NoSQL

Sesión 3 - **MongoDB Avanzado**

Índice

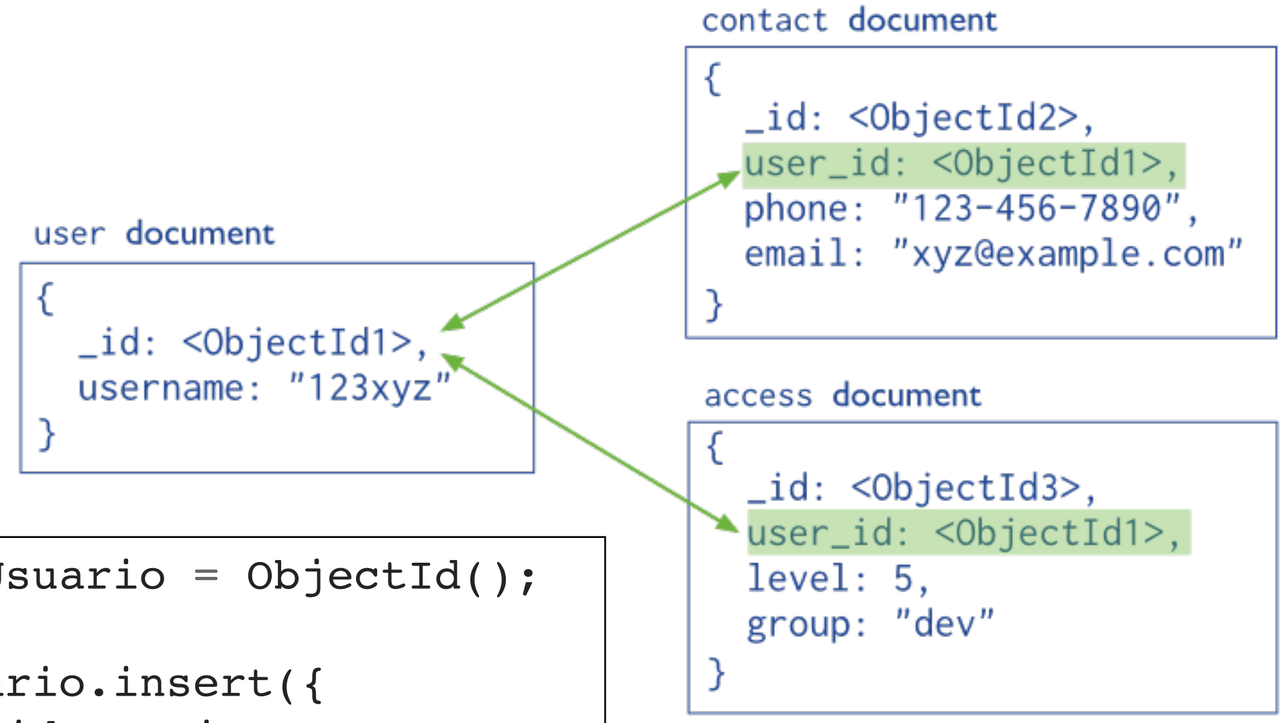
3.1 Diseño del Esquema

- Enfoque diferente al relacional
- No 3FN → tendencia a denormalizar
- *MongoDB* no soporta transacciones
  - Asegura que las operaciones son atómicas
  - Solución:
    1. Restructurar el código para que toda la información esté contenida en un único documento.
    2. Implementar un sistema de bloqueo por software (semáforo, etc…).
    3. Tolerar un grado de inconsistencia en el sistema.

