



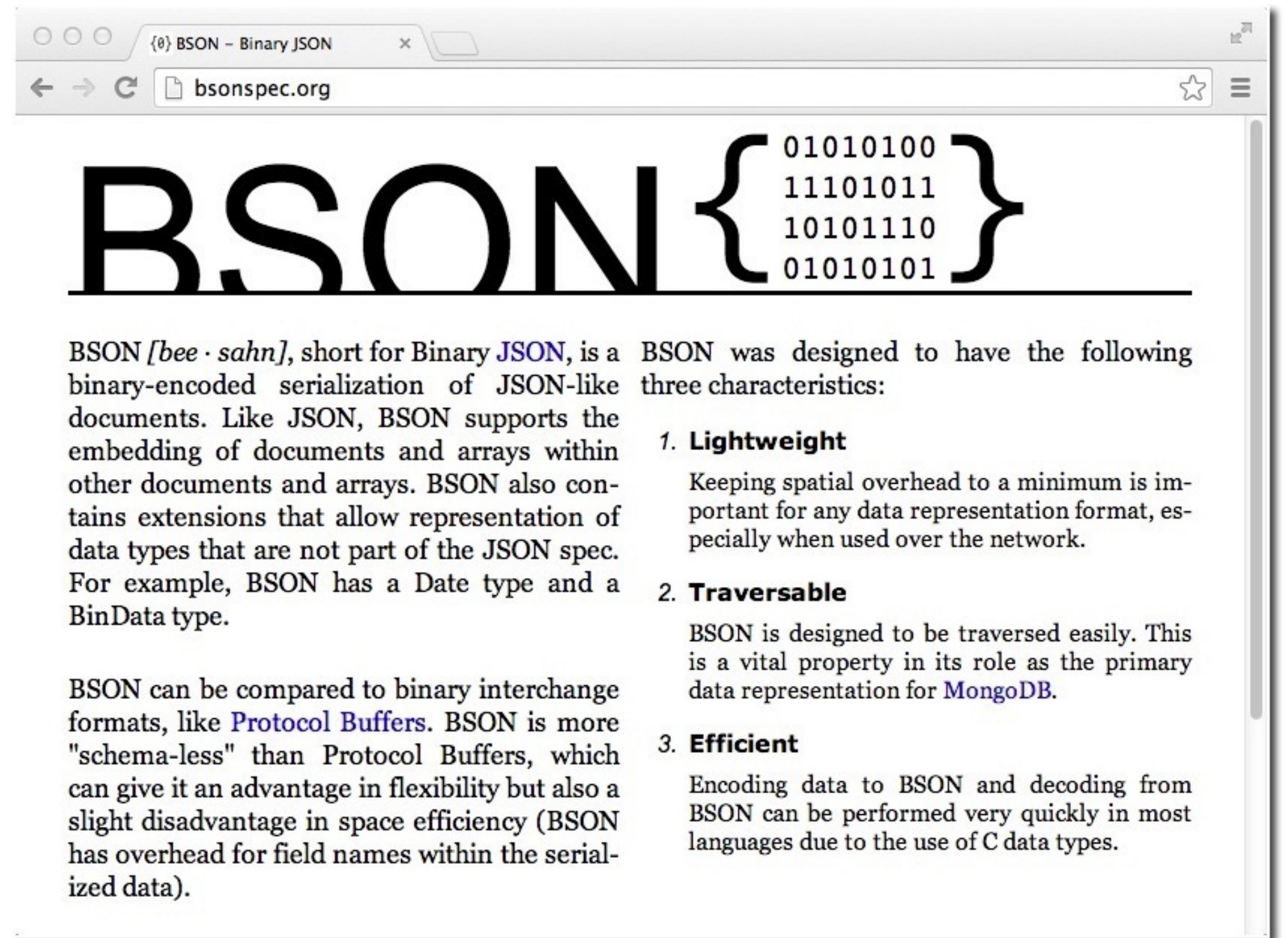
Índice

- BSON
- Trabajando con el shell
- `ObjectId`
- Operaciones
 - Consultas
 - Actualización
 - Borrado



2.1 BSON

- *Binary JSON*
- <http://bsonspec.org/>
- Representa un *superset* de JSON:
 - Almacena datos en binario
 - Incluye un conjunto de tipos de datos no incluidos en JSON
ObjectId, Date o BinData.





