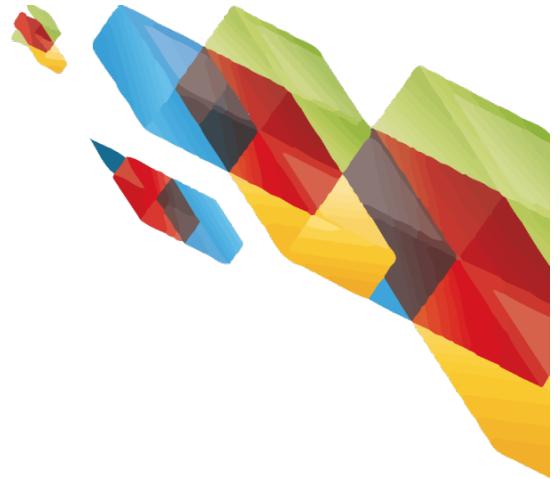




Ubaldo Taladriz
utaladriz@exe.cl
@utaladriz





Historia y características

- mongoDB = “Humongous DB”
- Open source
- Basada en documentos
- Rendimiento y Disponibilidad
- Escalamiento automático
- C-P en CAP
- Replicación y Sharding
- Map Reduce
- Updates rápidos
- Soporte completo para indexación

[-blog.mongodb.org/post/475279604/on-distributed-consistency-part-1](http://blog.mongodb.org/post/475279604/on-distributed-consistency-part-1)
[-mongodb.org/manual](http://mongodb.org/manual)



Tipos de Base de datos

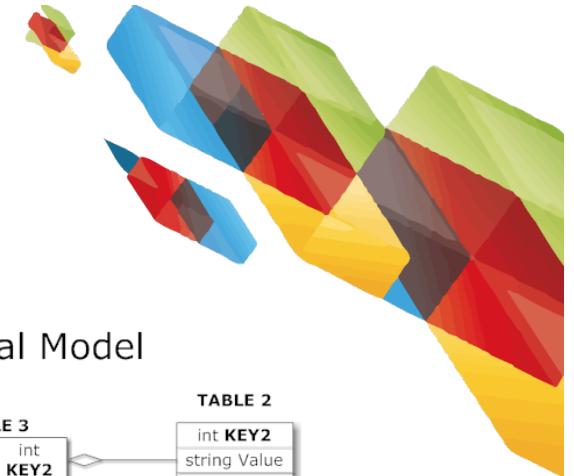
Key/value (Dynamo)

Columnar/tabular (HBase)

Documentales (mongoDB)

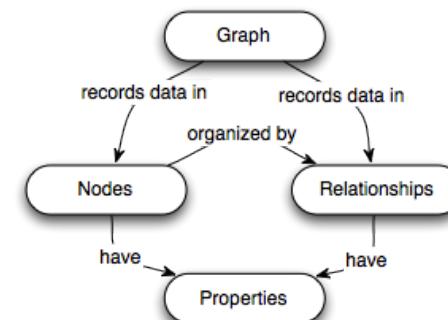
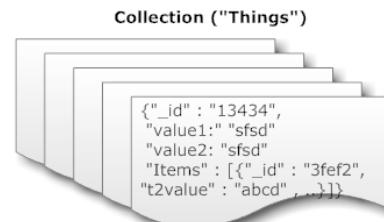
De Grafos (neo4j, Titan)

<http://www.aaronstannard.com/post/2011/06/30/MongoDB-vs-SQL-Server.aspx>



Relational Model

Document Model





Motivaciones

- Problemas con SQL
 - Esquema (¿?)
 - No escalan fácilmente (Diseñadas con tecnologías de los 90's)
 - Requiere de joins, que a veces no son intuitivos
- Algunos extras mongoDB
 - Soporte para distintos lenguajes (Java, Javascript, PHP, etc.)
 - “Run anywhere” (SO, VM's, cloud, etc.)
 - Mantiene los aspectos esenciales de una RDBMS pero con el aprendizaje de los sistemas noSQL de tipo key-value

http://www.slideshare.net/spf13/mongodb-9794741?v=qf1&b=&from_search=13



¿Alguien usa MongoDB ?

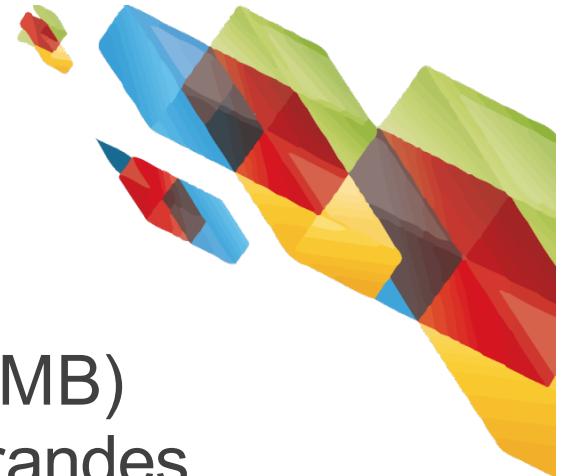


In Good Company



-Steve Francia, http://www.slideshare.net/spf13/mongodb-9794741?v=qf1&b=&from_search=13



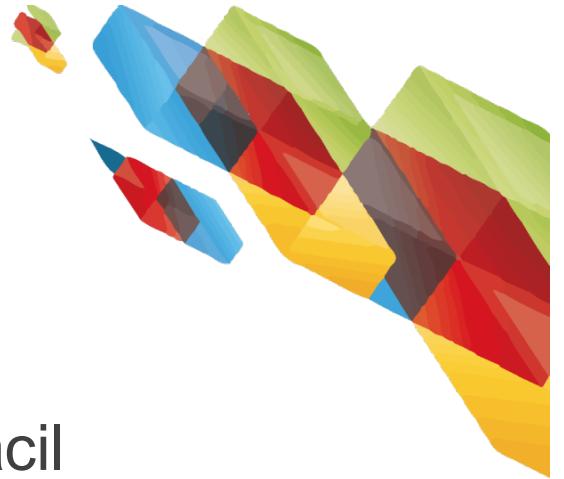


Modelo de datos

- Basado en documentos (máximo 16 MB)
- Para almacenar documentos más grandes MongoDB provee la GridFS API
- Los Documentos son en formato BSON, que son pares campo-valor
- Cada documento es almacenado en una colección
- Colecciones
 - Tienen índices en común
 - Son como una tabla ...
 - Pero no hay nada que garantice que los documentos tienen la misma estructura
- Los documentos pueden ser anidados (Máx 100)

[-docs.mongodb.org/manual/](http://docs.mongodb.org/manual/)





JSON

- ▶ “JavaScript Object Notation”
- ▶ Fácil de escribir y leer por humanos y fácil para computacionalmente procesarlos/generarlos
- ▶ Pueden contener objetos anidados
- ▶ Basado en
 - ▶ Pares nombre/valor
 - ▶ Listas ordenadas de valores

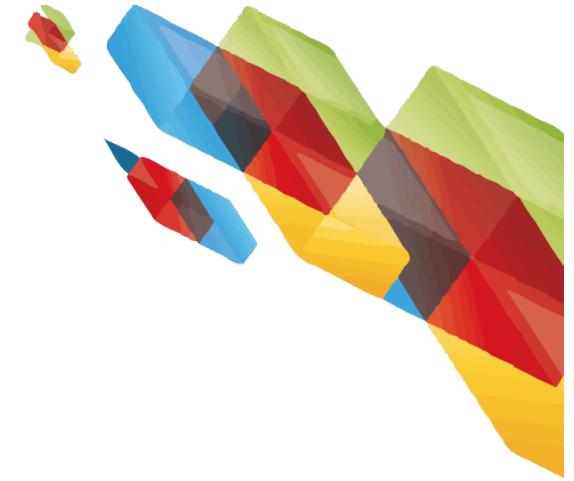
<http://json.org/>

BSON



- “Binary JSON”
- Codificación binaria de documentos JSON
- También permiten “referencias”
- Su estructura permite tener objetos como valores reduciendo la necesidad de joins
- Metas
 - Liviano
 - Navegable
 - Eficiente(Decodificación y codificación)

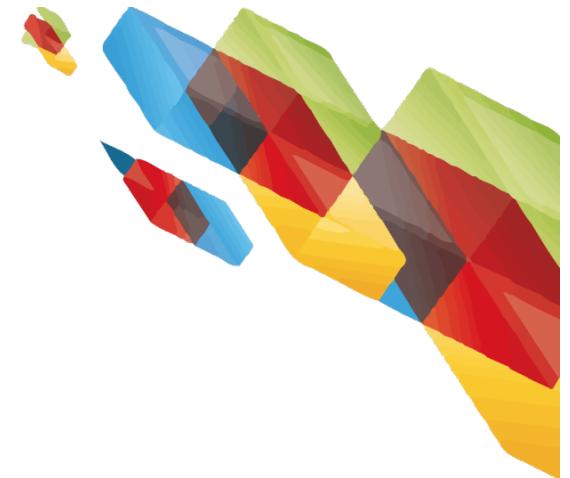
<http://bsonspec.org/>



Ejemplo BSON

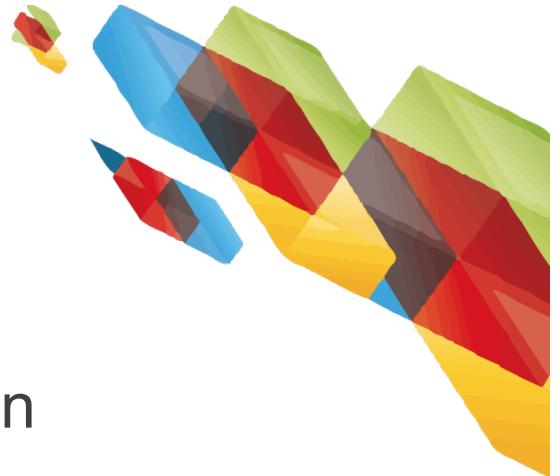
```
{  
  "_id" : "37010"  
  "city" : "ADAMS",  
  "pop" : 2660,  
  "state" : "TN",  
  "councilman" : {  
    "name": "John Smith"  
    "address": "13 Scenic Way"  
  }  
}
```

Tipos BSON



Tipo	Número
Double	1
String	2
Object	3
Array	4
Binary data	5
Objectid	7
Boolean	8
Date	9
Null	10
Regular Expression	11
JavaScript	13
Symbol	14
JavaScript (with scope)	15
32-bit integer	16
Timestamp	17
64-bit integer	18
Min key	255
Max key	127

El numero puede ser usado con el operador \$type para consultar el tipo



El campo _id

- Por omisión cada documento contiene un campo `_id` :
 - Actúa como llave primaria dentro de la colección
 - Es un valor único, inmutable y puede ser de cualquier tipo exceptuando arreglos.
 - El tipo de datos por omisión es **ObjectId**, el cual es “pequeño, único, rápido para generar y ordenar”
 - Ordenar por el valor de ObjectId es casi equivalente a ordenar en el tiempo de creación, eso implica casi costo cero a futuro.

<http://docs.mongodb.org/manual/reference/bson-types/>

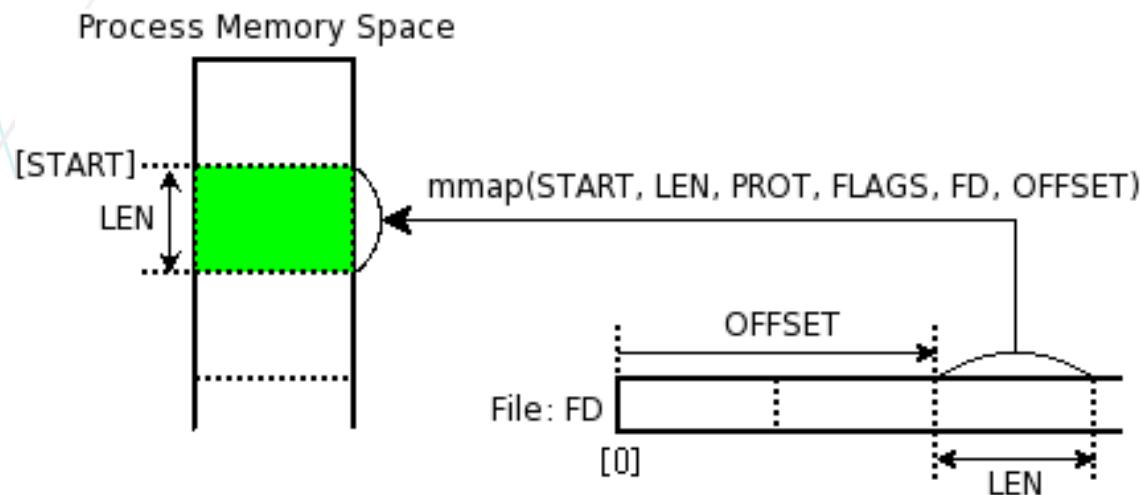
mongoDB vs. SQL



mongoDB	SQL
Documento / Document	Tupla
Colección / Collection	Tabla/Vista
PK: campo _id	PK: Cualquiera de los campos
No es uniforme	Uniforme basado en esuqemas
Índices / Index	Índices /
Estructuras empotradas / Embedded Structure	Joins
Sharding (Es como una partición)	Particiones

Archivos mapeados en memoria

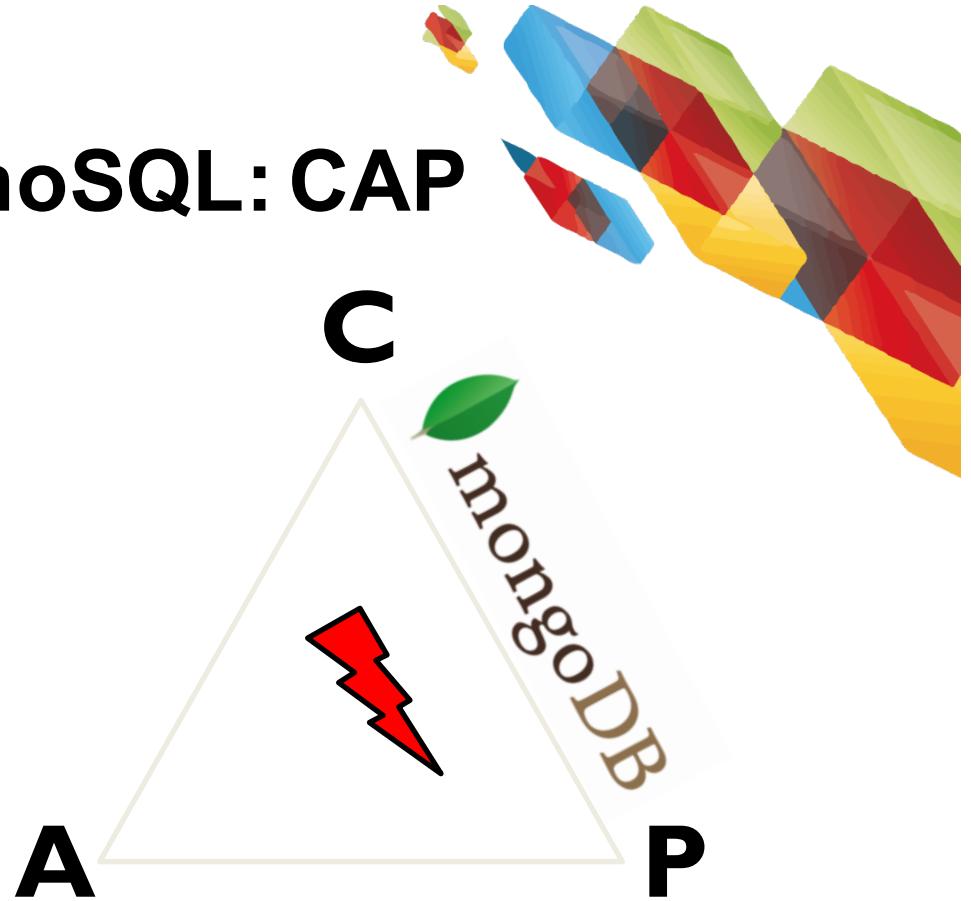
- Un archivo mapeado en memoria, es un segmento virtual, en el cual cada byte tiene una correlación uno a uno con los bytes de una porción del archivo”
- `mmap()`