- Denormalizar los datos para minimizar la redundancia pero facilitando que mediante operaciones atómicas se mantenga la integridad de los datos

Referencias Manuales

- Almacenar el campo `_id` como clave ajena
- La aplicación realiza una 2ª consulta para obtener los datos relacionados.
- Son sencillas y suficientes para la mayoría de casos de uso




```
var idUsuario = ObjectId();

db.usuario.insert({
  _id: idUsuario,
  nombre: "123xyz"
});

db.contacto.insert({
  usuario_id: idUsuario,
  telefono:  "123-456-7890",
  email: "xyz@ejemplo.com"
});
```

DBRef

- Objeto que representa una referencia de un documento a otro mediante el valor del campo `_id`, el nombre de la colección y, opcionalmente, el nombre de la base de datos
- `{ "$ref" : <nombreColeccion>, "$id" : <valorCampo_id>, "$db" : <nombreBaseDatos> }`
- las *DBRef* permite referenciar documentos localizados en diferentes colecciones.
- En *Java*, mediante la clase `DBRef`

```
db.contacto.insert({
  usuario_id: new DBRef("usuario", idUsuario),
  telefono:  “123 456 7890",
  email: "xyz@ejemplo.com"
});
```

Datos Embebidos

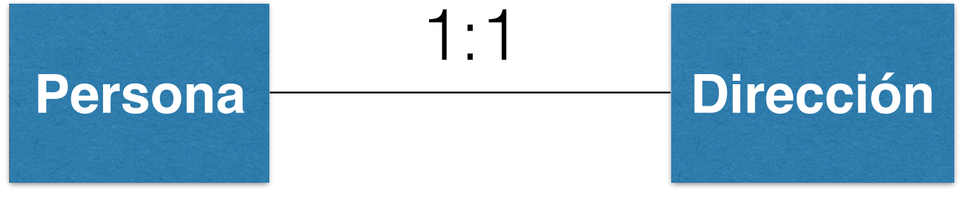
- Mediante sub-documentos
- Dentro de un atributo o un array
- Permite obtener todos los datos con un acceso
- Se recomienda su uso en relaciones:
  - contiene
  - uno a uno
  - uno a pocos

⚠️ Un documento BSON puede contener un máximo de 16MB

- Si un documento crece mucho → usar referencias o *GridFS*

Relaciones - 1:1

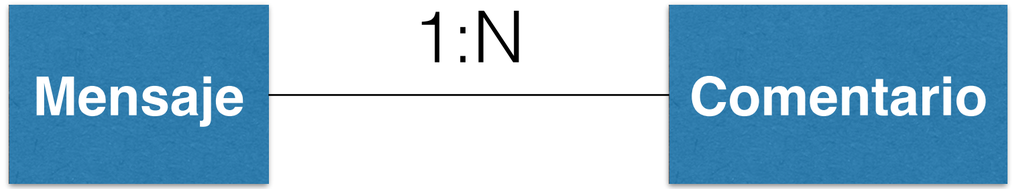
- Embeber un documento dentro de otro
- Motivos para no embeber:
  - Frecuencia de acceso.
    - Si a uno de ellos se accede muy poco
      - Al separarlos → se libera memoria
  - Tamaño de los elementos.
    - Si hay uno que es mucho más grande que el otro
    - O uno lo modificamos muchas más veces que el otro




```
{
  nombre: "Aitor",
  edad: 37,
  direccion: {
    calle: "Mayor",
    ciudad: "Elx"
  }
}
```

Relaciones 1:N - *1 a pocos*

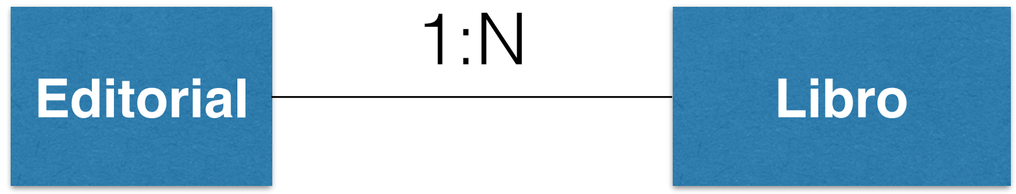
- Embeber los datos
- Crear un array dentro de la entidad 1
  - El Mensaje contiene un array de Comentario




```
{
  titulo: "La broma asesina",
  url: "http://es.wikipedia.org/wiki/Batman:_The_Killing_Joke",
  text: "La dualidad de Batman y Joker",
  comentarios: [
    {
      autor: "Bruce Wayne",
      fecha: ISODate("2015-04-01T09:31:32Z"),
      comentario: "A mi me encantó"
    }, {
      autor: "Bruno Díaz",
      fecha: ISODate("2015-04-03T10:07:28Z"),
      comentario: "El mejor"
    }
  ]
}
```