Restricciones BSON

- No pueden tener un tamaño superior a 16 MB.
- El atributo `_id` queda reservado para la clave primaria.
- Los nombres de los campos no pueden empezar por \$.
- Los nombres de los campos no pueden contener el `.`

```
var yo = {  
  nombre: "Aitor",  
  apellidos: "Medrano",  
  fnac: new Date("Oct 3, 1977"),  
  hobbies: ["programación", "videojuegos", "baloncesto"],  
  casado: true,  
  hijos: 2,  
  fechaCreacion = new Timestamp()  
}
```



A tener en cuenta con *MongoDB*

- No asegura que el orden de los campos se respete.
- Es sensible a los tipos de los datos
- Es sensible a las MAYÚSCULAS

```
{ "edad" : "18" }  
{ "edad" : 18 }  
{ "Edad" : 18 }
```



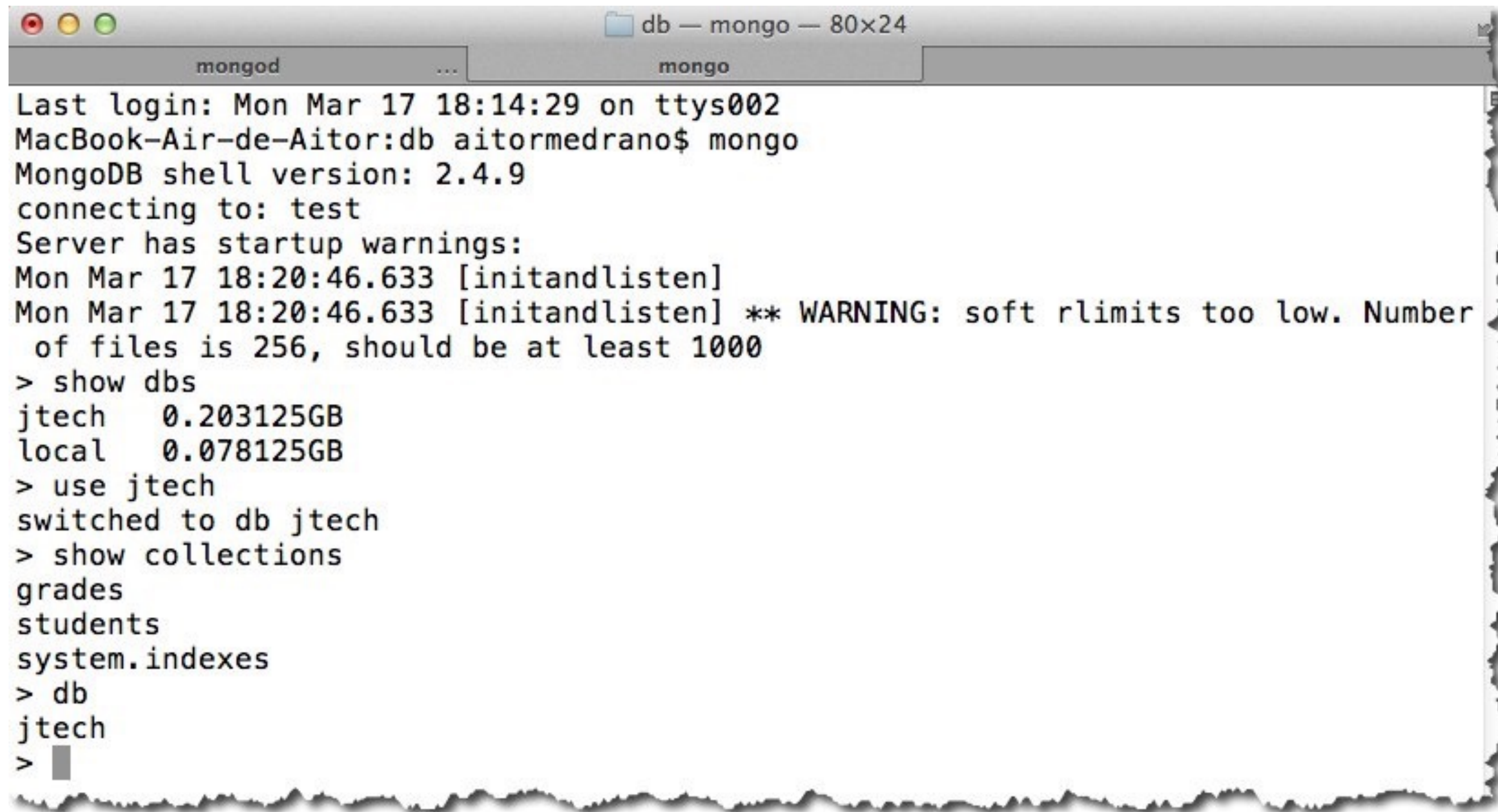
2.2 Trabajando con el *shell*

- *Shell* con sintaxis *JavaScript*
 - Cursores para subir/bajar comandos recientes

Comando	Función
<code>show dbs</code>	Muestra el nombre de las bases de datos
<code>show collections</code>	Muestra el nombre de las colecciones
<code>db</code>	Muestra el nombre de la base de datos que estamos utilizando
<code>db.dropDatabase()</code>	Elimina la base de datos actual
<code>db.help()</code>	Muestra los comandos disponibles
<code>db.version()</code>	Muestra la versión actual del servidor



Ejemplo *shell* I



```
db — mongo — 80x24
mongod ... mongo
Last login: Mon Mar 17 18:14:29 on ttys002
MacBook-Air-de-Aitor:db aitormedrano$ mongo
MongoDB shell version: 2.4.9
connecting to: test
Server has startup warnings:
Mon Mar 17 18:20:46.633 [initandlisten]
Mon Mar 17 18:20:46.633 [initandlisten] ** WARNING: soft rlimits too low. Number
  of files is 256, should be at least 1000
> show dbs
jtech    0.203125GB
local    0.078125GB
> use jtech
switched to db jtech
> show collections
grades
students
system.indexes
> db
jtech
> █
```