13

Teoría de noSQL: CAP

- Muchos nodos (Sistemas distribuídos)
- Los nodos contienen replicas de los datos particionados
- Consistency / Consistencia
 - Todas las replicas tienen la misma versión de los datos
- Availability / Disponibilidad
 - El sistema permanece en operación aún cuando fallen nodos
- Partition tolerance / Particiones
 - Múltiples puntos de entradas
 - El sistema opera dividido



Teorema de CAP :
Satisfacer los tres al mismo
tiempo en un sistema
distribuido es imposible

14

ACID - BASE



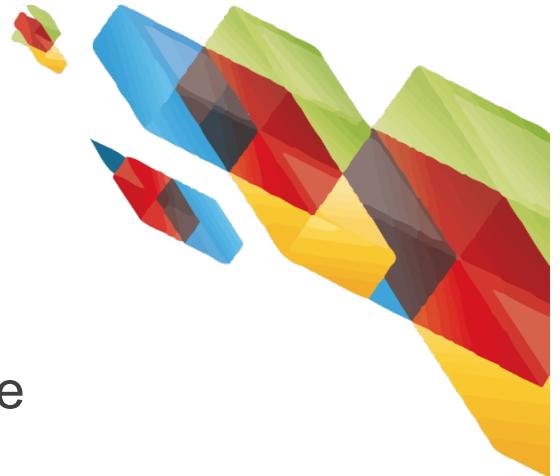
- Atomicity / Atómica
- Consistency / Consistencia
- Isolation / Aislación
- Durability / Durables



- Disponibilidad básica (CP)
- Soft-state
- Eventualmente consistente (AP)

Pritchett, D.: BASE: An Acid Alternative (queue.acm.org/detail.cfm?id=1394128)

Instalación



Para instalar mongoDB, en el siguiente link se debe seleccionar la arquitectura y el sistema operativo apropiado: <http://www.mongodb.org/downloads>

Primero extraer los archivos (preferentemente en el drive C).

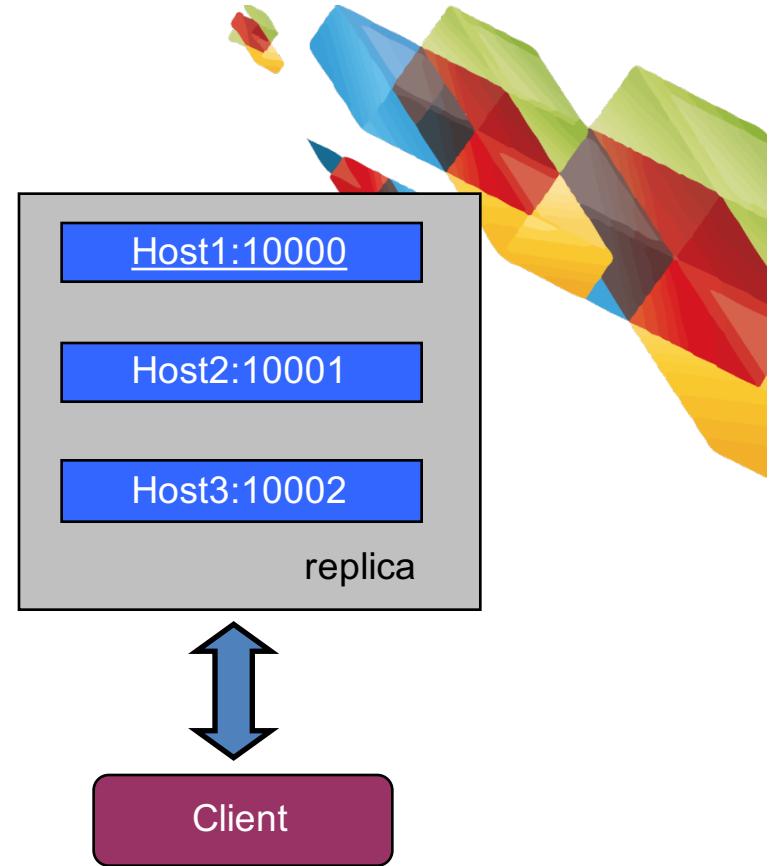
Finalmente se debe crear un directorio para los datos en algún lugar del drive para el uso de mongoDB

Ejemplo: “md data” seguido por “md data\db”

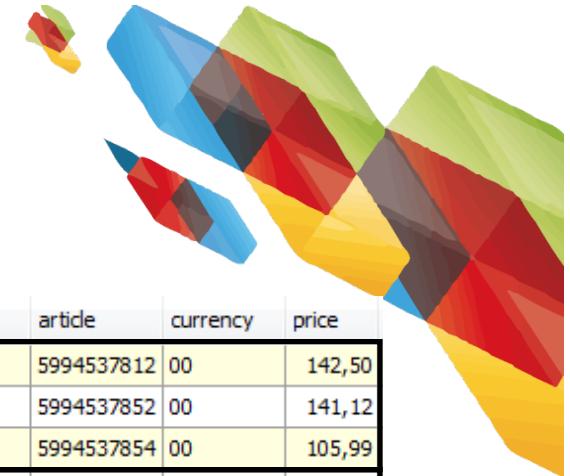
<http://docs.mongodb.org/manual/tutorial/install-mongodb-on-windows/>

Replicación

- Redundancia y tolerancia a fallas
- Zero downtime, para upgrades y mantenciones
- Replicación Master-slave en dos modalidades:
 - Strong Consistency
 - Delayed Consistency
- Replicación Geoespacial

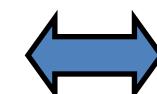
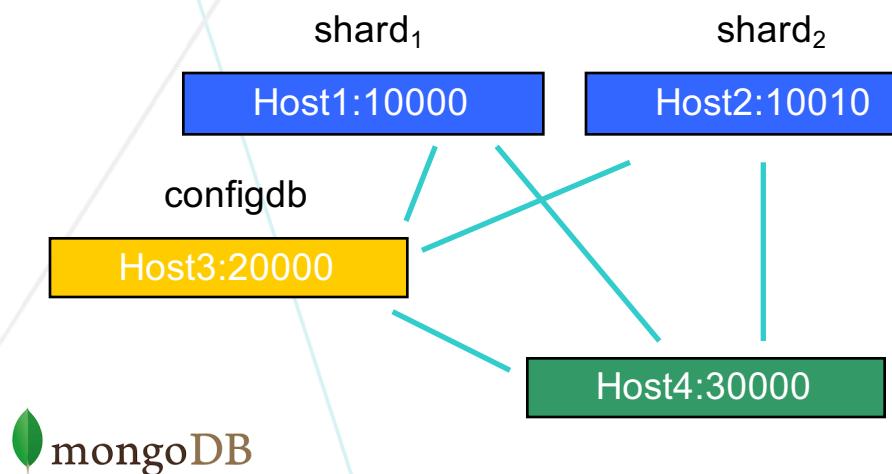


Sharding



- Particionamiento de los datos
- Escalamiento en el throughput de escritura
- Incremento de la capacidad
- Auto balanceo

id	company	customer	article	currency	price
4250250	020	073000	5994537812	00	142,50
4250251	020	073000	5994537852	00	141,12
4250252	020	073000	5994537854	00	105,99
4250253	020	073000	5994537856	00	109,52
4250254	020	073000	5994537862	00	131,49
4250255	020	073000	5994567308	00	29,86
4250256	020	073000	5994567422	00	57,13
4250257	020	073000	5994567428	00	68,59
4250258	020	073000	5994605089	00	51,09
4250259	020	073000	5994607975	00	93,93
4250260	020	073000	5994701005	00	74,22



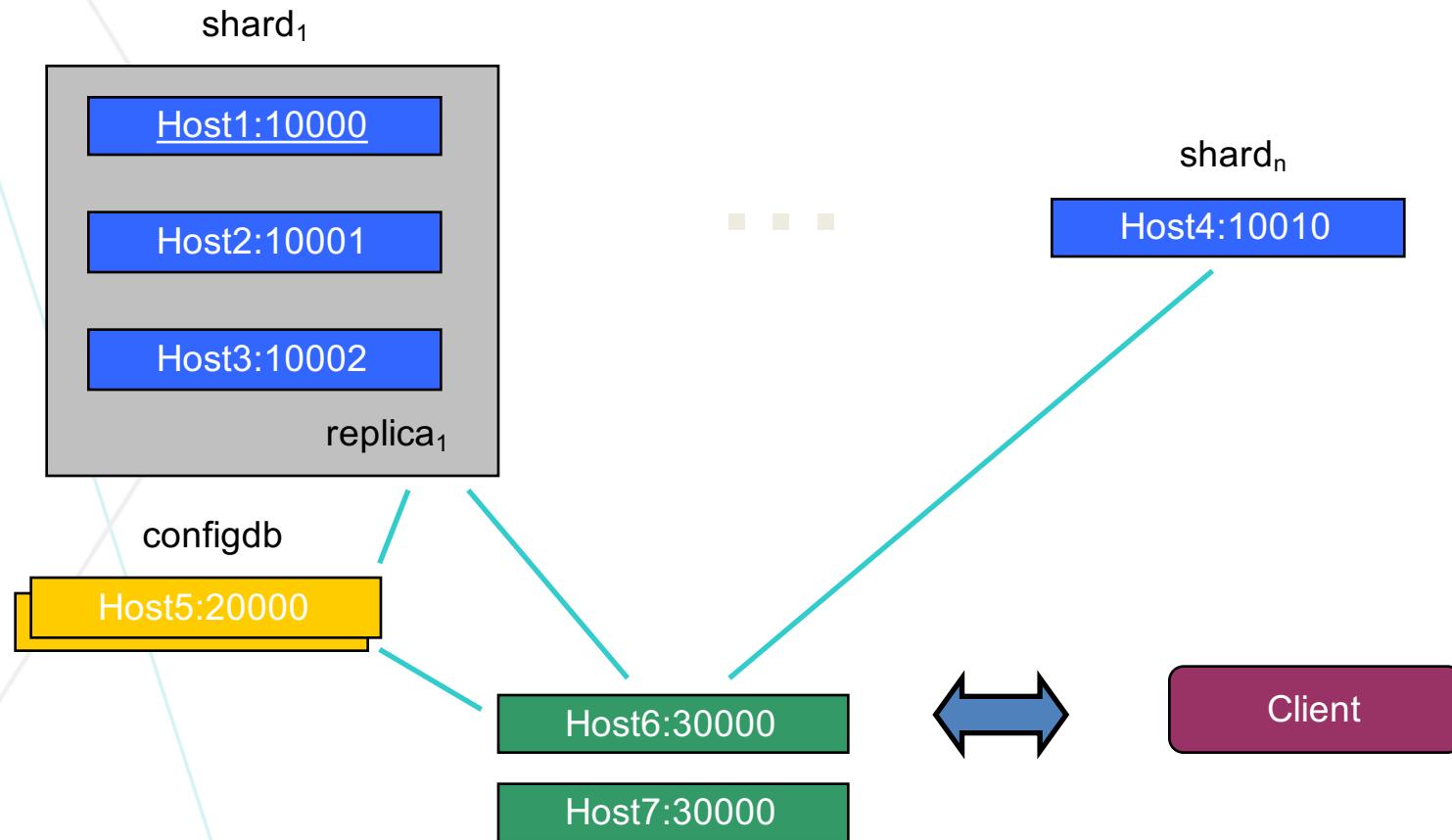
Client

18



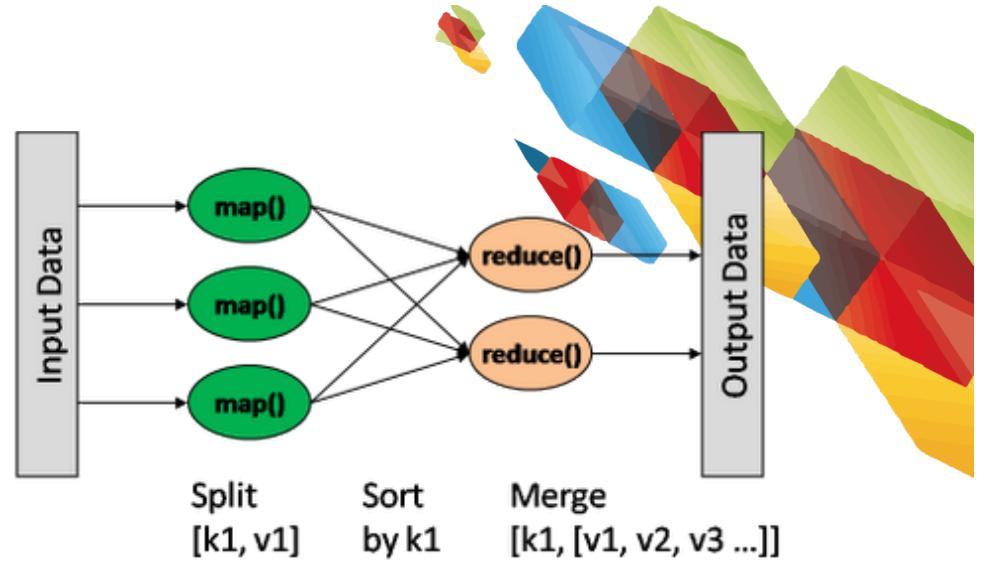
mongoDB

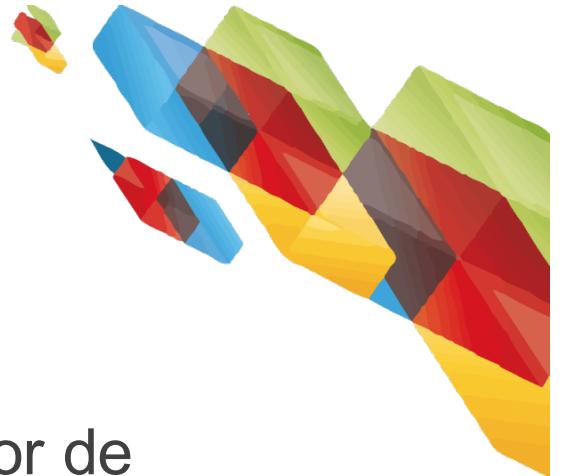
Configuración Mixta



Map/Reduce

```
db.collection.mapReduce(  
    <mapfunction>,  
    <reducefunction>,  
    {  
        out: <collection>,  
        query: <>,  
        sort: <>,  
        limit: <number>,  
        finalize: <function>,  
        scope: <>,  
        jsMode: <boolean>,  
        verbose: <boolean>  
    }  
)  
  
var mapFunction1 = function() { emit(this.cust_id, this.price); };  
  
var reduceFunction1 = function(keyCustId, valuesPrices)  
{ return sum(valuesPrices); };
```



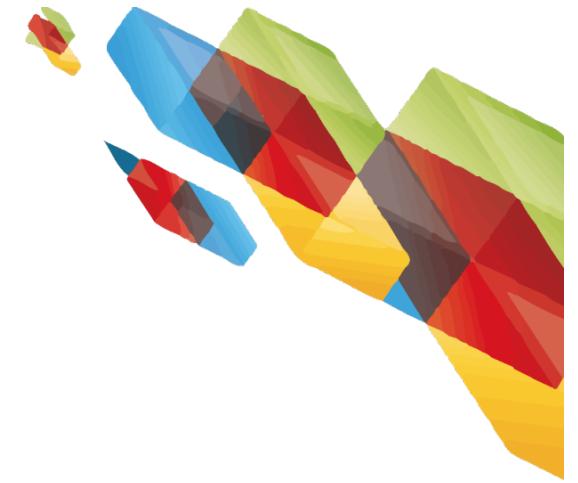


Instalación

En el directorio mongodb/bin ejecutar el programa mongod.exe para iniciar el motor de base de datos.

