3. GUI software
4. Plataforma web MEAN stack

Replica set

- Provee “sólo” de redundancia y disponibilidad
- Nada de balanceo de carga
- Es un grupo de instancias mongod configuradas para replicar el contenido de determinadas bases de datos según las entradas del log de un servidor primario



Replica set

- Solo el nodo primario recibe las operaciones de escritura
- Los nodos secundarios replican las operaciones que realiza el nodo primario cuando este las anota en el log de operaciones de mongo
- Si un nodo primario cae, los nodos secundarios deciden quien juega el papel de primario desde ese momento
- Pueden definirse instancias de mongod como *árbitros* que no replican información, únicamente tienen voto a la hora de elegir un nuevo nodo primario
- Los nodos árbitro nunca cambian de estado hacia otro tipo de nodo

Replica set - Renovación de primarios

- Si el nodo primario del cluster se cae o pierde la comunicación durante 10 segundos, se le considera caído y se pasa a buscar un nuevo nodo primario
- El proceso de caída/reelección dura aproximadamente 1 minuto
  - 10-30 segundos caída del nodo primario y comunicación de la misma
  - 10-30 segundos elecciones

Replica set

- Por defecto se realizan las operaciones de lectura sobre el nodo primario
- Puede configurarse un cluster con un readConcern que especifique que se lea de nodos secundarios cuando sea necesario
  - primary
  - primaryPreferred
  - secondary
  - secondaryPreferred
  - nearest
- CUIDADO con la replicación asíncrona

```
db.getMongo().setReadPref('primaryPreferred')
```

Replica set

- Miembros con prioridad 0: sets de réplica secundarios que no pueden ser elegidos como primarios
  - Aceptan operaciones de lectura
  - Mantienen una copia de los datos
  - Votan en las elecciones
  - No pueden ser elegidos
  - No pueden lanzar unas elecciones
  - Pueden configurarse en stand-by

```
cfg = rs.conf()
cfg.members[2].priority = 0
rs.reconfig(cfg)
```

Replica set

- Miembros ocultos: sets de réplica secundarios que no pueden ser elegidos como primarios ni recibir operaciones de lectura
  - No aceptan operaciones de lectura
  - Mantienen una copia de los datos
  - Votan en las elecciones
  - No pueden ser elegidos
  - No pueden lanzar unas elecciones

```
cfg = rs.conf()
cfg.members[0].priority = 0
cfg.members[0].hidden = true
rs.reconfig(cfg)
```

Replica set

- Miembros “retrasados”: sets de réplica secundarios que guardan el estado del nodo primario con un retraso de tiempo definido
  - No aceptan operaciones de lectura
  - Mantienen una copia de los datos
  - Votan en las elecciones si se especifica
  - No pueden ser elegidos
  - No pueden lanzar unas elecciones
  - Deben ser *hidden* y *priority 0*
  - Su “retraso” debe ser más pequeño que la capacidad del opLog

```
{
      "_id"                 :                 <num>,
"host"              :              <hostname:port>,
     "priority"               :                 0,
"slaveDelay"            :              <seconds>,
     "hidden"                :                 true
}
```

Replica set

- El log de operaciones es una capped collection que mantiene un registro de las operaciones de la BBDD
- 5% de la capacidad libre del disco {990MB-50GB}
- CUIDADO con las actualizaciones “masivas”
- CUIDADO con hacer muchas pequeñas actualizaciones

```
rs.printReplicationInfo()
```

Replica set

- Sincronización inicial
  - Clona todas las bases de datos y construye los índices _id
  - Refleja los nuevos cambios del opLog a la nueva instancia
  - Construye el resto de índices
  - La instancia pasa a ser una instancia secundaria
- Se sincronizan las réplicas por *batches*
- Los miembros secundarios que tengan voto nunca copian de un miembro que no tenga voto
- Los miembros secundarios nunca copian de un miembro “retrasado”
- Pueden darse rollbacks de miembros primarios que reconectan tras una caída

Replica set

- Un set de réplicas sólo puede tener 7 miembros con voto
- Un set de réplicas sólo puede tener 50 miembros en total
- SIEMPRE es recomendable configurar un número impar de miembros con voto para evitar los empates
  - En caso de tener un número par de votantes, mejor añadir un miembro árbitro (No requiere mucho espacio)
- Considerar la tolerancia al fallo
  - Diferencia entre el número de miembros y el número de votos requeridos para elegir un primario
- Balancea la carga de lecturas si el número de operaciones es alto

¿Qué posibilidades hay para montar un set de 3 miembros?