Relaciones 1:N - *1 a muchos*

- Referencia de N a 1
  - Igual que clave ajena
- Restricción 16MB BSON
- Se pueden emplear documentos embebidos cuando la información que interesa es la que contiene en un momento determinado
  - Productos (pvp) de un Pedido
  - Dirección (de envío) de un Cliente




```
{
  _id: 1,
  nombre: "O'Reilly",
  pais: "EE.UU."
}
```

Editorial

```
{
  _id: 1234,
  titulo: "MongoDB: The Definitive Guide",
  autor: [ "Kristina Chodorow", "Mike Dirolf" ],
  numPaginas: 216,
  editorial_id: 1,
}
{
  _id: 1235,
  titulo: "50 Tips and Tricks for MongoDB Developer",
  autor: "Kristina Chodorow",
  numPaginas: 68,
  editorial_id: 1,
}
```

Libro

N:M

- Suelen ser relaciones pocos a pocos
- 3 posibilidades
  1. Enfoque relacional con colección intermedia
     - Desaconsejado → 3 consultas
  2. Dos documentos, cada uno con un array que contenga los ids del otro documento (*2 Way Embedding*).
     - Vigilar la inconsistencia de datos
  3. Embeber un documento dentro de otro (*One Way Embedding*)
     - No se recomienda si alguno de los documentos puede crecer mucho
     - Revisar si un documento depende de otro para su creación


```
{
  _id: 1,
  titulo: "La historia interminable",
  anyo: 1979,
  autores: [1]
},{
  _id: 2,
  titulo: "Momo",
  anyo: 1973,
  autores: [1]
}
```

```
{
  _id: 1,
  nombre: "Michael Ende",
  pais: "Alemania",
  libros: [1,2]
}
```

2 way embedding

Consejos de Rendimiento I

- Si se realizan más lecturas que escrituras → denormalizar los datos para usar **datos embebidos** y así con sólo una lectura obtener más información.
- Si se realizan muchas inserciones y sobretodo actualizaciones, será conveniente usar referencias con dos documentos.
- El mayor beneficio de embeber documentos es el rendimiento, sobretodo el de lectura.
- El acceso a disco es la parte más lenta, pero una vez la aguja se ha colocado en el sector adecuado, la información se obtiene muy rápidamente (alto ancho de banda).
- Si toda la información a recuperar esta almacenada de manera secuencial favorece que el rendimiento de lectura sea muy alto

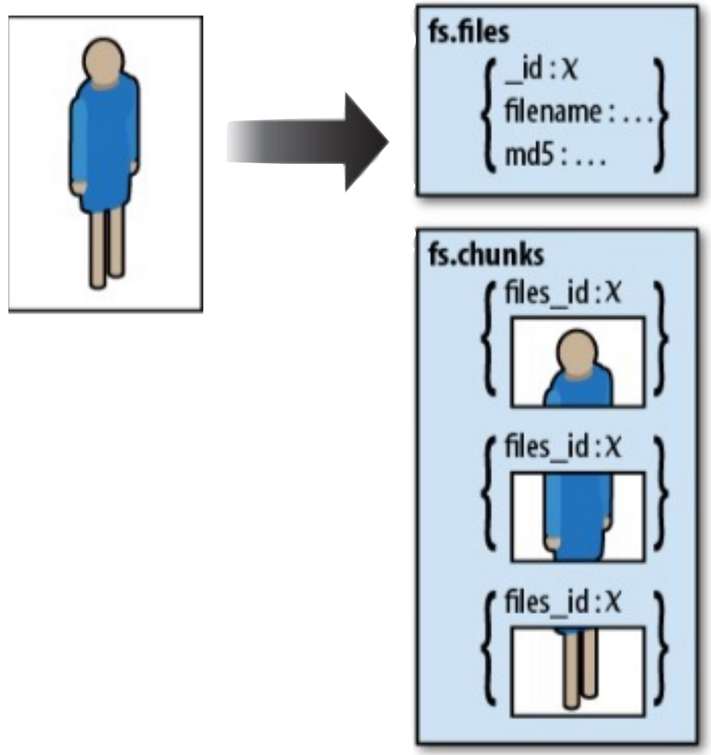
  Sólo se hace un acceso a la BBDD.