Para establecer la conexión con el servidor, abrir otra ventana en el mismo directorio y ejecutar el programa mongo.exe. Esto permite acceder al shell de MongoDB.

<http://docs.mongodb.org/manual/tutorial/getting-started/>



CRUD Básico

Create, Read, Update, Delete





CRUD: Utilizando el Shell

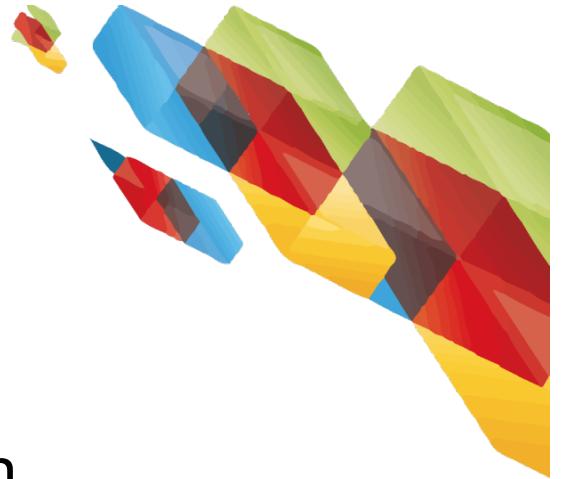
Para chequear cual base de datos se está usando se utiliza el comando: **db**

Mostrar todas las bases de datos: **show dbs**

Cambiar de base de datos o crear una: **use <nombre>**

Ver las colecciones existentes: **show collections**

Nota: Las bases de datos no son creadas hasta que se insertan datos



CRUD: Utilizando el Shell

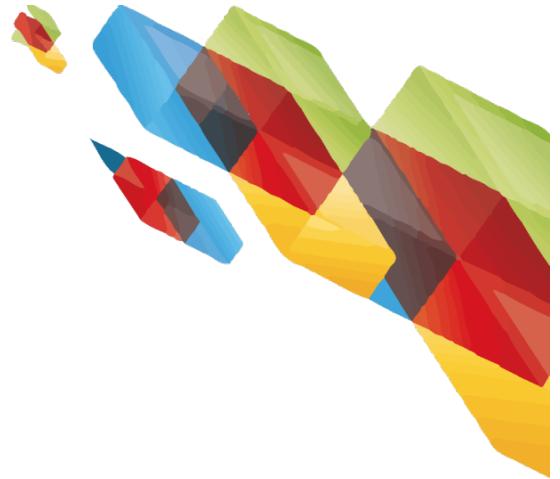
Para insertar documentos en una colección existente o **crear** una colección:

```
db.<colección>.insert(<documento>)
```

<=>

```
INSERT INTO <tabla>
VALUES(<valoresatributos>);
```

CRUD: Insertando datos



Insertando un documento

```
db.<colección>.insert({<campo>:<valor>})
```

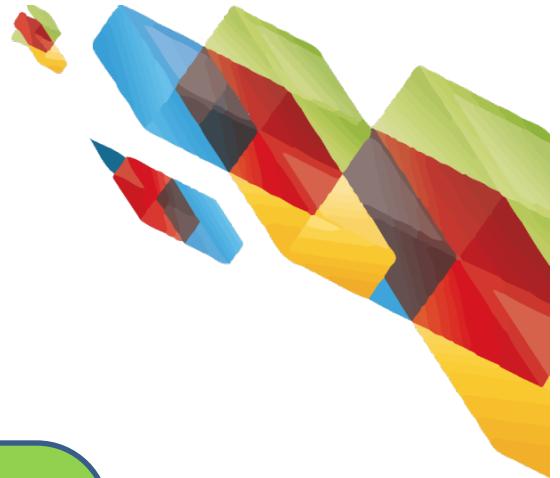
Insertar un documento con un campo que no existe en la colección es algo completamente soportado y aceptado por el modelo de objetos BSON.

Para insertar múltiples documentos se puede usar un arreglo.



CRUD: Consultas

- ▶ Se hacen sobre las colecciones
- ▶ Obtener todos: `db.<colección>.find()`
 - ▶ Retorna un cursor, el cual muestra en el shell los primeros 20 resultados.
 - ▶ Se puede agregar `.limit(<número>)` para limitar los resultados
 - ▶ `SELECT * FROM <tabla>;`
- ▶ Obtener un documento: `db.<colección>.findOne()`



CRUD: Consultas

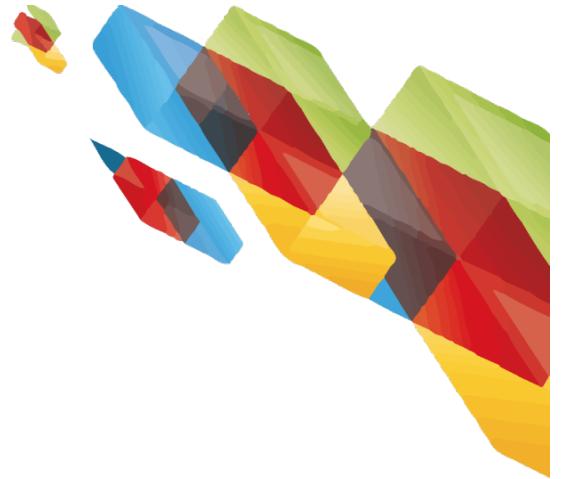
Para obtener un valor específico:

```
db.<colección>.find({<campo>:<valor>})
```

“AND”

```
db.<collection>.find({<campo1>:<valor1>,
                      <campo2>:<valor2>
                    })
```

```
SELECT *
FROM <tabla>
WHERE <campo1> = <valor1>
AND <campo2> = <valor2>;
```



CRUD: Consultas

OR

```
db.<colección>.find({ $or: [  
    <campo>:<valor1>  
    <campo>:<valor1>      ]  
})
```

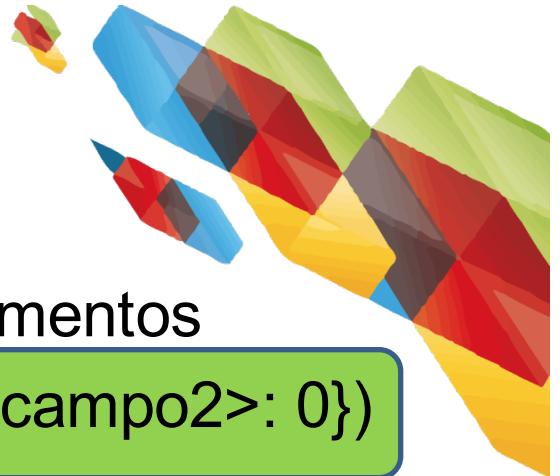
SELECT *

FROM <tabla>

```
WHERE <campo> = <valor1> OR <campo> = <valor2>;
```

Múltiples valores en el mismo campo

```
db.<colección>.find({<campo>: {$in [<valor1>, <valor2>]} })
```



CRUD: Consultas

Incluyendo/excluyendo campos de los documentos

```
db.<colección>.find({<campo1>:<valor>}, {<campo2>: 0})
```

```
SELECT campo1  
FROM <tabla> where <campo1> = <valor>;
```

```
db.<colección>.find({<campo>:<valor>}, {<campo2>: 1})
```

Encontrar documentos con un campo

```
db.<colección>.find({<campo>: { $exists: true}})
```

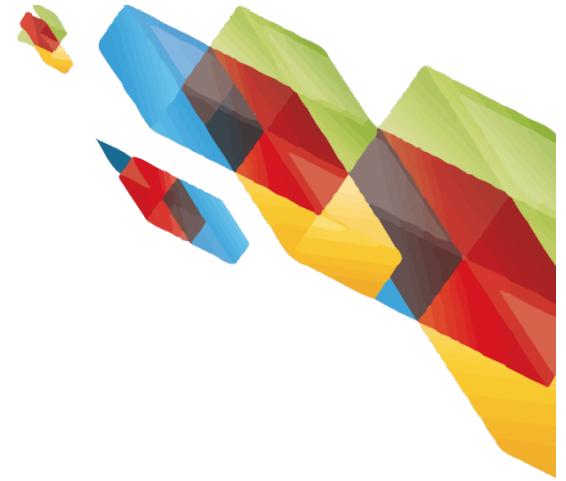


CRUD: Actualización

```
db.<colección>.update(  
  {<campo1>:<valor1>}, //todos los docs con campo1 = valo1  
  {$set: {<campo2>:<valor2>}}, //cambiar el valor del campo2  
  {multi:true} )           //actualizar múltiples documentos
```

upsert: Crea un nuevo documento cuando no se obtienen resultados a partir del criterio de búsqueda.

```
UPDATE <tabla>  
SET <campo2> = <valor2>  
WHERE <campo1> = <valor1>;
```



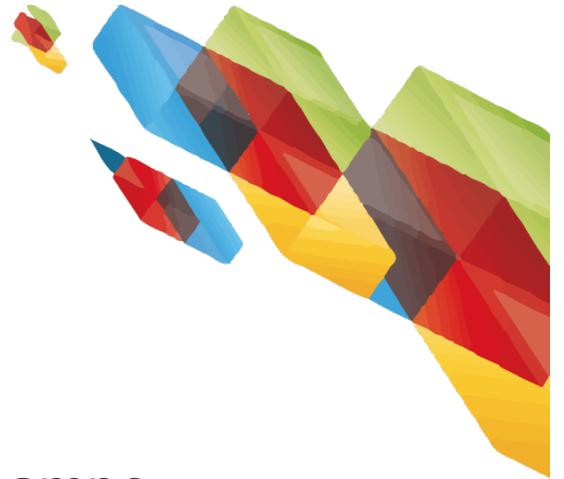
CRUD: Actualización

Para eliminar un campo

```
db.<colección>.update({<campo>:<valor>},  
                      { $unset: { <campo>: 1}})
```

Reemplazar todos los valores de un objeto

```
db.<collection>.update({<campo>:<valor>},  
                      { <campo>:<valor>,  
                        <campo>:<valornuevo>})
```



CRUD: Eliminar

Eliminar todos los registros para un valor de un campo

```
db.<colección>.remove({<campo>:<valor>})
```

```
DELETE FROM <tabla>
WHERE <campo> = <valor>;
```

Borrar solo el primer documento

```
db.<colección>.remove({<campo>:<valor>}, true)
```

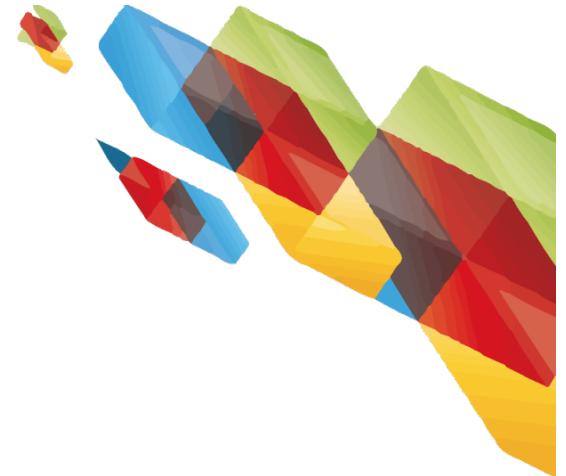


CRUD: Aislación

- Por omisión, todas las escrituras son atómicas, a nivel de documento.
- Esto implica que todas las escrituras pueden ser intercaladas con otras operaciones.
- Se puede aislar las escrituras en colecciones que no estén en sharding (**unsharded**) agregando la opción `$isolated:1` el área de consulta :

```
db.<collection>.remove({<field>:<value>, $isolated: 1})
```

Importación y exportación de datos



- Utilitarios mongoexport y mongoimport
- Utilizan strict mode representation
- Para respaldo se utiliza mongodump y mongorestore (A revisar más adelante)

```
mongoexport --db test --collection traffic --out traffic.json
```

```
mongoimport --db test --collection traffic --file traffic.json
```

Diseño de “Esquemas”



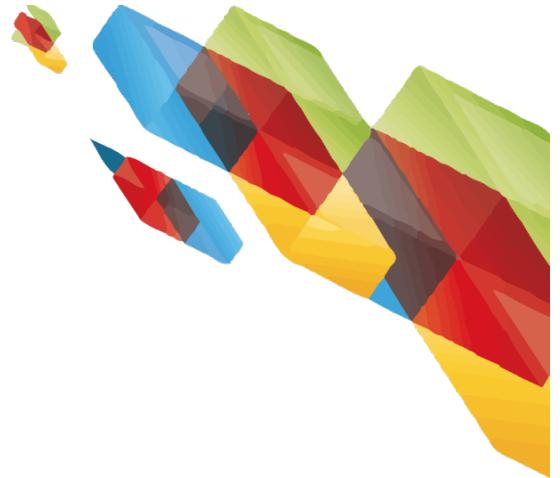


RDBMS		MongoDB
Database	→	Base de datos / Database
Table	→	Colección / Collection
Row	→	Documento / Document
Index	→	Indice / Index
Join	→	Documento empotrado/ Embedded Document
Foreign Key	→	Referencia / Reference

¿Por qué existen las bases de datos?