Ejemplo *shell* II

```
> db.people.insert(yo)
> db.people.find()
{ "_id" : ObjectId("53274f9883a7adeb6a573e64"), "nombre" : "Aitor", "apellidos" : "Medrano",
"fnac" : ISODate("1977-10-02T23:00:00Z"), "hobbies" : [ "programación", "videojuegos",
"baloncesto" ], "casado" : true, "hijos" : 2, "fechaCreacion" : Timestamp(1425633249, 1) }
> yo.email = "aitormedrano@gmail.com"
aitormedrano@gmail.com
> db.people.save(yo) // upsert
> db.people.find()
{ "_id" : ObjectId("53274f9883a7adeb6a573e64"), "nombre" : "Aitor", "apellidos" : "Medrano",
"fnac" : ISODate("1977-10-02T23:00:00Z"), "hobbies" : [ "programación", "videojuegos",
"baloncesto" ], "casado" : true, "hijos" : 2, "fechaCreacion" : Timestamp(1425633249, 1) }
{ "_id" : ObjectId("53274fca83a7adeb6a573e65"), "nombre" : "Aitor", "apellidos" : "Medrano",
"fnac" : ISODate("1977-10-02T23:00:00Z"), "hobbies" : [ "programación", "videojuegos",
"baloncesto" ], "casado" : true, "hijos" : 2, "fechaCreacion" : Timestamp(1425633373, 1), "email"
: "aitormedrano@gmail.com" }
> db.people.find().forEach(printjson)
```



Si tenemos una colección vacía...

```
db.people.insert({ nombre : "Aitor", edad : 37, profesion : "Profesor" })
db.people.save({ nombre : "Aitor", edad : 37, profesion : "Profesor" })
```




Shell y JavaScript

- Carga de *scripts* desde el *shell*:

```
> load( "scripts/misDatos.js" );  
> load( "/data/db/scripts/misDatos.js" );
```

- Lanzar *script* desde consola:

```
mongo expertojava misDatos.js
```

- Ejecutar fragmento de código en el *shell*:

```
> for (var i=0;i<10;i++) {  
... db.espias.insert({"nombre":"James Bond " + i, "agente":"00" + i});  
... }  
WriteResult({ "nInserted" : 1 })  
> db.espias.find()  
{ "_id" : ObjectId("56ac98f42561aa7b170e4299"), "nombre" : "James Bond 0", "agente" : "000" }  
{ "_id" : ObjectId("56ac98f42561aa7b170e429a"), "nombre" : "James Bond 1", "agente" : "001" }  
{ "_id" : ObjectId("56ac98f42561aa7b170e429b"), "nombre" : "James Bond 2", "agente" : "002" }  
{ "_id" : ObjectId("56ac98f42561aa7b170e429c"), "nombre" : "James Bond 3", "agente" : "003" }  
{ "_id" : ObjectId("56ac98f42561aa7b170e429d"), "nombre" : "James Bond 4", "agente" : "004" }  
{ "_id" : ObjectId("56ac98f42561aa7b170e429e"), "nombre" : "James Bond 5", "agente" : "005" }  
{ "_id" : ObjectId("56ac98f42561aa7b170e429f"), "nombre" : "James Bond 6", "agente" : "006" }  
{ "_id" : ObjectId("56ac98f42561aa7b170e42a0"), "nombre" : "James Bond 7", "agente" : "007" }  
{ "_id" : ObjectId("56ac98f42561aa7b170e42a1"), "nombre" : "James Bond 8", "agente" : "008" }  
{ "_id" : ObjectId("56ac98f42561aa7b170e42a2"), "nombre" : "James Bond 9", "agente" : "009" }  
> 
```




2.3 ObjectId

- atributo `_id` → global, único e inmutable
- clave primaria
- BSON de 12 bytes formado por:
 - el *timestamp* actual (4 bytes)
 - un identificador de la máquina / *hostname* (3 bytes) donde se genera
 - un identificador del proceso (2 bytes) donde se genera
 - un número aleatorio (3 bytes).
- Lo crea el *driver*
- Podemos obtener a partir del `ObjectId` la fecha de creación del documento, mediante el método `getTimestamp()` del atributo `_id`.

```
> db.people.find()[0]._id
ObjectId( "53274f9883a7adeb6a573e64" )
> db.people.find()[0]._id.getTimestamp()
ISODate( "2014-03-17T19:40:08Z" )
```



`_id` y `ObjectId`

- Si al insertar un documento, no definimos el atributo `_id`, el *driver* crea un `ObjectId` de manera automática
- Si lo ponemos nosotros de manera explícita, *MongoDB* no añadirá ningún `ObjectId`.
 - Debemos asegurarnos que sea único (podemos usar números, cadenas, etc...).

```
db.people.insert({_id:3, nombre:"Marina", edad:6 })
```