Consejos de Rendimiento II

- Planteamiento inicial para modelar los datos es basarse en **unidades de aplicación**
  - Petición al *backend* → click de un botón, carga de los datos para un gráfico.
  - Cada unidad de aplicación se debe poder conseguir con una única consulta → datos embebidos.
- Si necesitamos consistencia de datos → normalizar y usar referencias.
  - Puede que un sistema NoSQL no sea la elección correcta
  - Necesitemos dos o más lecturas para obtener la información deseada.
- Si la consistencia es secundaria, duplicar los datos (pero de manera limitada)
  - El espacio en disco es más barato que el tiempo de computación

3.2 GridFS

- `size(BSON)` < 16MB
- *GridFS* divide el fichero en partes/trozos (*chunks*)
- Almacena cada uno de estos trozos en un documento separado
  - Por defecto, se limita el tamaño de cada trozo a 256KB.
- Cuando se consulta un almacén *GridFS* por un fichero, el *driver* unirá los trozos tal como necesite.
- Permite consultas sobre ficheros almacenados con GridFS.
- También acceder a información de secciones arbitrarias de los ficheros, por ejemplo, saltar a la mitad de un archivo de sonido o video.



`mongofiles`

- Comando que permite interactuar con los archivos almacenados desde la consola.
- Útil para explorar y probar los archivos almacenados
- Recibe parámetro con las opciones
  - `list` → muestra todos los archivos

```
$ mongofiles list
```

  - `put` → inserta un archivo en *GridFS*

```
$ mongofiles put video.mp4
connected to: 127.0.0.1
added file: { _id: ObjectId('550957b83f627a4bb7f28bc8'), filename: "video.mp4",
chunkSize: 261120, uploadDate: new Date(1426675642227), md5:
"b7d51c0c83ef61ccf69f223eded44797", length: 39380552 }
done!
```

- Otras operaciones: `search`, `delete`, `get`

```
$ mongofiles list
connected to: 127.0.0.1
video.mp4 39380552
```

Colecciones

- Se utilizan dos colecciones para almacenar los archivos:
  - La colección `chunks` almacena los trozos de los ficheros
  - La colección `files` almacena los metadatos de los ficheros.
- Se crean en el espacio de nombre `fs` → `fs.chunks` y `fs.files`

```
> db.fs.files.find()
{ "_id" : ObjectId("550957b83f627a4bb7f28bc8"), "filename" : "video.mp4", "chunkSize" :
261120, "uploadDate" : ISODate("2015-03-18T10:47:22.227Z"), "md5" :
"b7d51c0c83ef61ccf69f223eded44797", "length" : 39380552 }
```

```
> db.fs.chunks.find({}, {"data":0})
{ "_id" : ObjectId("550957b856eb8d804bc96fb8"), "files_id" :
ObjectId("550957b83f627a4bb7f28bc8"), "n" : 0 }
{ "_id" : ObjectId("550957b956eb8d804bc96fb9"), "files_id" :
ObjectId("550957b83f627a4bb7f28bc8"), "n" : 1 }
...
{ "_id" : ObjectId("550957ba56eb8d804bc9704e"), "files_id" :
ObjectId("550957b83f627a4bb7f28bc8"), "n" : 150 }
```

GridFS desde Java I

```
MongoClient client = new MongoClient();
DB db = client.getDB("expertojava");
FileInputStream inputStream = null;

GridFS videos = new GridFS(db);
try {
  inputStream = new FileInputStream("video.mp4");  // archivo a cargar
} catch (FileNotFoundException e) {
  System.err.println("No puedo abrir el fichero");
}

GridFSInputFile video = videos.createFile(inputStream, "video.mp4"); // nombre del archivo

BasicDBObject meta = new BasicDBObject("descripcion", "Prevención de riesgos laborales");
List<String> tags = new ArrayList<String>();
tags.add("Prevención");
tags.add("Ergonomía");
meta.append("tags", tags);

video.setMetaData(meta);
video.save();

System.out.println("Object ID: " + video.get("_id"));
```