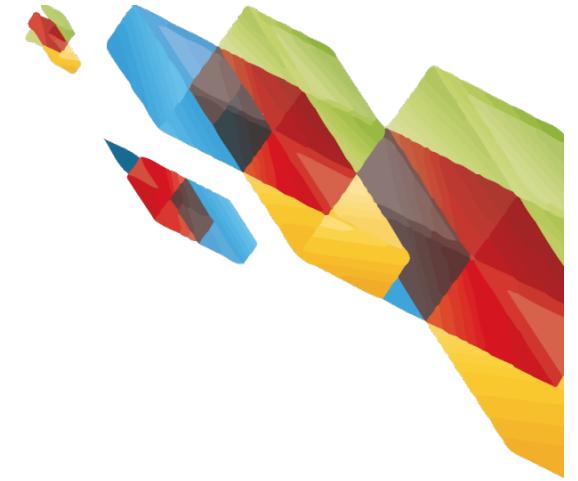


- ¿Por qué no escribimos programas que operen directamente con los objetos?
 - Límites de memoria
 - No es factible por parte del SO intercambiar info desde el disco a memoria directamente.
- ¿Por qué no podemos tener en la base de datos la misma estructura que en los programas?
 - Esa es una de las ideas de mongoDB



Mongo es libre de esquemas

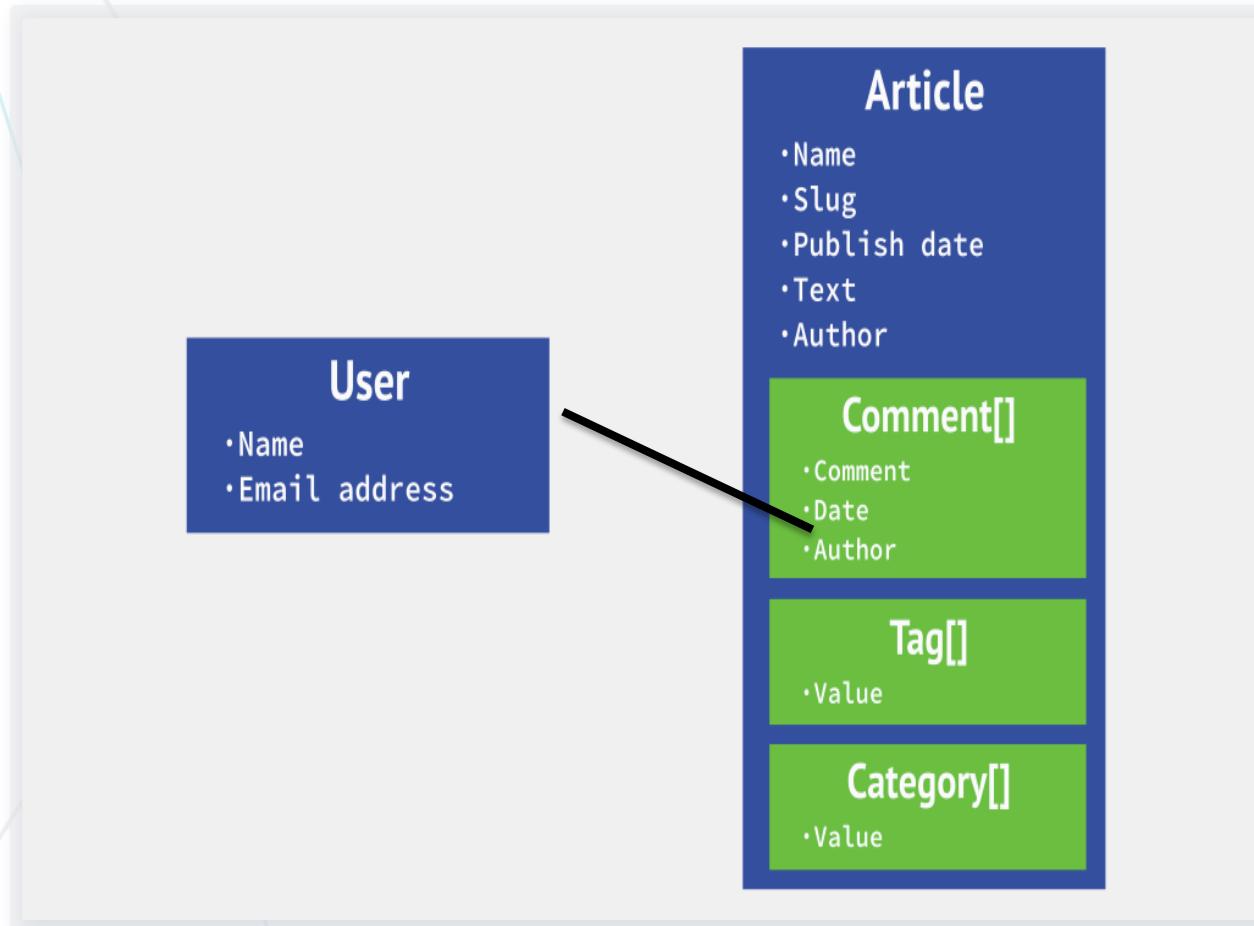
- El propósito de los esquemas en SQL es satisfacer los requerimientos de las tablas y el modelo de implementación de SQL
- Cada “fila” en una “tabla” es una estructura de datos, similar a un “struct” de C, o a “class” en Java. Una tabla es entonces un arreglo o lista de esas estructuras de datos
- El diseño de mongoDB es de la misma manera en que se componen documentos en JSON



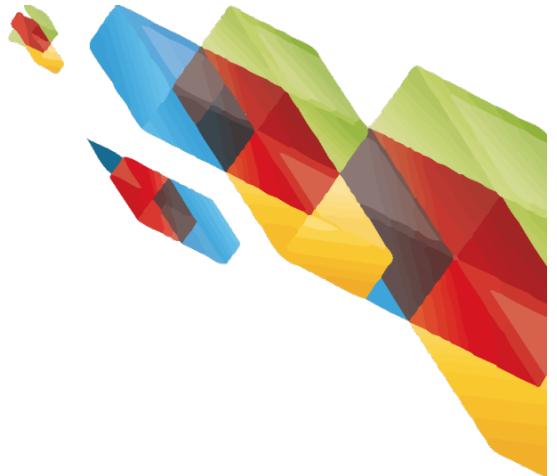
Existen algunos patrones

- Empotramiento/Embedding
- Vínculos/Linking

Empotramiento y vínculos



Relaciones uno a uno



```
zip = {  
    _id: 35004,  
    city: "ACMAR",  
    loc: [-86, 33],  
    pop: 6065,  
    State: "AL"  
}
```



```
Council_person = {  
    zip_id = 35004,  
    name: "John Doe",  
    address: "123 Fake St.",  
    Phone: 123456  
}
```

 mongoDB

```
zip = {  
    _id: 35004 ,  
    city: "ACMAR"  
    loc: [-86, 33],  
    pop: 6065,  
    State: "AL",  
  
    council_person: {  
        name: "John Doe",  
        address: "123 Fake St.",  
        Phone: 123456  
    }  
}
```



Otro ejemplo

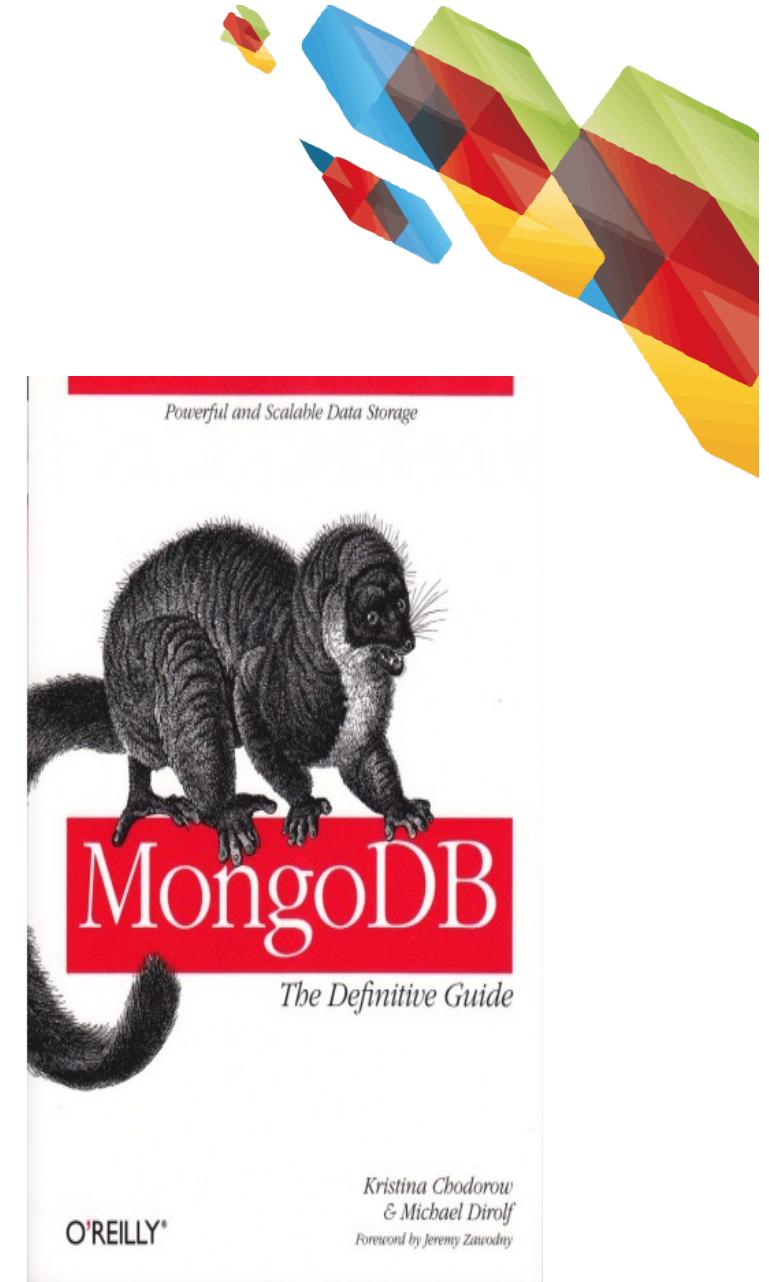
MongoDB: The Definitive Guide,
By Kristina Chodorow and Mike Dirolf

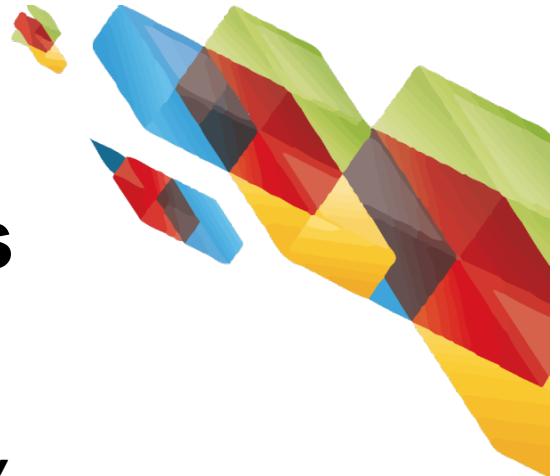
Published: 9/24/2010

Pages: 216

Language: English

Publisher: O'Reilly Media, CA





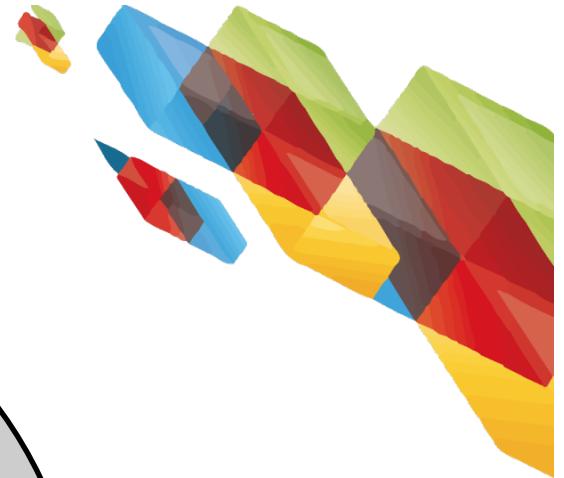
Empotramiento uno a muchos

```
book = {  
    title: "MongoDB: The Definitive Guide",  
    authors: [ "Kristina Chodorow", "Mike Dirolf" ]  
    published_date: ISODate("2010-09-24"),  
    pages: 216,  
    language: "English",  
    publisher: {  
        name: "O'Reilly Media",  
        founded: "1980",  
        location: "CA"  
    }  
}
```



Vínculos uno a muchos

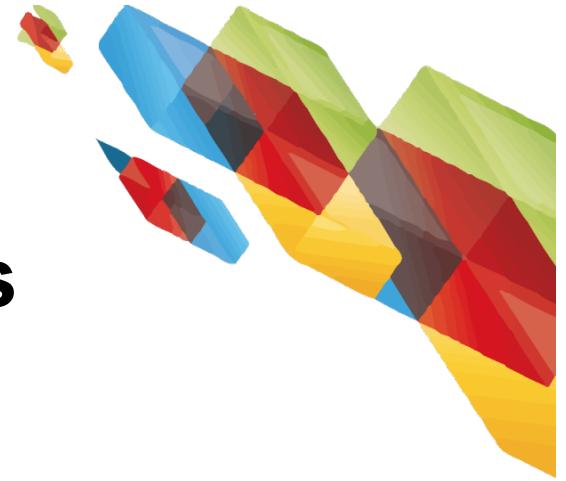
```
publisher = {  
    _id: "oreilly",  
    name: "O'Reilly Media",  
    founded: "1980",  
    location: "CA"  
}  
  
book = {  
    title: "MongoDB: The Definitive Guide",  
    authors: [ "Kristina Chodorow", "Mike Dirolf" ]  
    published_date: ISODate("2010-09-24"),  
    pages: 216,  
    language: "English",  
    publisher_id: "oreilly"  
}
```





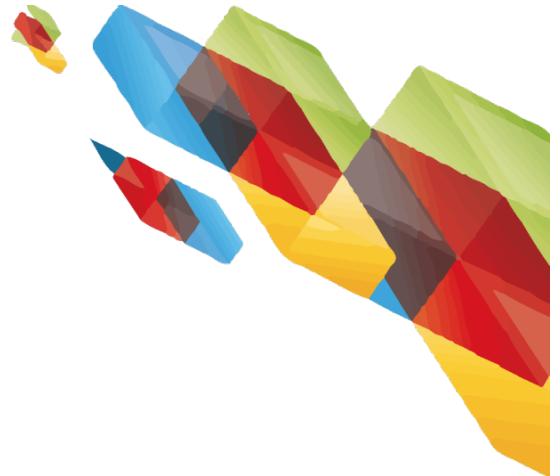
Empotramiento vs. Vínculos

- Empotramiento es como tener un join materializado previamente
- Las operaciones a nivel de documentos son muy eficientes y el servidor no tiene problemas para ejecutarlas.
- Se debe usar empotramiento cuando el objeto empotrado es usado o visualizado siempre dentro del contexto de su objeto padre.
- Los vínculos ofrecen más independencia en la manipulación de objetos pero agregando un costo extra a las operaciones



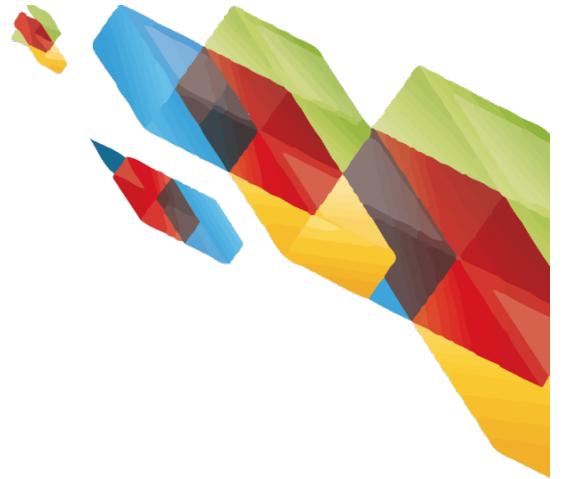
Relaciones muchos a muchos

- La relación se puede colocar en uno de los documentos (Vínculo) o con redundancia.
- ¿En cuál? Depende de cómo son consultados los documentos