- El `_id` también puede ser un documento en sí, y no un valor numérico.

```
db.people.insert({_id:{nombre:'Aitor', apellidos:'Medrano',  
twitter:'@aitormedrano'}, ciudad:'Elx'});
```



2.4 Consultas

- Método `.find()`
- Devuelve un cursor
 - Se queda abierto con el servidor y se cierra automáticamente a los 10 minutos de inactividad o al finalizar su recorrido.
- Si hay muchos resultados, la consola nos mostrará un subconjunto de los datos (20) → `it`

```
> db.people.find()  
{ "_id" : ObjectId("53274f9883a7adeb6a573e64"), "nombre" : "Aitor", "apellidos" :  
"Medrano", "fnac" : ISODate("1977-10-02T23:00:00Z"), "hobbies" : [ "programación",  
"videojuegos", "baloncesto" ], "casado" : true, "hijos" : 2 }  
{ "_id" : ObjectId("53274fca83a7adeb6a573e65"), "nombre" : "Aitor", "apellidos" :  
"Medrano", "fnac" : ISODate("1977-10-02T23:00:00Z"), "hobbies" : [ "programación",  
"videojuegos", "baloncesto" ], "casado" : true, "hijos" : 2, "email" :  
"aitormedrano@gmail.com" }  
{ "_id" : 3, "nombre" : "Marina", "edad" : 6 }
```




Recuperar un documento

- Método `findOne()`
- Resultado formateado
- Mismo resultado con `find()`:
 - `db.people.find().pretty()`
 - `db.people.find().forEach(printjson)`

```
> db.people.findOne()  
{  
  "_id" : ObjectId("53274f9883a7adeb6a573e64"),  
  "nombre" : "Aitor",  
  "apellidos" : "Medrano",  
  "fnac" : ISODate("1977-10-02T23:00:00Z"),  
  "hobbies" : [  
    "programación",  
    "videojuegos",  
    "baloncesto"  
  ],  
  "casado" : true,  
  "hijos" : 2  
}
```



Preparando los ejemplos

- Colección de 800 calificaciones que han obtenido diferentes estudiantes en trabajos, exámenes o cuestionarios
 - El campo `type` puede tomar los siguientes valores: `quiz`, `homework` o `exam`

```
mongoimport -d expertojava -c grades --file grades.json
```

```
> db.grades.findOne()  
{  
  "_id" : ObjectId("50906d7fa3c412bb040eb577"),  
  "student_id" : 0,  
  "type" : "exam",  
  "score" : 54.6535436362647  
}
```



2.4.1 Criterios en Consultas

- Primer parámetro de `find`
 - Documento con criterios a cumplir (Y)

```
db.grades.find({student_id:0, type:"quiz"})
```

- *MongoDB* ofrece operadores lógicos para los campos numéricos:
 - Se pueden utilizar de forma simultánea sobre uno o más valores
 - Se colocan como un nuevo documento en el valor del campo a filtrar
 - nombre → operador
 - valor → valor a comparar

```
db.grades.find({ score:{$gt:95} })
db.grades.find({ score:{$gt:95, $lte:98}, type:"exam" })
db.grades.find({ type:"exam", score:{$gte:65} })
```

Comparador	Operador
menor que (<)	\$lt
menor o igual que (≤)	\$lte
mayor que (>)	\$gt
mayor o igual que (≥)	\$gte



Otros operadores de consulta

- ⚠ Cuidado al usar **polimorfismo** y almacenar en un mismo campo un entero y una cadena
 - Al hacer comparaciones para recuperar datos, no podemos mezclar cadenas con valores numéricos.
 - Se considera un *antipatrón*.

- **\$ne** → *not equals* → campos que no tienen un determinado valor

```
db.grades.find( {type: { $ne: "quiz" } } )
```

- **\$exists** → similar a la condición Valor No Nulo → campos que tienen algún valor

```
db.grades.find( { "score" : { $exists: true } } )
```

- **\$not** → operador negado

```
db.grades.find( {score: { $not: { $mod: [5,0] } } } )
```