Sharding

- Método que permite distribuir datos a través de múltiples máquinas
- Se emplea para arquitecturas y aplicaciones consolidadas, cuando el tamaño de la base de datos y el volumen de operaciones crece demasiado para mantenerlo en una sola máquina (CPU y RAM)
- El escalado vertical puede ser útil si la máquina que alberga la instancia mongoDB no era ya demasiado potente
- El escalado horizontal (sharding) permite dividir el sistema en varios servidores y balancear la carga de trabajo

Sharding

- Un cluster de instancias mongoDB fragmentado siempre tiene implicado a tres tipos de actores:
  - Fragmentos (*shards*): Cada fragmento contiene parte del total de datos que almacena la base de datos y puede despleagarse como un set de réplicas a su vez
  - Routers (*mongos*): Son la interfaz que une a los clientes con los distintos fragmentos
  - Servidores de configuración: Servidores que almacenan los metadatos necesarios para que el cluster opere con normalidad

Sharding

- Entendiendo el sharding con una baraja de cartas
  - 48 cartas - 4 palos
  - Cuatro jugadores y un repartidor
  - Cada jugador tendrá un máximo de 12 cartas y el repartidor mirará cada carta antes de repartirla

- Los jugadores son los shards
- Las cartas son documentos
- El repartidor es la instancia mongos (router)
- La shard key será el número de carta

Sharding

- El criterio que define la distribución de los documentos entre los distintos shards lo define la shard key
- La shard key es uno o varios campos que existen en TODOS los documentos de la colección
- La shard key se especifica antes de fragmentar la colección y no puede cambiarse después
- Para hacer un sharding de colecciones con datos ya insertados es imprescindible tener un índice definido cuyo primer campo sea la shard key
- La elección de la mejor shard key posible es imprescindible para el buen rendimiento del cluster
- Cada “rango” de shard keys ubicadas en un fragmento se denomina chunk

Sharding

- El sharding balancea la carga de lecturas/escrituras y distribuye el trabajo de la base de datos por todo el cluster
- La capacidad de almacenamiento de mongoDB se distribuye en el cluster y el crecimiento del mismo es muy fácilmente implementable
- Si un shard se cayera, las operaciones de lectura/escritura de los shards que sigan activos seguirán realizándose

Sharding

- El sharding requiere un estudio previo de necesidades para ser eficientes en la elección de la shard key así como la configuración del cluster
- Si se realiza una query a una colección fragmentada y se filtra por un campo que no esté incluido en la shard key (o en su índice compuesto), el mongos realizará una query broadcast a todos los shard
- La elección de la shard key es de vital importancia puesto que no puede modificarse una vez escogida

Sharding

- Para una misma base de datos pueden tenerse colecciones fragmentadas y no fragmentadas
- En cualquier caso será necesario acceder a través de ellas a través de un mongos, aunque no esté fragmentada

Sharding

- MongoDB admite dos estrategias de distribución de datos basado en shard keys
  - Hashed keys: La distribución por shards se realiza en base al hash del campo que se escoja
  - Ranged sharding: La distribución se realiza en base a rangos definidos en base al índice escogido

Sharding

- La arquitectura mínima recomendada para producción con sharding incluye:
  - 3 servidores de configuración como set de réplicas
  - Cada shard como un set de réplicas de 3 miembros cada uno
  - Un router mongos como mínimo

Sharding

- Cada base de datos que implemente sharding tiene un shard primario definido en el que se almacenan las colecciones no fragmentadas

```
db.runCommand({movePrimary:<databaseName>,to:<newPrimaryShard>})

sh.status()
```

Sharding

- Los servidores de configuración almacenan metadatos relativo a los chunks almacenados en cada shard así como la distribución de shard keys
- Cada sharded cluster requiere de su propio servidor de configuración, no puede definirse uno genérico
- Se escribe información en ellos cuando se produce una migración o una división de chunks
- Se lee de ellos cuando se inicia un nuevo mongos (por primera vez o restart) y cuando cambian sus metadatos
- Los metadatos se almacenan en la base de datos config