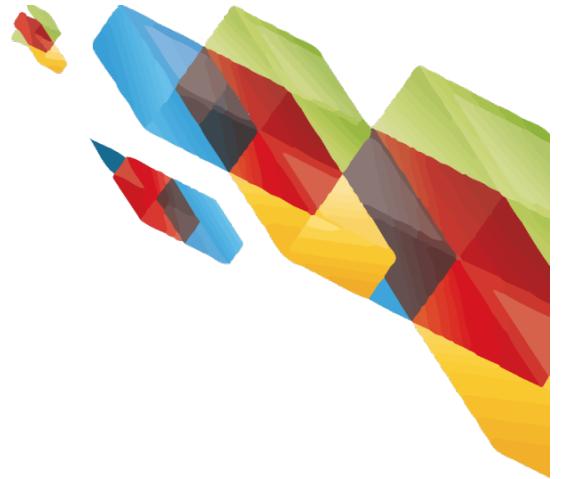
Ejemplo

```
book = {  
    title: "MongoDB: The Definitive Guide",  
    authors : [  
        { _id: "kchodorow", name: "Kristina  
Chodorow" },  
        { _id: "mdirolf", name: "Mike Dirolf" }  
    ]  
    published_date: ISODate("2010-09-24"),  
    pages: 216,  
    language: "English"  
}  
  
author = {  
    _id: "kchodorow",  
    name: "Kristina Chodorow",  
    hometown: "New York"  
}  
  
db.books.find( { authors.name : "Kristina Chodorow" }  
)
```



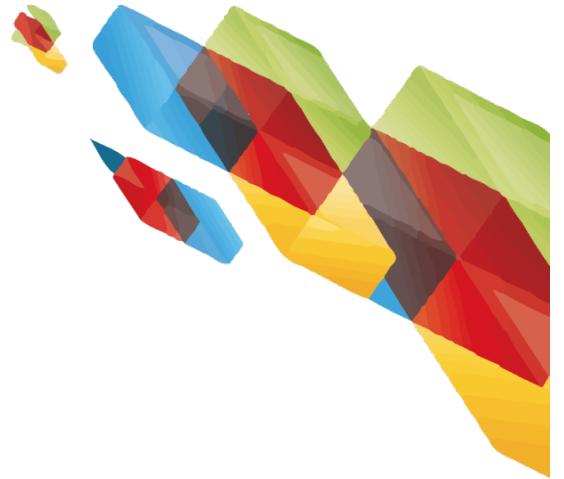
Otro ejemplo

- Un libro puede ser prestado a un estudiante (checkout)
- Un estudiante puede solicitar muchos libros en préstamo



Modelamiento de préstamos

```
student = {  
    _id: "joe"  
    name: "Joe Bookreader",  
    join_date: ISODate("2011-10-15"),  
    address: { ... }  
}  
  
book = {  
    _id: "123456789"  
    title: "MongoDB: The Definitive Guide",  
    authors: [ "Kristina Chodorow", "Mike Dirolf"  
,  
    ...  
}
```



Modelamiento de préstamos

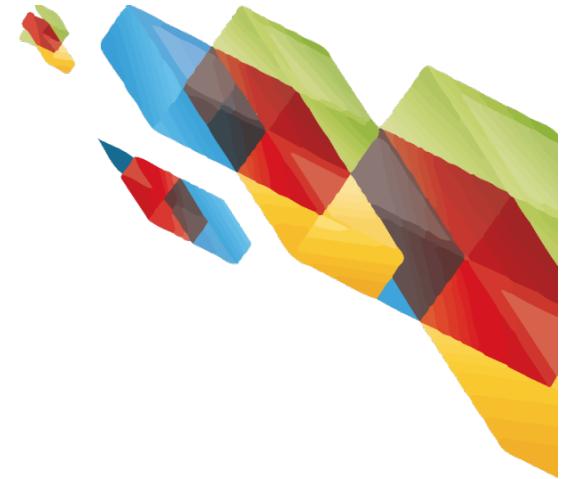
```
student = {  
    _id: "joe"  
    name: "Joe Bookreader",  
    join_date: ISODate("2011-10-15"),  
    address: { ... },  
    checked_out: [  
        { _id: "123456789", checked_out: "2012-  
10-15" },  
        { _id: "987654321", checked_out: "2012-  
09-12" },  
        ...  
    ]  
}
```



¿Desnormalizar?

La desnormalización provee **Datos Locales**, y **Datos Locales implican velocidad**

Los Join “matan”



Índices

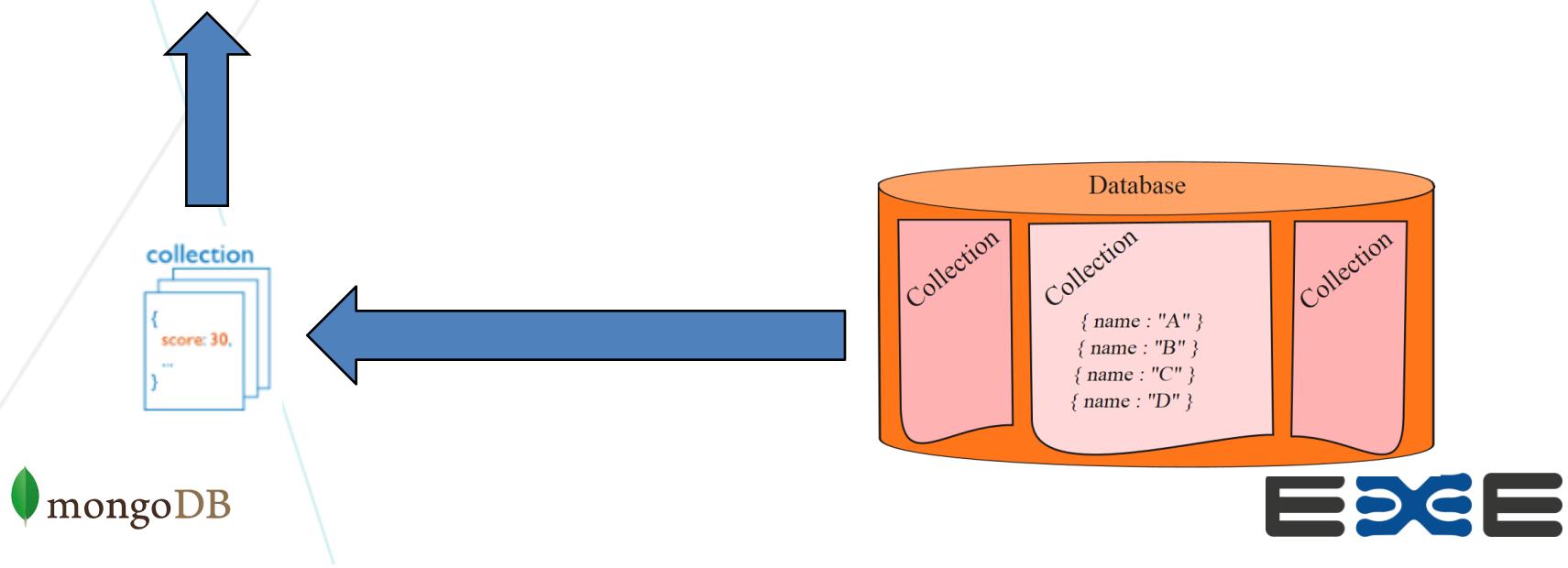


Antes de Indexar



- ¿Qué es lo que hace normalmente la base de datos cuando se hace una consulta?
 - MongoDB debe “mirar” **cada** documento.
 - Es ineficiente porque implica procesar un gran volumen de datos

```
db.users.find( { score: { "$lt" : 30} } )
```



Definición de Índice

► Definición

- Los Índices son estructuras especiales de datos que almacena una parte pequeña de la colección para facilitar su recorrido.

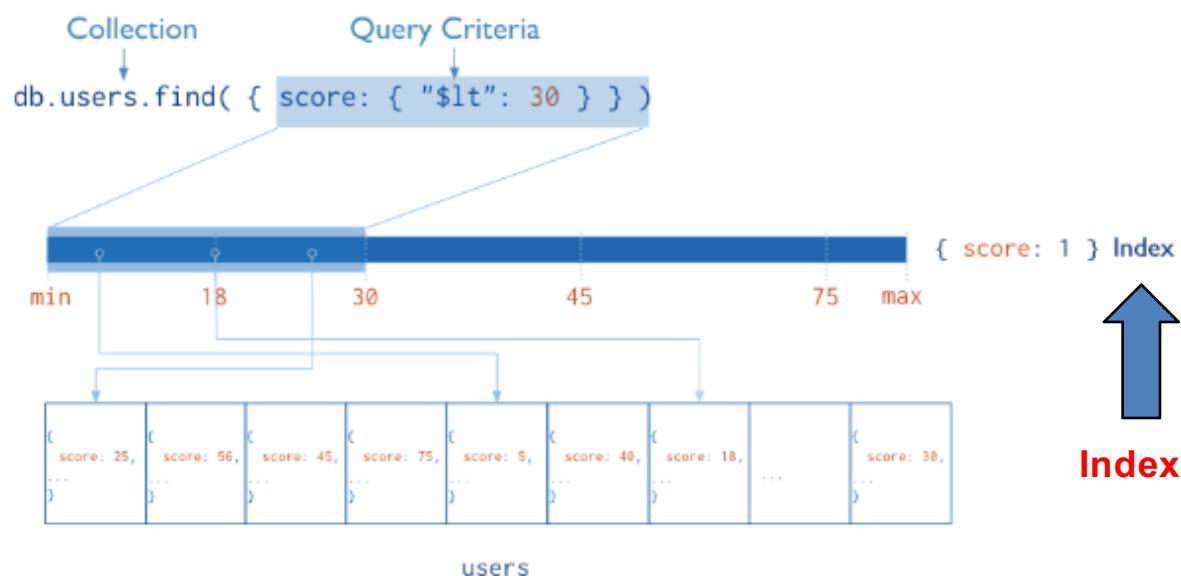
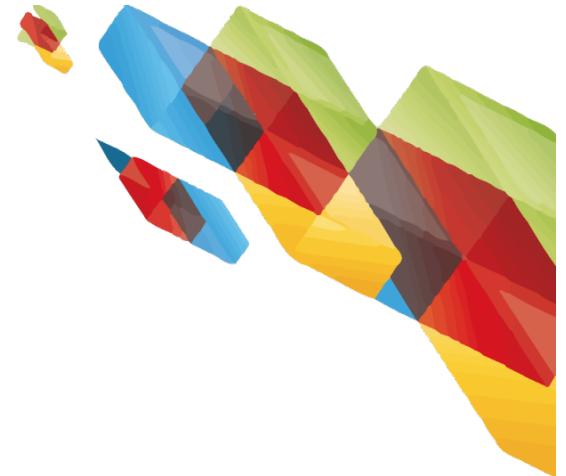


Diagrama de una consulta que utiliza un índice

Índices en MongoDB



► Creación

► db.users.ensureIndex({ score: 1 })

► Mostrar los índices existentes

► db.users.getIndexes()

► Eliminar

► db.users.dropIndex({score: 1})

► Explicar

► db.users.find().explain()

► Devuelve un documento que describe como será resuleta la consulta y que índices utiliza

► Ayuda

► db.users.find().hint({score: 1})

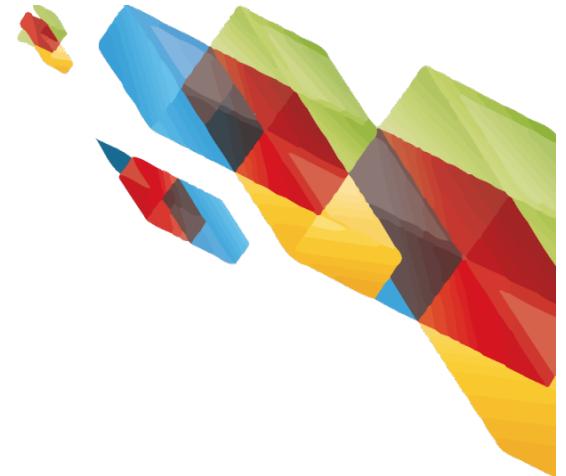
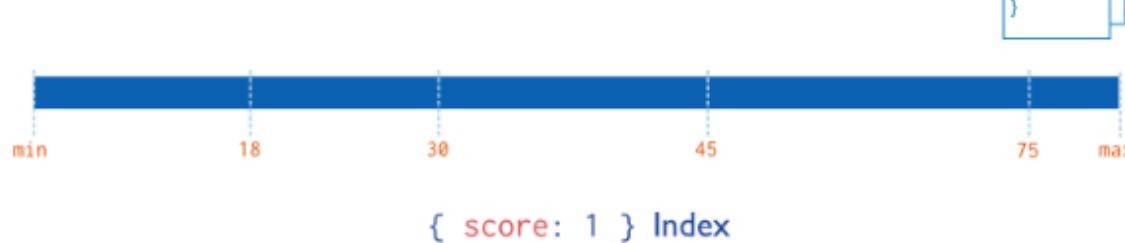
► Esto fuerza a mongoDB a utilizar ese índice

Índices en MongoDB

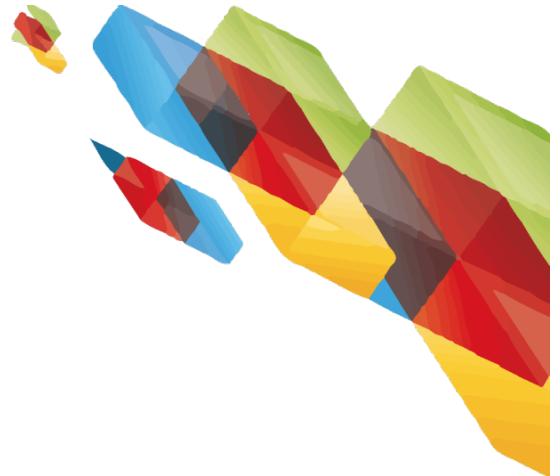
Tipos

- De un solo campo
- De varios campos (Compuesto)
- Multillave
- Índices de un solo campo

```
db.users.ensureIndex( { score: 1 } )
```

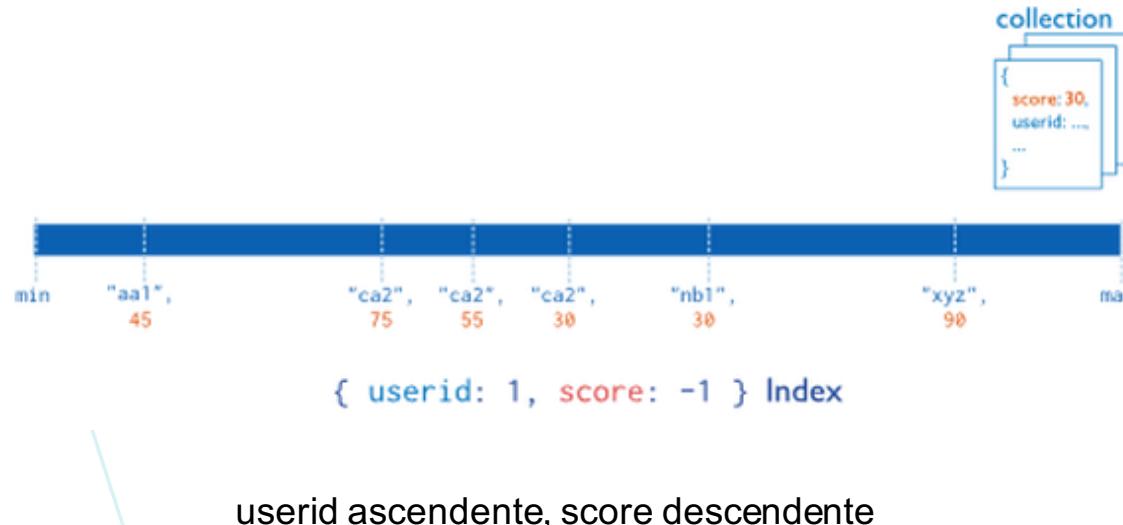


Índices en MongoDB

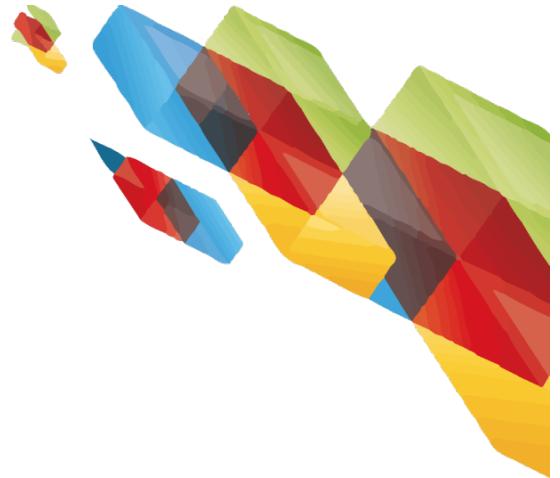


Tipos

- De un solo campo
 - De varios campos (Compuesto)
 - Multillave
-
- **Compuesto**
 - db.users.ensureIndex({ userid:1, score: -1 })