- Se puede utilizar de manera conjunta con otros operadores

- **\$regex** → expresión regular

- Campos de texto

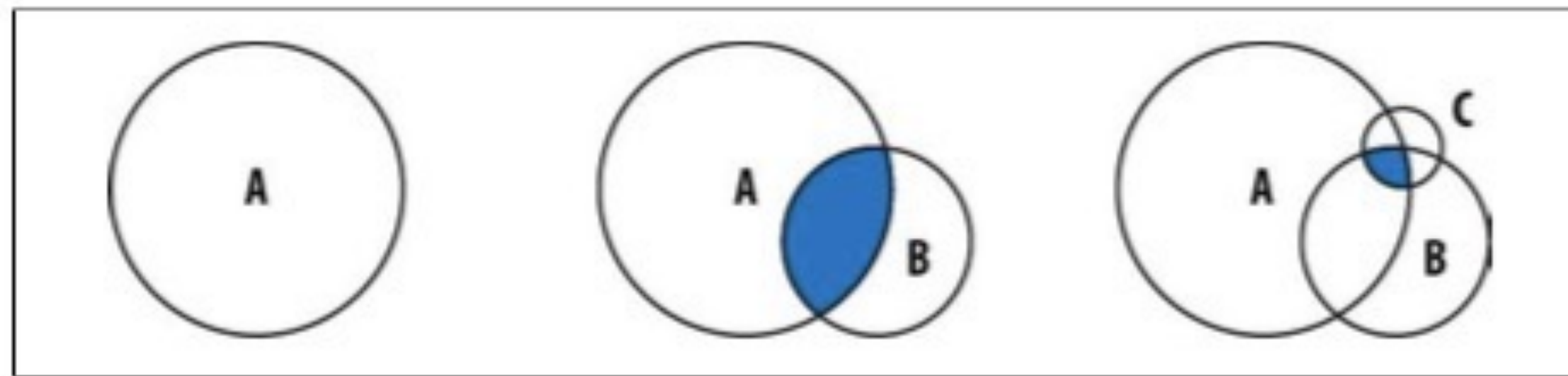
```
db.people.find( {nombre: /Aitor/ } )  
db.people.find( {nombre: /aitor/i } )  
db.people.find( {nombre: { $regex: /aitor/i } } )
```

- Similar a LIKE en SQL

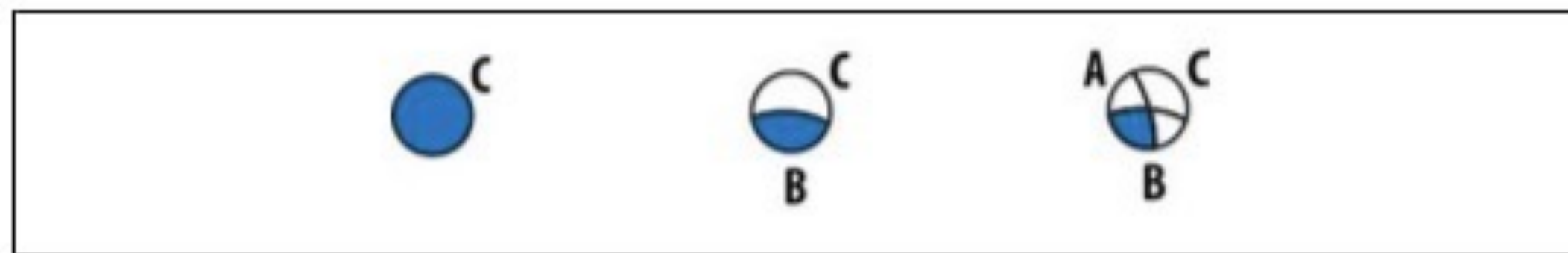


Optimizar consultas compuestas (Y)

- Supongamos que vamos a consultar documentos que cumplen los criterios A, B y C. Digamos que el criterio A lo cumplen 40.000 documentos, el B lo hacen 9.000 y el C sólo 200.



- Filtrar **el conjunto más pequeño** cuanto más pronto posible.





2.4.2 Proyección de Campos

- Para elegir los campos a devolver, pasar un segundo parámetro de tipo *JSON* con aquellos campos que deseamos mostrar con el valor `true` o `1`.
 - Si no se indica nada, por defecto siempre mostrará el campo `_id`

```
> db.grades.findOne({student_id:3},{score:true})  
{ "_id" : ObjectId("50906d7fa3c412bb040eb583"), "score" : 92.6244233936537 }
```

- Si queremos que no se muestre el `_id`, lo pondremos a `false` a `0`:

```
> db.grades.findOne({student_id:3},{score:true, _id:false})  
{ "score" : 92.6244233936537 }
```




2.4.3 Campos Anidados

- Notación punto
- Da igual el nivel en el que esté y su orden respecto al resto de campos.

```
{
  "producto" : "Condensador de Fluzo",
  "precio" : 1000000000000,
  "reviews" : [
    {
      "usuario" : "emmett",
      "comentario" : "¡Genial!",
      "calificacion" : 5
    }, {
      "usuario" : "marty",
      "comentario" : "¡Justo lo que necesitaba!",
      "calificacion" : 4
    }
  ]
}
```

```
db.catalogo.find( {"precio":{$gt:10000}, "reviews.calificacion":{$gte:5}} )
```



2.4.4 Condiciones Compuestas

- `$and` y `$or` para conjunción y la disyunción
- Operadores prefijo → se ponen antes de las subconsultas que se van a evaluar.
 - Reciben un array como parámetro

```
db.grades.find({ $or:[ {"type":"exam"}, {"score":{"$gte":65}} ]})  
db.grades.find({ $or:[ {"score":{"$lt":50}}, {"score":{"$gt":90}} ]})
```

```
=  
db.grades.find({ type:"exam", score:{ $gte:65} })  
db.grades.find({ $and:[ {type:"exam"}, {score:{ $gte:65}} ] })
```