GridFS desde Java II

```
MongoClient client = new MongoClient();
DB db = client.getDB("expertojava");

GridFS videos = new GridFS(db);

// Buscamos un fichero
GridFSDBFile gridFile = videos.findOne(new BasicDBObject("filename", "video.mp4"));

FileOutputStream outputStream = new FileOutputStream("video_copia.mp4");
gridFile.writeTo(outputStream);

// Buscamos varios ficheros
List<GridFSDBFile> ficheros = videos.find(new BasicDBObject("descripción", "Prueba"));

for (GridFSDBFile fichero: ficheros) {
  System.out.println(fichero.getFilename());
}
```

Casos de Uso

- Superar la restricción de los 16MB de los documentos BSON.
- Almacenar *metadatos* de los archivos de video/sonido y acceder a ellos desde aplicaciones ajenos al sistemas de archivos
- Replicar el contenido para ofrecer una alta disponibilidad.
- Cachear contenido generado por el usuario como grandes informes o datos estáticos que no suelen cambiar y que cuestan mucho de generar.

- Inconvenientes
  - Pérdida de rendimiento
    - crear una prueba de concepto en el sistema a desarrollar
  - No es posible la actualización atómica.
    - Si el contenido es inferior a 16 MB → usar directamente documentos BSON los cuales aceptan datos binarios.

3.3 Índices

- Estructura de datos que almacena información sobre los valor de determinados campos de los documentos de una colección.
- Permite recorrer los datos y ordenarlos de manera muy rápida
- Se utilizan tanto al buscar un documento como al ordenar los datos de una consulta.
- Todas las colecciones contienen un índice sobre el campo `_id`

```
> db.students.findOne()
{
  "_id" : 0,
  "name" : "aimee Zank",
  "scores" : [
    {
      "type" : "exam",
      "score" : 1.463179736705023
    }, {
      "type" : "quiz",
      "score" : 11.78273309957772
    }, {
      "type" : "homework",
      "score" : 6.676176060654615
    }, {
      "type" : "homework",
      "score" : 35.8740349954354
    }
  ]
}
```

Plan de Ejecución

```
> db.students.find({"name" : "Kaila Deibler"}).explain()
{
   "cursor" : "BasicCursor",
   "isMultiKey" : false,
   "n" : 2,
   "nscannedObjects" : 200,
   "nscanned" : 200,
   "nscannedObjectsAllPlans" : 200,
   "nscannedAllPlans" : 200,
   "scanAndOrder" : false,
   "indexOnly" : false,
   "nYields" : 0,
   "nChunkSkips" : 0,
   "millis" : 755,
   "indexBounds" : {

   },
   "server" : "MacBook-Air-de-Aitor.local:27017"
}
```

```
> db.students.find({_id:30}).explain()
{
   "cursor" : "BtreeCursor _id_",
   "isMultiKey" : false,
   "n" : 1,
   "nscannedObjects" : 1,
   "nscanned" : 1,
   "nscannedObjectsAllPlans" : 1,
   "nscannedAllPlans" : 1,
   "scanAndOrder" : false,
   "indexOnly" : false,
   "nYields" : 0,
   "nChunkSkips" : 0,
   "millis" : 3,
   "indexBounds" : {
      "start" : {"_id" : 30},
      "end" : {"_id" : 30}
   },
   "server" : "MacBook-Air-de-Aitor.local:27017"
}
```

Índices Simples

- **`ensureIndex`**`({atributo:orden})`

```
db.students.ensureIndex( {name:1} )
```