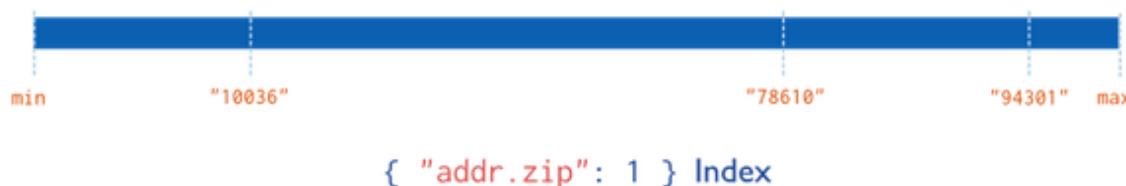
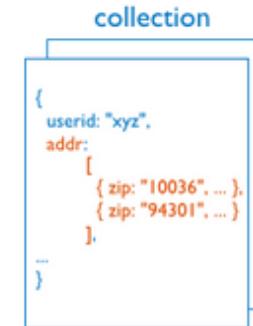


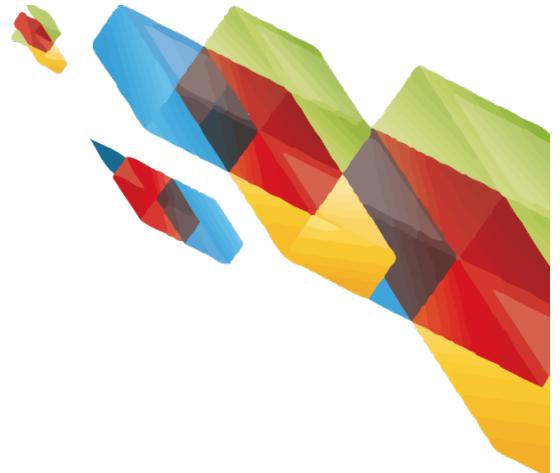
Índices en MongoDB



Types

- De un solo campo
 - De varios campos (Compuesto)
 - Multillave
-
- Multillave
 - db.users.ensureIndex({ addr.zip:1})

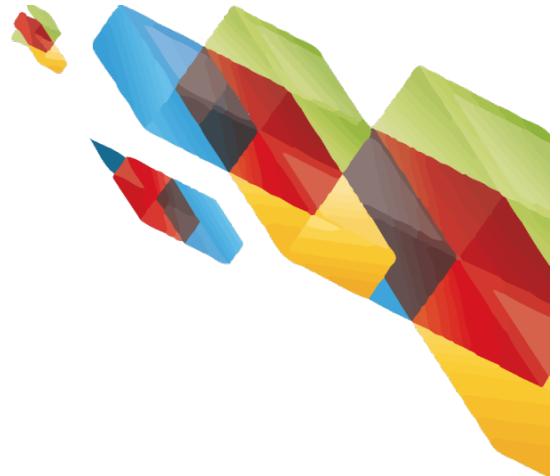




Índices en MongoDB

- Importar Datos
- Crear índices
 - De campo
 - Compuesto
 - Multillave
- Mostrar los índices
- Hint
- Explain
- Comparar sin índices

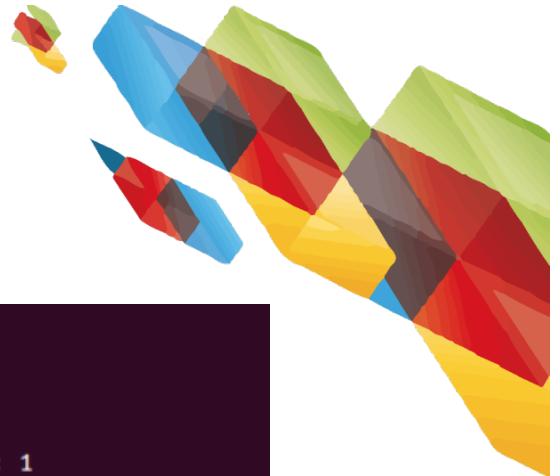
```
> db.zips.find().limit(20)
[{"city": "ACMAR", "loc": [-86.51557, 33.584132], "pop": 6055, "state": "AL", "_id": "35004"}, {"city": "ADAMSVILLE", "loc": [-86.959727, 33.588437], "pop": 10616, "state": "AL", "_id": "35005"}, {"city": "ADGER", "loc": [-87.167455, 33.434277], "pop": 3205, "state": "AL", "_id": "35006"}, {"city": "KEYSTONE", "loc": [-86.812861, 33.236868], "pop": 14218, "state": "AL", "_id": "35007"}, {"city": "NEW SITE", "loc": [-85.951086, 32.941445], "pop": 19942, "state": "AL", "_id": "35010"}, {"city": "ALPINE", "loc": [-86.208934, 33.331165], "pop": 3062, "state": "AL", "_id": "35014"}, {"city": "ARAB", "loc": [-86.489638, 34.328339], "pop": 13650, "state": "AL", "_id": "35016"}, {"city": "BAILEYTON", "loc": [-86.621299, 34.268298], "pop": 1781, "state": "AL", "_id": "35019"}, {"city": "BESSEMER", "loc": [-86.947547, 33.409002], "pop": 40549, "state": "AL", "_id": "35020"}, {"city": "HUEYTOWN", "loc": [-86.999607, 33.414625], "pop": 39677, "state": "AL", "_id": "35023"}, {"city": "BLOUNTSVILLE", "loc": [-86.568628, 34.092937], "pop": 9058, "state": "AL", "_id": "35031"}, {"city": "BREMEN", "loc": [-87.004281, 33.973664], "pop": 3448, "state": "AL", "_id": "35033"}, {"city": "BRENT", "loc": [-87.211387, 32.93567], "pop": 3791, "state": "AL", "_id": "35034"}, {"city": "BRIERFIELD", "loc": [-86.951672, 33.042747], "pop": 1282, "state": "AL", "_id": "35035"}, {"city": "CALERA", "loc": [-86.755987, 33.1098], "pop": 4675, "state": "AL", "_id": "35040"}, {"city": "CENTREVILLE", "loc": [-87.11924, 32.950324], "pop": 4902, "state": "AL", "_id": "35042"}, {"city": "CHELSEA", "loc": [-86.614132, 33.371582], "pop": 4781, "state": "AL", "_id": "35043"}, {"city": "COOSA PINES", "loc": [-86.337622, 33.266928], "pop": 7985, "state": "AL", "_id": "35044"}, {"city": "CLANTON", "loc": [-86.642472, 32.835532], "pop": 13990, "state": "AL", "_id": "35045"}, {"city": "CLEVELAND", "loc": [-86.559355, 33.992106], "pop": 2369, "state": "AL", "_id": "35049"}]
> db.zips.find().count()
29467
```



Índices en MongoDB

- ▶ Importar Datos
- ▶ Crear índices
 - ▶ De campo
 - ▶ Compuesto
 - ▶ Multillave
- ▶ Mostrar los índices
- ▶ Hint
- ▶ Explain
- ▶ Comparar sin índices

```
db.zips.ensureIndex({pop: -1})
db.zips.ensureIndex({state: 1, city: 1})
db.zips.ensureIndex({loc: -1})
```



Índices en MongoDB

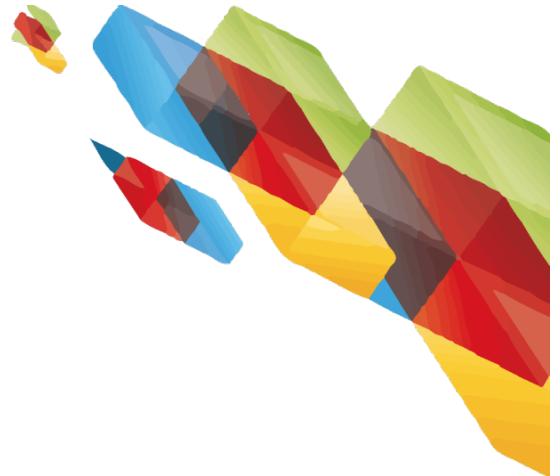
- Importar Datos
- Crear índices
 - De campo
 - Compuesto
 - Multillave
- Mostrar los índices
- Hint
- Explain
- Comparar sin índices



```
> db.zips.getIndexes()
[  
  {  
    "v" : 1,  
    "key" : {  
      "_id" : 1  
    },  
    "ns" : "blog.zips",  
    "name" : "_id_"  
  },  
  {  
    "v" : 1,  
    "key" : {  
      "pop" : 1  
    },  
    "ns" : "blog.zips",  
    "name" : "pop_1"  
  },  
  {  
    "v" : 1,  
    "key" : {  
      "state" : 1,  
      "city" : 1  
    },  
    "ns" : "blog.zips",  
    "name" : "state_1_city_1"  
  },  
  {  
    "v" : 1,  
    "key" : {  
      "loc" : 1  
    },  
    "ns" : "blog.zips",  
    "name" : "loc_1"  
  }]
```

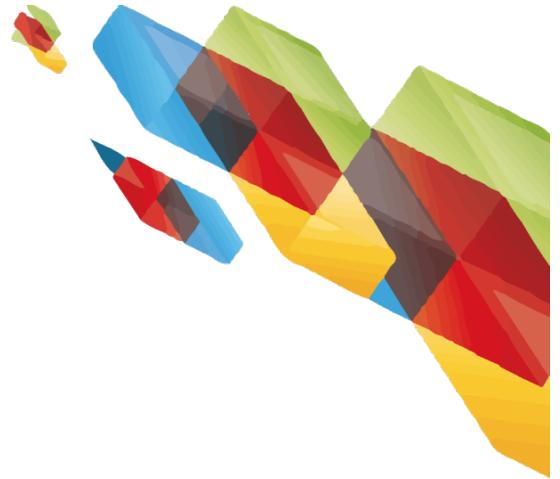


Índices en MongoDB



- Importar Datos
- Crear índices
 - De campo
 - Compuesto
 - Multillave
- Mostrar los índices
- Hint
- Explain
- Comparar sin índices

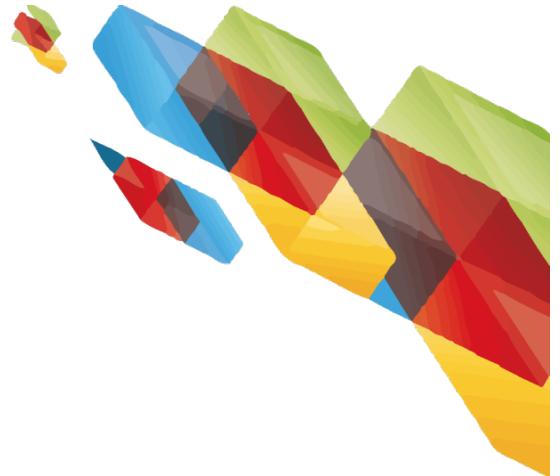
```
> db.zips.find().limit(20).hint({pop: -1})
{ "city" : "CHICAGO", "loc" : [ -87.7157, 41.849015 ], "pop" : 112047, "state" : "IL", "_id" : "60623" }
{ "city" : "BROOKLYN", "loc" : [ -73.956985, 40.646694 ], "pop" : 111396, "state" : "NY", "_id" : "11226" }
{ "city" : "NEW YORK", "loc" : [ -73.958805, 40.768476 ], "pop" : 106564, "state" : "NY", "_id" : "10021" }
{ "city" : "NEW YORK", "loc" : [ -73.968312, 40.797466 ], "pop" : 100027, "state" : "NY", "_id" : "10025" }
{ "city" : "BELL GARDENS", "loc" : [ -118.17205, 33.969177 ], "pop" : 99568, "state" : "CA", "_id" : "90201" }
{ "city" : "CHICAGO", "loc" : [ -87.556012, 41.725743 ], "pop" : 98612, "state" : "IL", "_id" : "60617" }
{ "city" : "LOS ANGELES", "loc" : [ -118.258189, 34.007856 ], "pop" : 96074, "state" : "CA", "_id" : "90011" }
{ "city" : "CHICAGO", "loc" : [ -87.704322, 41.920903 ], "pop" : 95971, "state" : "IL", "_id" : "60647" }
{ "city" : "CHICAGO", "loc" : [ -87.624277, 41.693443 ], "pop" : 94317, "state" : "IL", "_id" : "60628" }
{ "city" : "NORWALK", "loc" : [ -118.081767, 33.90564 ], "pop" : 94188, "state" : "CA", "_id" : "90650" }
{ "city" : "CHICAGO", "loc" : [ -87.654251, 41.741119 ], "pop" : 92005, "state" : "IL", "_id" : "60620" }
{ "city" : "CHICAGO", "loc" : [ -87.706936, 41.778149 ], "pop" : 91814, "state" : "IL", "_id" : "60629" }
{ "city" : "CHICAGO", "loc" : [ -87.653279, 41.809721 ], "pop" : 89762, "state" : "IL", "_id" : "60689" }
{ "city" : "CHICAGO", "loc" : [ -87.704214, 41.946401 ], "pop" : 88377, "state" : "IL", "_id" : "60618" }
{ "city" : "JACKSON HEIGHTS", "loc" : [ -73.878551, 40.740388 ], "pop" : 88241, "state" : "NY", "_id" : "11373" }
{ "city" : "ARLETA", "loc" : [ -118.420692, 34.258081 ], "pop" : 88114, "state" : "CA", "_id" : "91331" }
{ "city" : "BROOKLYN", "loc" : [ -73.914483, 40.662474 ], "pop" : 87079, "state" : "NY", "_id" : "11212" }
{ "city" : "SOUTH GATE", "loc" : [ -118.201349, 33.94617 ], "pop" : 87026, "state" : "CA", "_id" : "90280" }
{ "city" : "RIDGEWOOD", "loc" : [ -73.896122, 40.703613 ], "pop" : 85732, "state" : "NY", "_id" : "11385" }
{ "city" : "BRONX", "loc" : [ -73.871242, 40.873671 ], "pop" : 85710, "state" : "NY", "_id" : "10467" }
```



Índices en MongoDB

- Importar Datos
- Crear índices
 - De campo
 - Compuesto
 - Multillave
- Mostrar los índices
- Hint
- Explain
- Comparar sin índices