Operadores para conjunciones

- `$nor` → negación de `$or`

```
db.grades.find({ score:{$gte:65}, $nor:[ {type:"quiz"}, {type:"homework"} ] })
```

- `$in` → admite un array con los posibles valores

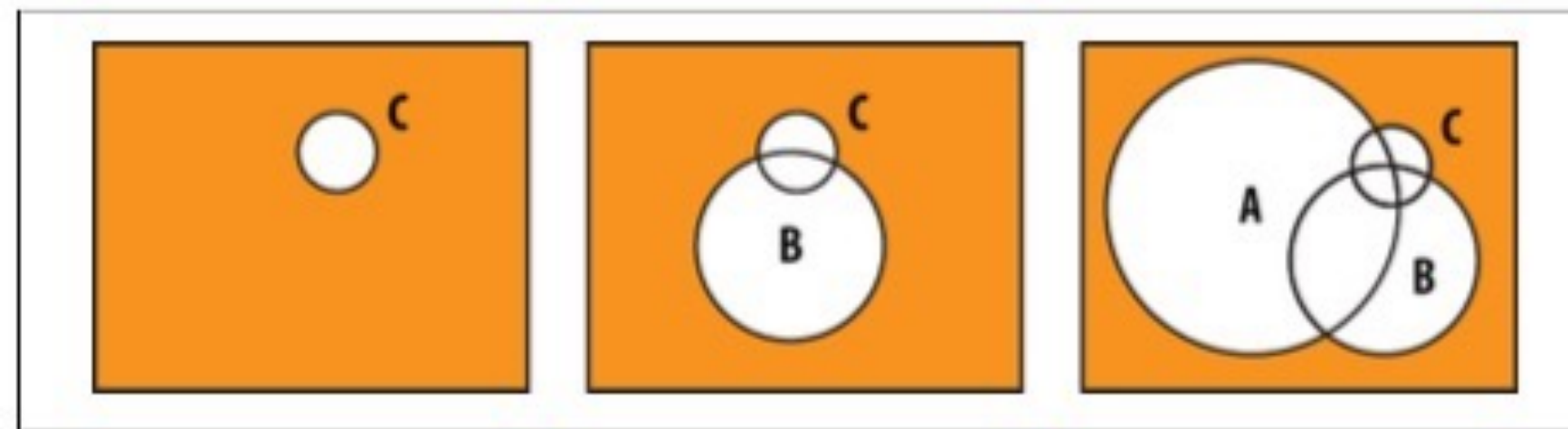
```
db.grades.find({ type:{$in:["quiz","exam"]} })
```

- `$nin` → negación de `$in`

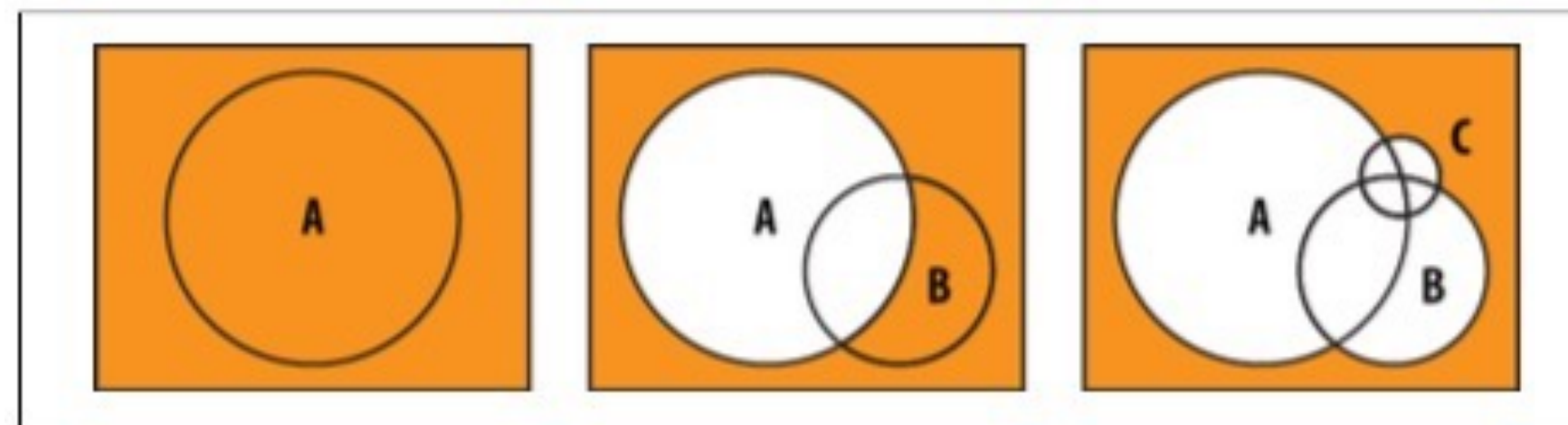


Optimizar consultas compuestas (O)

- Supongamos que vamos a consultar documentos que cumplen los criterios A, B y C. Digamos que el criterio A lo cumplen 40.000 documentos, el B lo hacen 9.000 y el C sólo 200.