- Orden de los índices (`1` para ascendente, `-1` para descendente)
  - No importa para un índice sencillo
  - Si que tendrá un impacto en los índices compuestos cuando se utilizan para ordenar o con una condición de rango.

```
> db.students.find({"name" : "Kaila Deibler"}).explain()
{
  "cursor" : "BtreeCursor name_1",
  "isMultiKey" : false,
  "n" : 2,
  "nscannedObjects" : 2,
  "nscanned" : 2,
  "nscannedObjectsAllPlans" : 2,
  "nscannedAllPlans" : 2,
  "scanAndOrder" : false,
  "indexOnly" : false,
  "nYields" : 0,
  "nChunkSkips" : 0,
  "millis" : 757,
  "indexBounds" : {
    "name" : [
      [
      "Kaila Deibler",
      "Kaila Deibler"
      ]
    ]
  },
  "server" : "MacBook-Air-de-Aitor.local:27017",
  "filterSet" : false
}
```

Información de los índices

- Toda la información relativa a los índices creados se almacenan en la colección `system.indexes`
- Podemos obtener los índices de una determinada colección mediante el método `getIndexes()`.
- Para borrar un índice emplearemos el método `dropIndex(campo)`.

```
db.system.indexes.find() // muestra los índices existentes
db.students.getIndexes() // muestra los índices de la colección students
db.students.dropIndex( {"name":1} ) // borra el índice que existe sobre la propiedad name
```

Propiedades de los índices

- Se pasan como segundo parámetro
- `unique` → Sólo permiten valores únicos en una propiedad. No puede haber valores repetidos y una vez creado no permitirá insertar valores duplicados.

```
db.students.ensureIndex( {students_id:1}, {unique:1} )
```

- `dropDups` → Al crear un índice único, podemos eliminar los duplicados. No se sabe cuales han sido eliminados, no sigue ningún criterio establecido → sólo se recomienda su uso para casos excepcionales.

```
db.students.ensureIndex( {students_id:1}, {unique:1, dropDups:1} )
```

- `spare` → Si queremos añadir un índice sobre una propiedad que no aparece en todos los documentos, necesitamos crear un índice *spare*. Se crea para el conjunto de claves que tienen valor.

```
db.students.ensureIndex( {size:1}, {spare:1} )
```

Autoevaluación

```
> db.people.find()
{ "_id" : ObjectId("50a464fb0a9dfcc4f19d6271"), "nombre" : "Juan", "cargo" : "Técnico" }
{ "_id" : ObjectId("50a4650c0a9dfcc4f19d6272"), "nombre" : "Pedro", "cargo" : "CEO" }
{ "_id" : ObjectId("50a465280a9dfcc4f19d6273"), "nombre" : "Sandra" }
```

```
db.people.ensureIndex( {cargo:1}, {spare:1} )
```

```
db.people.find({cargo:null})
```

- ¿Qué documentos aparecerán y por qué?

1. Ningún documento, ya que la consulta utiliza el índice y no puede haber documentos cuyo *cargo* sea nulo
2. Ningún documento, ya que la consulta de `cargo:null` sólo encuentra documentos que de manera explicita tienen el *cargo* a nulo, independientemente del índice.
3. El documento de *Sandra*, ya que la consulta no utilizará el índice
4. Todos los documentos de la colección, ya que todos los documentos cumplen `cargo:null`
5. El documento de *Sandra*, ya que el comando `ensureIndex` no se ejecutará sobre este documento.

Índices Compuestos

- Se aplican sobre más de una propiedad de manera simultánea

```
db.students.ensureIndex({name:1,scores.type:1})
```

- El **orden** de los índices **importa**
- Los índices se usan con los subconjuntos por la izquierda (prefijos) de los índices compuestos.
  - Si creamos un índice sobre los campos A,B,C, el índice se va a utilizar para las búsquedas sobre A, sobre la dupla A,B y sobre el trio A,B,C.
- Si tenemos varios índices candidatos a la hora de ejecutar, el optimizador de consultas los usará en paralelo y se quedará con el resultado del primero que termine

Índices Multiclave

- Al indexar una propiedad que es un array se crea un índice multiclave para todos los valores del array de todos los documentos.
  - Hacen que las consultas sobre documentos embebidos funcionen tan rápido.

```
db.students.ensureIndex({"teachers":1})
db.students.find({"teachers":{"$all":[1,3]}})
```

- Se pueden crear índices tanto en propiedades básicas, como en propiedades internas de un array, mediante la notación de .:

```
db.students.ensureIndex({“adresses.phones":1})
```

- Sólo se pueden crear índices compuestos multiclave cuando sólo una de las propiedades del índice compuesto es un array → no puede haber dos propiedades array en un índice compuesto.