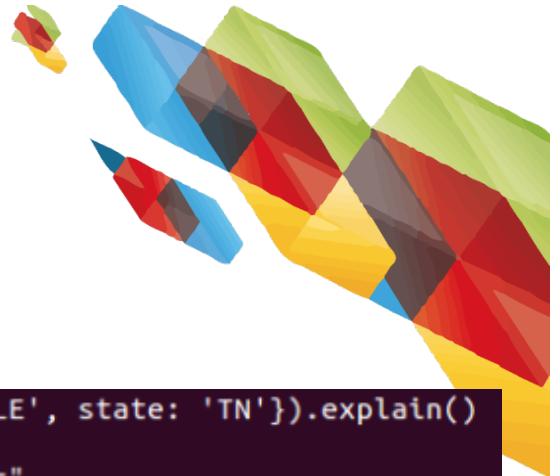
```
> db.zips.find().limit(20).hint({state: 1, city: 1})  
[{"city": "98791", "loc": [-176.310048, 51.938901], "pop": 5345, "state": "AK", "_id": "98791"}, {"city": "AKHIOK", "loc": [-152.500169, 57.781967], "pop": 13309, "state": "AK", "_id": "99615"}, {"city": "AKIACHAK", "loc": [-161.39233, 60.891854], "pop": 481, "state": "AK", "_id": "99551"}, {"city": "AKIAK", "loc": [-161.199325, 60.890632], "pop": 285, "state": "AK", "_id": "99552"}, {"city": "AKUTAN", "loc": [-165.785368, 54.143012], "pop": 589, "state": "AK", "_id": "99553"}, {"city": "ALAKANUK", "loc": [-164.60228, 62.746967], "pop": 1186, "state": "AK", "_id": "99554"}, {"city": "ALEKNAGIK", "loc": [-158.619882, 59.269688], "pop": 185, "state": "AK", "_id": "99555"}, {"city": "ALLAKAKET", "loc": [-152.712155, 66.543197], "pop": 170, "state": "AK", "_id": "99720"}, {"city": "AMBLER", "loc": [-156.455652, 67.46951], "pop": 8, "state": "AK", "_id": "99786"}, {"city": "ANAKTUVUK PASS", "loc": [-151.679005, 68.11878], "pop": 260, "state": "AK", "_id": "99721"}, {"city": "ANCHORAGE", "loc": [-149.876077, 61.211571], "pop": 14436, "state": "AK", "_id": "99501"}, {"city": "ANCHORAGE", "loc": [-150.093943, 61.096163], "pop": 15891, "state": "AK", "_id": "99502"}, {"city": "ANCHORAGE", "loc": [-149.893844, 61.189953], "pop": 12534, "state": "AK", "_id": "99503"}, {"city": "ANCHORAGE", "loc": [-149.74467, 61.203696], "pop": 32383, "state": "AK", "_id": "99504"}, {"city": "ANCHORAGE", "loc": [-149.828912, 61.153543], "pop": 20128, "state": "AK", "_id": "99507"}, {"city": "ANCHORAGE", "loc": [-149.810085, 61.205959], "pop": 29857, "state": "AK", "_id": "99508"}, {"city": "ANCHORAGE", "loc": [-149.897401, 61.119381], "pop": 17094, "state": "AK", "_id": "99515"}, {"city": "ANCHORAGE", "loc": [-149.779998, 61.10541], "pop": 18356, "state": "AK", "_id": "99516"}, {"city": "ANCHORAGE", "loc": [-149.936111, 61.190136], "pop": 15192, "state": "AK", "_id": "99517"}, {"city": "ANCHORAGE", "loc": [-149.886571, 61.154862], "pop": 8116, "state": "AK", "_id": "99518"}]
```



Índices en MongoDB

- Importar Datos
- Crear índices
 - De campo
 - Compuesto
 - Multillave
- Mostrar los índices
- Hint
- Explain
- Comparar sin índices

```
> db.zips.find().limit(20).hint({loc: -1})
[{"city": "BARRROW", "loc": [-156.817469, 71.234637], "pop": 3696, "state": "AK", "_id": "99723"}, {"city": "WAINWRIGHT", "loc": [-160.012532, 70.620064], "pop": 492, "state": "AK", "_id": "99782"}, {"city": "NUIQSUT", "loc": [-150.997119, 70.192737], "pop": 354, "state": "AK", "_id": "99789"}, {"city": "PRUDHOE BAY", "loc": [-148.559636, 70.070057], "pop": 153, "state": "AK", "_id": "99734"}, {"city": "KAKTOVIK", "loc": [-143.631329, 70.042889], "pop": 245, "state": "AK", "_id": "99747"}, {"city": "POINT LAY", "loc": [-162.906148, 69.705626], "pop": 139, "state": "AK", "_id": "99759"}, {"city": "POINT HOPE", "loc": [-166.72618, 68.312058], "pop": 640, "state": "AK", "_id": "99766"}, {"city": "ANAKTUVUK PASS", "loc": [-151.679005, 68.11878], "pop": 260, "state": "AK", "_id": "99721"}, {"city": "ARCTIC VILLAGE", "loc": [-145.423115, 68.077395], "pop": 107, "state": "AK", "_id": "99722"}, {"city": "KIVALINA", "loc": [-163.733617, 67.665859], "pop": 689, "state": "AK", "_id": "99750"}, {"city": "AMBLER", "loc": [-156.455652, 67.46951], "pop": 8, "state": "AK", "_id": "99786"}, {"city": "KIANA", "loc": [-158.152204, 67.18026], "pop": 349, "state": "AK", "_id": "99749"}, {"city": "BETTLES FIELD", "loc": [-151.062414, 67.100495], "pop": 156, "state": "AK", "_id": "99726"}, {"city": "VENETIE", "loc": [-146.413723, 67.010446], "pop": 184, "state": "AK", "_id": "99781"}, {"city": "NOATAK", "loc": [-160.509453, 66.97553], "pop": 395, "state": "AK", "_id": "99761"}, {"city": "SHUNGNAK", "loc": [-157.613496, 66.958141], "pop": 0, "state": "AK", "_id": "99773"}, {"city": "KOBUK", "loc": [-157.066864, 66.912253], "pop": 306, "state": "AK", "_id": "99751"}, {"city": "KOTZEBUE", "loc": [-162.126493, 66.846459], "pop": 3347, "state": "AK", "_id": "99752"}, {"city": "NOORVIK", "loc": [-161.044132, 66.836353], "pop": 534, "state": "AK", "_id": "99763"}, {"city": "CHALKYITSIK", "loc": [-143.638121, 66.719], "pop": 99, "state": "AK", "_id": "99788"}]
```



Índices en MongoDB

- ▶ Importar Datos
- ▶ Crear índices
 - ▶ De campo
 - ▶ Compuesto
 - ▶ Multillave
- ▶ Mostrar los índices
- ▶ Hint
- ▶ Explain
- ▶ Comparar sin índices

```
> db.zips.find({city: 'NASHVILLE', state: 'TN'}).explain()
{
    "cursor" : "BasicCursor",
    "isMultiKey" : false,
    "n" : 19,
    "nscannedObjects" : 29467,
    "nscanned" : 29467,
    "nscannedObjectsAllPlans" : 29467,
    "nscannedAllPlans" : 29467,
    "scanAndOrder" : false,
    "indexOnly" : false,
    "nYields" : 0,
    "nChunkSkips" : 0,
    "millis" : 33,
    "indexBounds" : {
        },
    "server" : "g:27017"
}
```

Índices en MongoDB

- Importar Datos
- Crear índices
 - De campo
 - Compuesto
 - Multillave
- Mostrar los índices
- Hint
- Explain
- Comparar sin índices

```
> db.zips.dropIndexes()
{
  "nIndexesWas" : 4,
  "msg" : "non-_id indexes dropped for collection",
  "ok" : 1
}
> db.zips.find({city: 'NASHVILLE', state: 'TN'}).explain()
{
  "cursor" : "BasicCursor",
  "isMultiKey" : false,
  "n" : 19,
  "nscannedObjects" : 29467,
  "nscanned" : 29467,
  "nscannedObjectsAllPlans" : 29467,
  "nscannedAllPlans" : 29467,
  "scanAndOrder" : false,
  "indexOnly" : false,
  "nYields" : 0,
  "nChunkSkips" : 0,
  "millis" : 33,
  "indexBounds" : {
    },
  "server" : "g:27017"
}
```

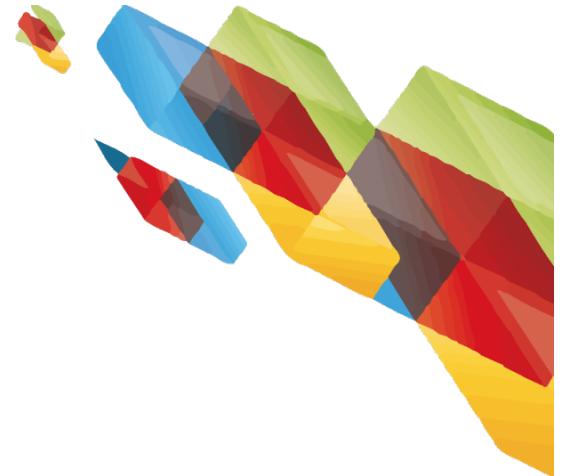
Sin índice

```
> db.zips.find({city: 'NASHVILLE', state: 'TN'}).explain()
{
  "cursor" : "BtreeCursor state_1_city_1",
  "isMultiKey" : false,
  "n" : 19,
  "nscannedObjects" : 19,
  "nscanned" : 19,
  "nscannedObjectsAllPlans" : 19,
  "nscannedAllPlans" : 19,
  "scanAndOrder" : false,
  "indexOnly" : false,
  "nYields" : 0,
  "nChunkSkips" : 0,
  "millis" : 0,
  "indexBounds" : {
    "state" : [
      [
        "TN",
        "TN"
      ]
    ],
    "city" : [
      [
        "NASHVILLE",
        "NASHVILLE"
      ]
    ],
  },
  "server" : "g:27017"
}
```

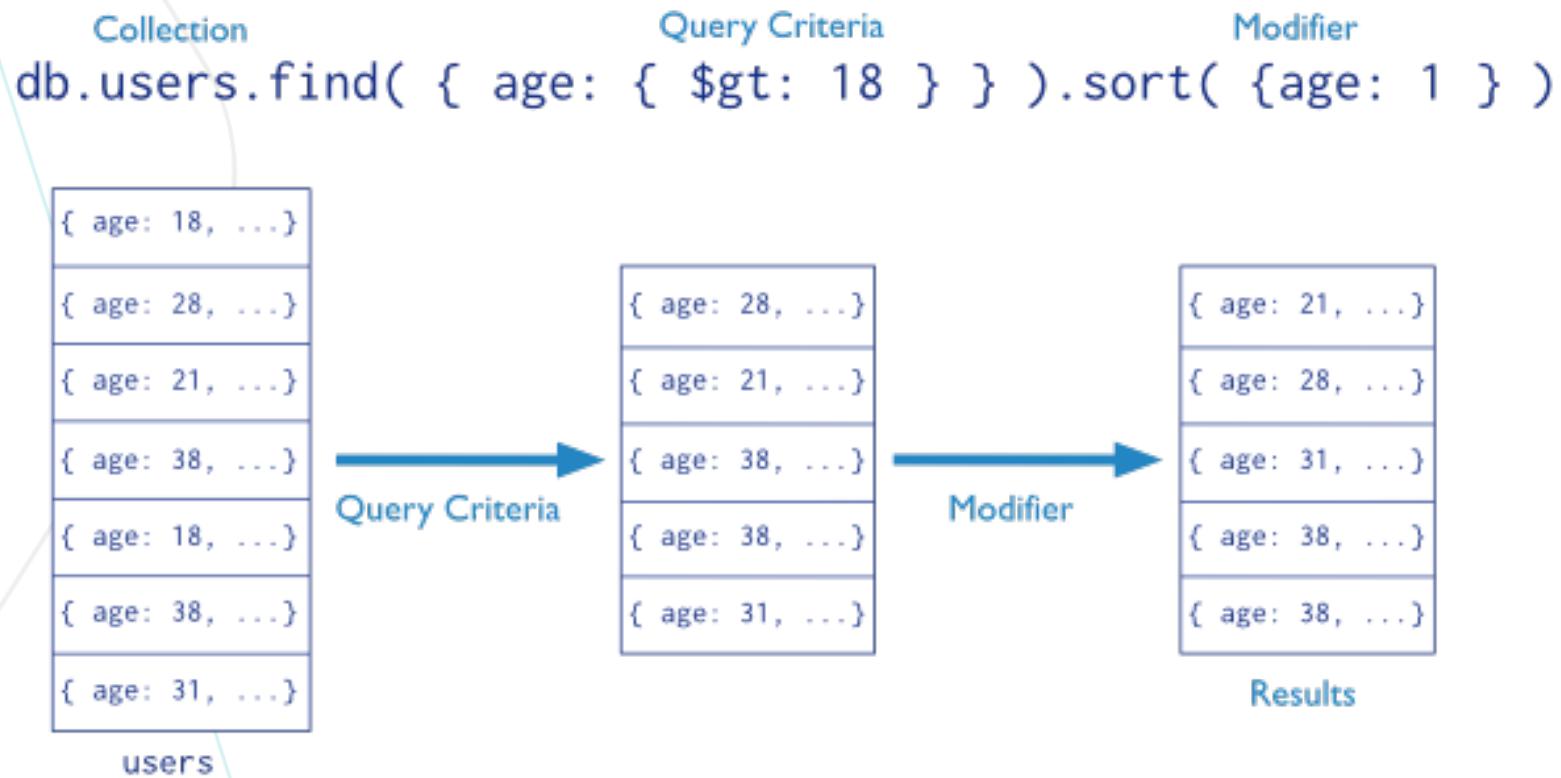
Con índice



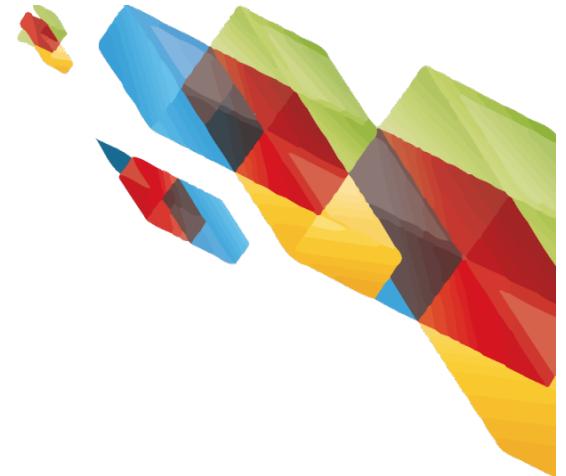
Revisitando CRUD. Lectura



- Read

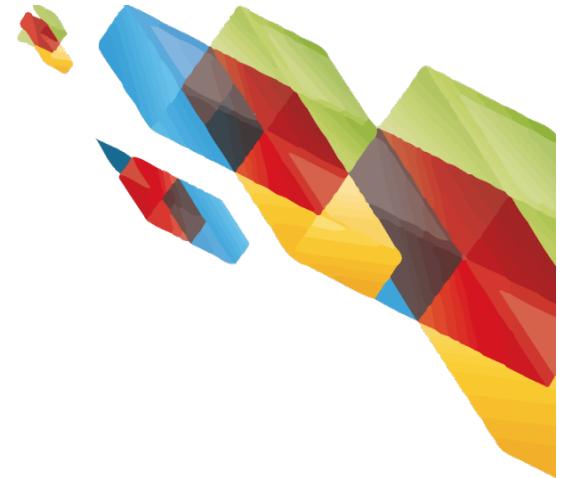


Lectura en MongoDB



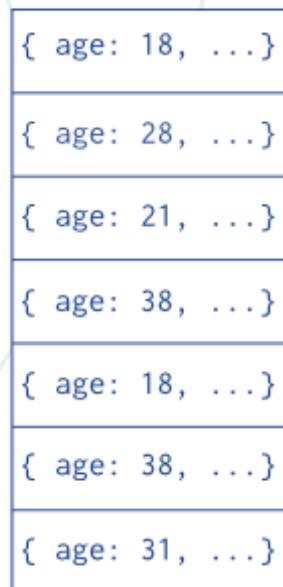
- Operaciones de Lectura.
- Los Queries son las operaciones core en MongoDB.
- Los queries retornan cursorres que son objetos iterables que permiten obtener el resultado completo de una consulta
- El análisis de las consultas mejora el desempeño y los tiempos de respuesta de las consultas.
- Las consultas distribuidas describen como los clusters en Sharding o en Réplica afectan las operaciones de lectura