- Filtrar **el conjunto más grande** cuanto más pronto posible.





2.4.5 Consultas sobre Arrays

- Se pueden consultar como un campo normal, siempre que sea un campo de 1^{er} nivel
- Consultas sobre la cantidad de elementos del array:
 - **\$all** → ocurrencias que tienen todos los valores del array
los valores pasados a la consulta serán un subconjunto del resultado
 - **\$in** → ocurrencias que cumple con alguno de los valores pasados
similar a usar **\$or** sobre un conjunto de valores de un mismo campo

```
db.people.find({ amistades: {$all: ["Juan", "David"]},  
               hobbies: {$in: ["footing", "baloncesto"]} })
```

- **\$elemMatch** → Para acceder a documentos embebidos

- **\$size** → cantidad de elementos

```
db.people.find( {hobbies : {$size : 3}} )
```

- **\$slice** → restringe el resultado

```
db.people.find( {hijos: {$gt:1}}, {hobbies: {$slice:2}} )
```

Sólo devuelve 2 hobbies



2.4.6 Conjunto de Valores

- Método `distinct`
- Permite obtener los diferentes valores que existen en un campo.

```
> db.grades.distinct('type')  
[ "exam", "quiz", "homework" ]
```

- Para filtrar los datos sobre los que se obtienen los valores → pasar un segundo parámetro con el criterio:

```
> db.grades.distinct('type', { score: { $gt: 99.9 } } )  
[ "exam" ]
```



2.4.7 Cursores

- Al hacer una consulta en el *shell* se devuelve un cursor.
- Se puede almacenar en una variable, y partir de ahí trabajar con él
- Métodos a partir de un cursor (devuelven un nuevo cursor):

Método	Uso	Lugar de ejecución
hasNext ()	true/false para saber si quedan elementos	Cliente
next ()	Pasa al siguiente documento	Cliente
limit (<i>numElementos</i>)	Restringe el número de resultados a <i>numElementos</i>	Servidor
sort ({ <i>campo</i> : 1 })	Ordena los datos por <i>campo</i> 1 ascendente o -1 o	Servidor
skip (<i>numElementos</i>)	Permite saltar <i>numElementos</i> con el cursor	Servidor



Trabajando con Cursores

- La consulta no se ejecuta hasta que el cursor comprueba o pasa al siguiente documento (`next/hasNext`),
 - Tanto `limit` como `sort` (ambos modifican el cursor) sólo se pueden realizar antes de recorrer cualquier elemento del cursor.
- Tras realizar una consulta con `find` se devuelve un cursor.
- Es habitual es encadenar una operación de `find` con `sort` y/o `limit` para **ordenar** el resultado por uno o más campos y posteriormente **limitar** el número de documentos a devolver.

```
db.grades.find({ type: 'homework' }).sort({score:-1}).limit(1)
```

```
db.grades.find().sort({score:-1}).skip(20).limit(10);
```




2.4.8 Contando Documentos

- Método `count()`
- También se puede emplear como un cursor.

```
db.grades.count( {type: "exam" } )  
db.grades.find( {type: "exam" } ).count( )  
db.grades.count( {type: "essay", score: { $gt: 90 } } )
```



2.5 Actualizar Documentos

- Para actualizar (y fusionar datos) → método `update` con 2 parámetros:
 1. la consulta para averiguar sobre qué documentos
 2. los campos a modificar

```
db.people.update( {nombre: "Steve Jobs"}, {nombre: "Domingo Gallardo", salario: 1000000} )
```



`update` hace un **reemplazo** de los campos

- si en el origen había 100 campos y en el `update` ponemos 2, el resultado sólo tendrá 2 campos
- Al actualizar, si el criterio de selección no encuentra el documento sobre el que hacer los cambios, no se realiza ninguna acción.



Upsert (Update + Insert)

- Acción de insertar un nuevo documento cuando no se encuentra ningún resultado en el criterio de una actualización.
- Para realizar un *upsert*, hay que pasarle un tercer parámetro al método con el objeto `{upsert: true}`

```
db.people.update( {nombre: "Domingo Gallardo"}, {name: "Domingo Gallardo",  
twitter: '@domingogallardo'}, {upsert: true})
```

- Otra manera es mediante la operación `save` (suponemos que `nombre` hace de `_id`):

```
db.people.save( {nombre: "Domingo Gallardo"}, {name: "Domingo Gallardo",  
twitter: '@domingogallardo'})
```

- Si no indicamos el valor `_id`, el comando `save` asume que es una inserción e inserta el documento en la colección.