Sharding

- Se encarga de enrutar las queries de lectura/escritura al shard que corresponda
- Los router cachean los metadatos acerca de la distribución de los chunk sobre los shards para enrutar
- Primero determina los shard que deben recibir la query y después establece un cursor apuntando al resultado de cada query
- Operaciones como la agregación o la ordenación se ejecutan en el primary shard (u otro) antes de devolver el resultado completo

```
db.isMaster()
```

Sharding

- Se define una shard key con el siguiente comando:

```
sh.shardCollection( namespace, key )
```

- Imprescindible definir previamente un índice si ya existen datos insertados
- Para colecciones fragmentadas solo pueden ser índices únicos el _id y la propia shard key
- Pueden ser un índice simple, compuesto (que empiece con la shard key) o hasheado
- **Son inmutables**
- Tres parámetros clave para elegir clave: Cardinalidad, frecuencia y monotonía.

Sharding

- Si definimos la shard key con un índice hasheado
  - Ganamos en balanceo de carga de datos
  - Perdemos en la eficiencia de las queries
- Son la solución idónea para keys de gran cardinalidad o muy monótonos en su crecimiento (_id u ISODates)

```
sh.shardCollection( "database.collection", { <field> : "hashed" } )
```

Sharding

- Si definimos la shard key con un índice por rangos
  - Los datos se distribuyen de manera menos uniforme
  - Las queries son más óptimas
- Son óptimos para keys distribuidas uniformemente y con mucha cardinalidad

```
sh.shardCollection( "database.collection", { <shard key> } )
```

Transacciones en MongoDB

- Queremos transferir fondos de la cuenta A a la cuenta B
- Tenemos dos colecciones:
  - Cuentas
  - Transacciones

```
db.cuentas.insert(
                                                                    [
        { _id: "cuenta_A", balance: 1000, pendingTransactions: [] },
        { _id: "cuenta_B", balance: 1000, pendingTransactions: [] }
                                                                    ]
)
```

Transacciones en MongoDB

- Para cada transacción creamos un nuevo documento con los siguientes campos:
  - Fuente
  - Destino
  - Valor
  - Estado: Inicial, pendiente, aplicada, hecha, cancelando y cancelada
  - Última modificación

Transacciones en MongoDB

1. Insertamos la transacción que implica a ambas cuentas en estado inicial

```
db.transactions.insert(
     { _id: 1, source: "cuenta_A", destination: "cuenta_B", value:
100,    state:    "initial",    lastModified:    new    Date()    }
)
```

2. Obtenemos el documento para operar con él de manera más sencilla y actualizamos su estado a pendiente

```
var t = db.transactions.findOne( { state: "initial" } )
db.transactions.update(
                 {    _id:    t._id,    state:    "initial"    },
                                                                 {
                      $set:    {    state:    "pending"    },
                  $currentDate:    {    lastModified:    true    }
                                                                 }
)
```

Transacciones en MongoDB

3. Actualizamos ambas cuentas

```
db.accounts.update(
    { _id: t.source, pendingTransactions: { $ne: t._id } },
        { $inc: { balance: -t.value }, $push: {
pendingTransactions: t._id } }
)

db.accounts.update(
    { _id: t.destination, pendingTransactions: { $ne: t._id }
},
   { $inc: { balance: t.value }, $push: { pendingTransactions:
t._id } }
)
```

Transacciones en MongoDB

4. Actualizamos el estado de la transacción

```
db.transactions.update(
            {    _id:    t._id,    state:    "pending"    },
                                                          {
                    $set:    {    state:    "applied"    },
                $currentDate:   {   lastModified:   true   }
                                                          }
)
```

Transacciones en MongoDB

5. Actualizamos la lista transacciones pendientes en cada cuenta

```
db.accounts.update(
       {  _id:  t.source,  pendingTransactions:  t._id  },
         {  $pull:  {  pendingTransactions:  t._id  }  }
)

db.accounts.update(
      {  _id:  t.destination,  pendingTransactions:  t._id  },
         {  $pull:  {  pendingTransactions:  t._id  }  }
)
```

Transacciones en MongoDB

6. Actualizamos el estado de la transacción a hecho

```
db.transactions.update(
            {    _id:    t._id,    state:    "applied"    },
                                                          {
                      $set:    {    state:    "done"    },
                $currentDate:   {   lastModified:   true   }
                                                          }
)
```

URLDIGA


Grupo iD
´Ahora viene cuando
aplaudes y tuiteas lo bueno
que ha sido el curso ☺
#hashtagcurso
@GrupoiDinfo
tutoria@grupoid.es
ENVIAR EMAIL →