Rendimiento

- Por defecto, los índices se crean en *foreground* → al crear un índice se van a bloquear a todos los *writers*.
- Para crearlos en *background* (de 2 a 5 veces más lento), segundo parámetro `background`:

```
db.students.ensureIndex( { twitter: 1}, {background: true} )
```

  - Sólo se puede realizar de uno en uno, y aunque el servidor admita más peticiones, el *shell* donde se crea el índice queda bloqueado.

- Operadores que no utilizan los índices eficientemente son: `$where`, `$nin` y `$exists`.
- Al emplearlos en una consulta hay que tener en mente un posible cuelo de botella cuando el tamaño de los datos incremente.

Gestión de la memoria

- **Los índices tienen que caber en memoria.**
- Si están en disco, pese a ser algorítmicamente mejores que no tener, al ser más grandes que la RAM disponible, no se obtienen beneficios por la penalización de la paginación.

```
db.students.stats()  // obtiene estadísticas de la colección
db.students.totalIndexSize()  // obtiene el tamaño del índice (bytes)
```

- Mucho cuidado con los índices *multikeys* porque crecen mucho y si el documento tiene que moverse en disco, el cambio supone tener que cambiar todos los puntos de índice del array.
- Aunque sea más responsabilidad de un DBA, los desarrolladores debemos saber si el índice va a caber en memoria.

- Si no usamos un índice o al usarlo su rendimiento es peor, es mejor borrarlo → `dropIndex`

```
db.students.dropIndex('nombreDeIndice')
```

Hints

- Fuerzan el uso de un índice
- Se ejecutan sobre un cursor, pasándole un parámetro con el campo → `.hint({campo:1})`

```
db.people.find({nombre:"Aitor Medrano",twitter:"aitormedrano"}).hint({twitter:1}})
```

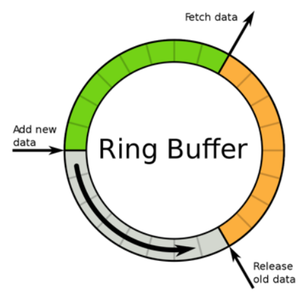
- Si queremos no usar índices, le pasaremos el operador `$natural`:

```
db.people.find({nombre:"Aitor Medrano",twitter:"aitormedrano"}).hint({$natural:1}})
```

- Los operadores `$gt`, `$lt`, `$ne`, … provocan un uso ineficiente de los índices, ya que la consulta tiene que recorrer toda la colección de índices.
- Si hacemos una consulta sobre varios atributos y en uno de ellos usamos `$gt`, `$lt` o similar, es mejor hacer un *hint* sobre el resto de atributos que tienen una selección directa.

```
db.grades.find({ score:{$gt:95, $lte:98}, type:"exam" }).hint('type')
```

3.4 Colecciones Limitadas (*capped collection*)



- Colección de tamaño fijo, donde se garantiza el orden natural de los datos
- Una vez se llena la colección, se eliminan los datos más antiguos, y los nuevos se añaden al final → similar a un buffer circular
- Se utilizan para logs y auto-guardado de información
- Rendimiento muy alto en inserciones.
- Se crean de manera explícita mediante el método `createCollection`, pasándole el tamaño (`size`) en bytes de la colección.

```
db.createCollection("auditoria", {capped:true, size:20480})
```

- Los documentos se pueden modifican, pero no pueden crecer en tamaño. Si sucede, la modificación fallará.
- Tampoco se pueden eliminar documentos de la colección → hay que borrar toda la colección (`drop`) y volver a crearla.