Operadores de actualización

- Simplifican la actualización de campos.
- **\$set** → evita el reemplazo (si el campo no existe, se creará):

```
db.people.update( {nombre: "Aitor Medrano"}, { $set: {salario: 1000000} } )
```

- **\$inc** → incrementa el valor de una variable:

```
db.people.update( {nombre: "Aitor Medrano"}, { $inc: {salario: 1000} } )
```

- **\$unset** → elimina un campo:

```
db.people.update( {nombre: "Aitor Medrano"}, { $unset: {twitter: ''} } )
```

- Otros operadores: **\$mul**, **\$min**, **\$max**, **\$currentDate**



Actualización Múltiple

⚠ Si al actualizar la búsqueda devuelve más de un resultado, la actualización sólo se realiza sobre el primer resultado obtenido.

- Para modificar múltiples documentos, en el tercer parámetro indicaremos `{multi: true}`

```
db.grades.update( {type: 'exam'}, { '$inc': { 'score': 1 } }, {multi: true} );
```

- Las actualizaciones múltiples no se realizan de manera atómica
- *MongoDB* no soporta transacciones *isolated* → se pueden producir pausas (*pause yielding*).
- **Cada documento sí es atómico** → ninguno se va a quedar a la mitad.



findAndModify

- Permite encontrar y modificar un documento de manera atómica
- Evita que entre la búsqueda y la modificación el estado del documento se vea afectado.
- Por defecto, el documento devuelto será el resultado que ha encontrado con la consulta.
 - Para que devuelva el documento modificado → parámetro `new` a `true`.
 - Si no lo indicamos o lo ponemos a `false`, tendremos el comportamiento por defecto.

```
db.grades.findAndModify({  
  query: { student_id: 0, type: "exam" },  
  update: { $inc: { score: 1 } },  
  new: true  
})
```

- Caso de Uso: contadores y casos similares.



Renombrado campos

- Caso particular de actualización
- Posibilidad de renombrar un campo mediante el operador **\$rename**:

```
db.people.update( { _id: 1 },  
  { $rename: { 'nickname': 'alias', 'cell': 'movil' } } )
```



Actualización sobre Arrays

Operador	Propósito
\$push	Añade un elemento
\$pushAll	Añade varios elementos
\$addToSet	Añade un elemento sin duplicados
\$pull	Elimina un elemento
\$pullAll	Elimina varios elementos
\$pop	Elimina el primer o el último

- Preparando ejemplos:

```
db.enlaces.insert( {titulo:"www.google.es", tags:["mapas", "videos"]} )
```

```
{
  "_id" : ObjectId("54f9769212b1897ae84190cf"),
  "titulo" : "www.google.es",
  "tags" : [
    "mapas", "videos"
  ]
}
```



Añadiendo elementos a un array

- **\$push** y **\$pushAll** → añade uno a varios elementos de una sola vez

```
db.enlaces.update( {titulo:"www.google.es"}, {$push: {tags:"blog"}} )  
db.enlaces.update( {titulo:"www.google.es"}, {$pushAll: {tags:["calendario",  
"email", "mapas"]}} )
```

- Tanto **\$push** como **\$pushAll** no tienen en cuenta el contenido del array
- Si un elemento ya existe, se repetirá y tendremos duplicados.
- **\$addToSet** → evita duplicados:

```
db.enlaces.update( {titulo:"www.google.es"}, {$addToSet: {tags:"buscador"}} )
```

- Para añadir más de un campo a la vez sin duplicados → anidar el operador **\$each**:

```
db.enlaces.update( {titulo:"www.google.es"}, {$addToSet: {tags: { $each:["drive",  
"traductor"] }} } )
```



Eliminando elementos a un array

- `$pull` y `$pullAll` → elimina uno o varios elementos

```
db.enlaces.update({titulo:"www.google.es"}, {$pull: {tags:"traductor"}})  
db.enlaces.update({titulo:"www.google.es"}, {$pullAll: {tags:["calendario","email"]}})
```

- `$pop` → elimina elementos por el principio (-1) o el final (1)

```
db.enlaces.update({titulo:"www.google.es"}, {$pop: {tags:-1}})
```




Operador posicional (\$)

- Modifica el elemento que ocupa una determinada posición del array.
- \$ referencia al campo/documento que ha cumplido el filtro de búsqueda

- *Cambiar la calificación 80 por 82*

```
{ "_id" : 1, "grades" : [ 80, 85, 90 ] }
```

```
db.students.update( { _id: 1, grades: 80 }, { $set: { "grades.$" : 82 } } )
```

- *Cambiar el campo std a 6 de la calificación con nota 85*

```
{ "_id" : 4, "grades" :  
  [ { grade: 80, mean: 75, std: 8 },  
    { grade: 85, mean: 90, std: 5 },  
    { grade: 90, mean: 85, std: 3 } ] }
```

```
db.students.update( { _id: 4, "grades.grade": 85 }, { $set: { "grades.$.std" : 6 } } )
```



2.6 Borrando Documentos

- Método **remove**

```
db.people.remove( {nombre: "Domingo Gallardo"} )
```

- Si no pasamos ningún parámetro, borra toda la colección documento a documento
- Para ello, es más eficiente usar el método **drop**

```
db.people.drop( )
```

- Eliminar un campo no se considera un borrado, sino una actualización mediante **\$unset**.