Número de elementos y Operaciones

- Podemos limitar el número de elementos que se pueden añadir a la colección mediante el parámetro `max` en la creación de la colección
  - Se ha de disponer de suficiente espacio en la colección para los elementos que queremos añadir.
  - Si la colección se llena antes de que el número de elementos se alcance, se eliminará el elemento más antiguo de la colección.

```
db.createCollection("auditoria", {capped:true, size:20480, max:100})
```

- `validate()` → cantidad de espacio utilizado por cada colección, ya sea limitada o no.

```
db.auditoria.validate()
```

- Al consultar los datos de una colección limitada los resultados aparecerán en el orden de inserción. Para obtenerlos en orden inverso → operador `$natural` al método `sort()`:

```
db.auditoria.find().sort({ $natural:-1 })
```

- Para averiguar si una colección es limitada → método `isCapped()`

3.5 *Profiling*

- La colección `db.system.profile` almacenará la auditoría de las consultas ejecutadas.
- Niveles de auditoría de consultas:
  - `0` (ninguna)
  - `1` (consultas lentas)
  - `2` (todas las consultas)
- A nivel de db → `db.setProfilingLevel(nivel)` o `db.setProfilingLevel(nivel, msMinimo)`

```
> db.setProfilingLevel(2)
```

- *MongoDB* automáticamente escribe en el log las consultas que tardan más de 100ms.
- Para indicar el nivel en el demonio → parámetros `--profile` y/o `--slowms`:

```
$ mongod --profile=1 --slowms=15
```

Estado y análisis

- Para obtener el estado del *profiling*:
  - `db.getProfilingLevel()`
  - `db.getProfilingStatus()`

```
> db.getProfilingLevel()
0
> db.getProfilingStatus()
{ "was" : 0, "slowms" : 100 }
```

- Sobre los datos auditados, podemos hacer consultas sobre la colección `system.profile` y filtrar por los campos mostrados:

```
db.system.profile.find({ millis : { $gt : 1000 } }).sort({ts : -1})
db.system.profile.find().limit(10).sort( { ts : -1 } ).pretty()
```

- Los campos más significativos son:
  - `op` → tipo de operación, ya sea `command`, `query`, `insert`, ...
  - `millis` → tiempo empleado en la operación
  - `ts` → *timestamp* de la operación

`mongotop`

- Similar a la herramienta `top` de UNIX
- Muestra el tiempo empleado por MongoDB en las diferentes colecciones, indicando tanto el tiempo empleado en lectura como en escrituras

```
MacBook-Air-de-Aitor:1314 aitormedrano$ mongotop 3
connected to: 127.0.0.1

                                ns       total        read       write
2014-04-06T09:51:16
                  local.system.users        0ms         0ms         0ms
                local.system.replset        0ms         0ms         0ms
                local.system.indexes        0ms         0ms         0ms
                   local.startup_log        0ms         0ms         0ms
                  jtech.system.users        0ms         0ms         0ms
                jtech.system.indexes        0ms         0ms         0ms
                         jtech.grades        0ms         0ms         0ms

                                ns       total        read       write
2014-04-06T09:51:19
                         jtech.grades        1ms         1ms         0ms
                  local.system.users        0ms         0ms         0ms
                local.system.replset        0ms         0ms         0ms
```

`mongostat`

- Muestra el número de operaciones por cada tipo que se realizan por segundo a nivel de servidor
- Instantánea de los que está haciendo el servidor

```
MacBook-Air-de-Aitor:1314 aitormedrano$ mongostat 5
connected to: 127.0.0.1
insert  query update delete getmore command flushes mapped  vsize    res faults  locked db
idx miss %     qr|qw   ar|aw  netIn netOut  conn       time
    *0     *0     *0     *0       0     0|0        0   160m  2.73g    35m      0  test:0.0%
          0      0|0    0|0     12b   598b     2  11:53:34
    *0     *0     *0     *0       0     1|0        0   160m  2.73g    35m      0  test:0.0%
          0      0|0    0|0    136b   739b     2  11:53:39
    *0     *0     *0     *0       0     0|0        0   160m  2.73g    35m      0 jtech:0.0%
          0      0|0    0|0     39b     2k     2  11:53:44
    *0     *0     *0     *0       0     0|0        0   160m  2.73g    35m      0     .:0.0%
          0      0|0    0|0     65b   653b     2  11:53:49
```

- `idx miss %` → índices perdidos → fuerzan paginación

¿Preguntas?