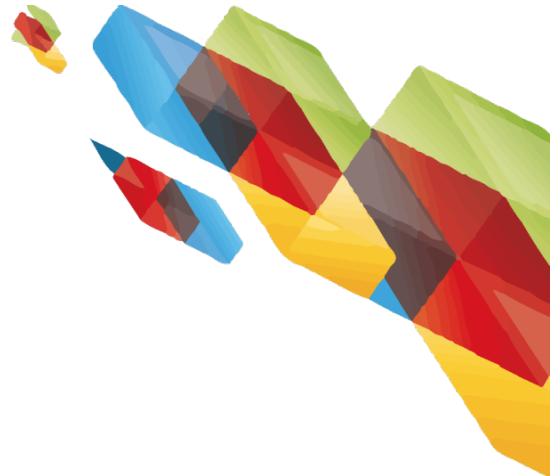
Lectura en MongoDB



- Operaciones de lectura

```
Collection           Query Criteria           Projection  
db.users.find( { age: 18 }, { name: 1, _id: 0 } )
```





Lectura en MongoDB

- Proyecciones

- Por omisión , el campo `_id` field está incluido
- Para suprimirlo se debe incluir `_id: 0` en el documento de proyección
- Para arreglos existen las funciones `$elemMatch`, `$slice`, y `$`.

Excluir un campo:

```
db.records.find( { "user_id": { $lt: 42 } }, { "history": 0 } )
```

Dos campos y `_id`:

```
db.records.find( { "user_id": { $lt: 42 } }, { "name": 1, "email": 1 } )
```

Dos campos:

```
db.records.find( { "user_id": { $lt: 42 } }, { "_id": 0, "name": 1, "email": 1 } )
```



Lectura en MongoDB

- Cursos

- Por omisión iteran de 20 documentos
- El servidor cierra el cursor después de 10 minutos de inactividad o al haberlo recorrido completo.
- El cursor puede retornar un documento más de una vez (Nivel de aislación)
- MongoDB retorna los resultados en batches. El tamaño del batch no debe exceder el tamaño máximo de un documento BSON. Lo habitual es que el primer batch retorne 101 documentos o los documentos suficientes para pasar 1MB. Los batches siguientes son de 4MB. Para cambiar este comportamiento se puede usar batchSize() o limit().

Cursor sin timeout:

```
var myCursor = db.inventory.find().addOption(DBQuery.Option.noTimeout);
var myFirstDocument = myCursor.hasNext() ? myCursor.next() : null;
myCursor objsLeftInBatch();
```

BatchSize:

```
db.inventory.find().batchSize(10)
```

 mongoDB





Lectura en MongoDB

- Cursos

- El método db.serverStatus() retorna un documento que contiene las métricas del servidor .
- Las métricas se encuentran en el campo metrics. Dentro de el, en el campo cursor, encontramos la cantidad de cursos abiertos

```
db.serverStatus().metrics.cursor
```

```
{  
    "timedOut" : <number>  
    "open" : {  
        "noTimeout" : <number>,  
        "pinned" : <number>,  
        "total" : <number>  
    }  
}
```

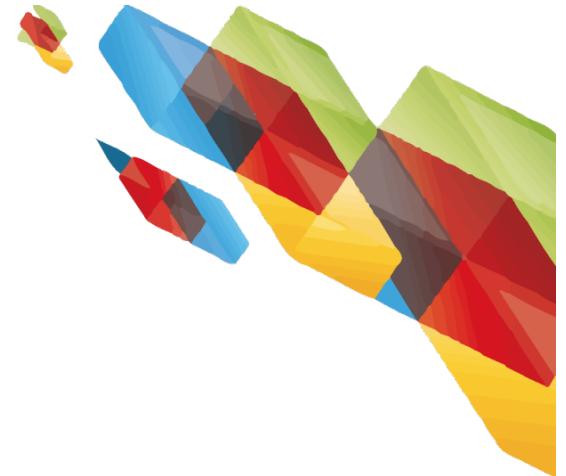


Lectura en MongoDB



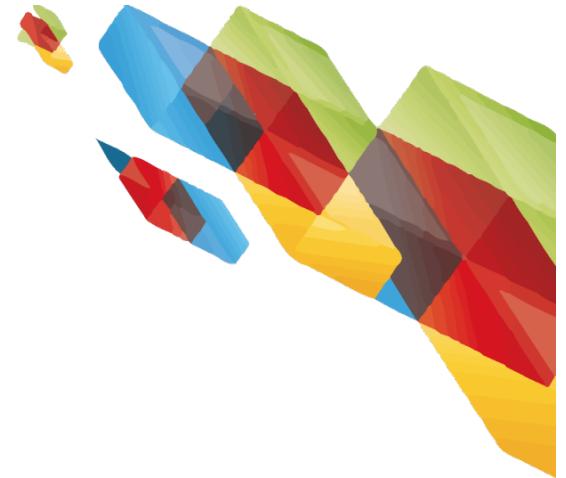
- Planificación y optimización de Queries
 - El optimizador de consultas escoge el plan más eficiente de acuerdo a los índices.
 - El sistema utiliza esos planes para la resolución de consultas. Solo se guardan los planes si existe más de un plan.
 - El optimizar ocasionalmente reevalúa los planes para asegurar el óptimo.
 - Para crear un plan el optimizado ejecuta la consulta utilizando los índices candidatos en paralelo y chequea los resultados comunes (Si el resultado es ordenado, esto no es necesario).

Lectura en MongoDB



- Planificación y optimización de queries.
- A medida que cambian las colecciones en el tiempo, el optimizador reevalúa los planes.
- Los eventos que gatillan la reevaluación son:
 - La colección ha efectuado o recibido más de 1000 operaciones de escritura.
 - El índice es reconstruido.
 - Se agrega o se saca un índice
 - Se reinicia el proceso mongod

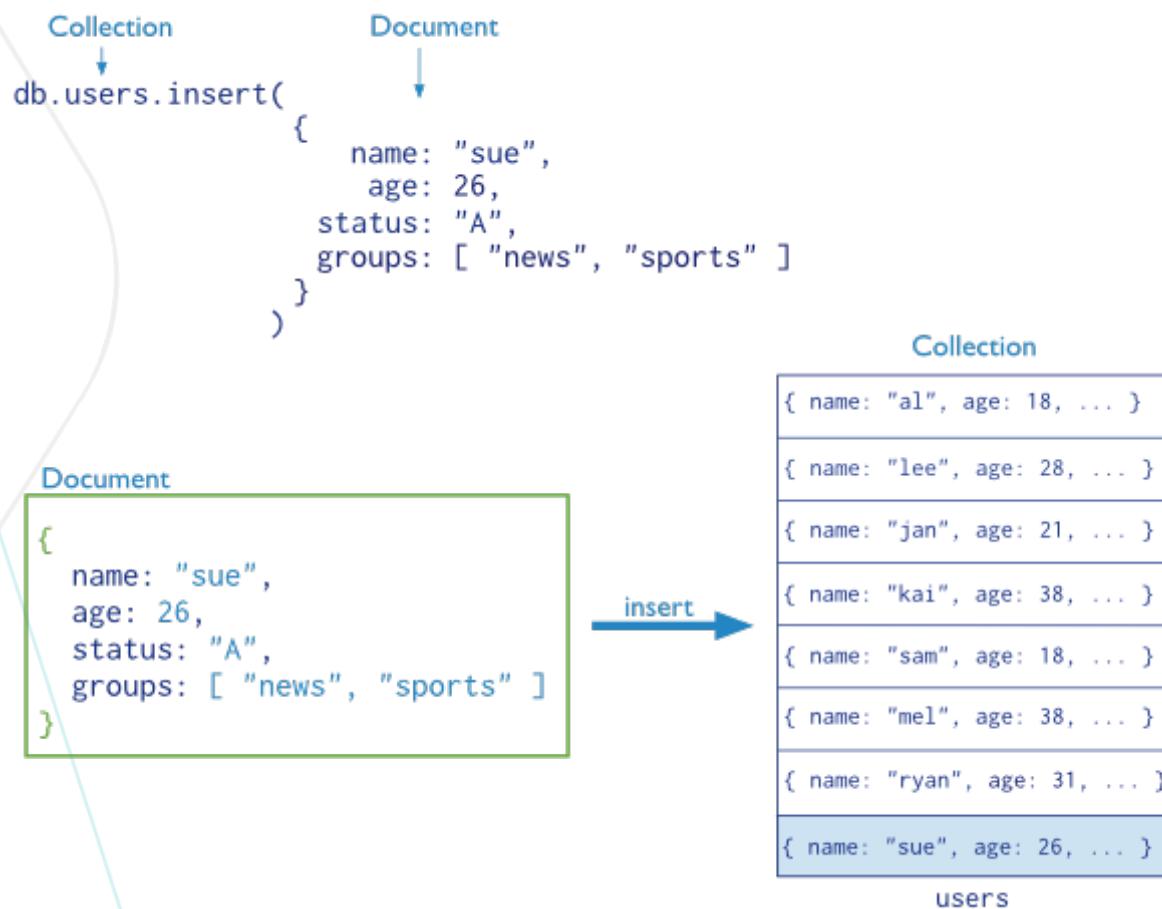
CRUD FORMAL



```
db.people.find( { name: "John Doe",  
zipcode: { $gt: "63000" } } )  
.hint( { zipcode: 1 } )  
.explain("executionStats")
```

Escritura en MongoDB

- Write



Escritura en MongoDB



- Memory mapped Files

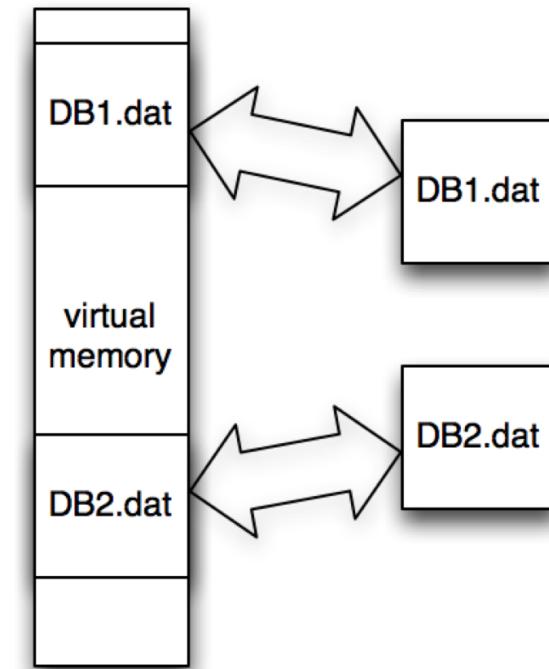
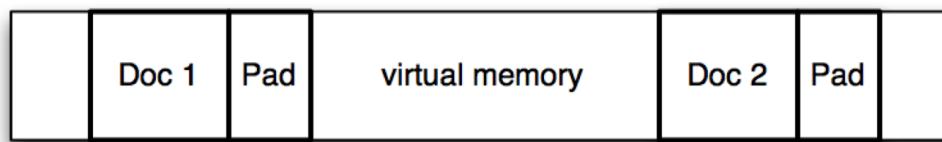
```
use mydb
```

```
db.serverStatus().extra_info.page_faults
```

- Padding

```
use mydb
```

```
db.my_collection.stats()
```



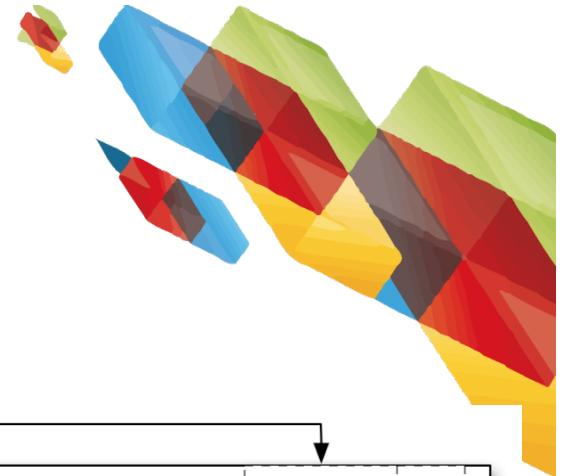


Escritura en MongoDB

- Para compactar

```
db.runCommand ( { compact: '<collection>', paddingFactor: 1.1 } )  
db.runCommand ( { compact: '<collection>', paddingBytes: 100 } )  
db.currentOp()
```
- Por omisión: 1.0 Mínimo: 1.0 (no padding), Máximo: 4.0
- Recordar que compact bloquea la base de datos

Escritura en MongoDB



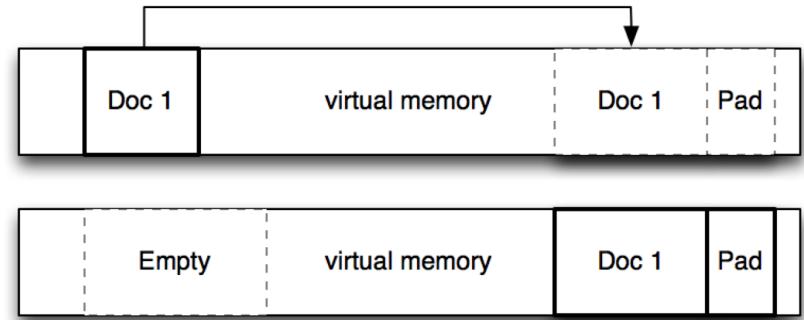
- Fragmentación

- Al mover documentos quedan hoyos
- MongoDB trata de reutilizarlos
- La fragmentación se ve reflejada en la RAM

```
use mydb
var s = db.my_collection.stats()
var frag = s.storageSize / (s.size + s.totalIndexSize)
```

- >1 tenemos fragmentación

- compact, es un comando para eliminar la fragmentación. Requiere de tener el servidor offline
- **usePowerOf2Sizes** es una opción que permite a MongoDB utilizar la memoria en potencias de 2 (128 bytes, 256 bytes, 512 bytes, 1024 bytes etc.). Hay menos chanches de hoyos ya que los tamaños son estándares, pero aumenta el desperdicio de espacio. Un documento de 257 bytes ocupa 512 bytes. Es la opción por omisión



Escritura en MongoDB

- Insert

```
db.users.insert ( ← collection
  {
    name: "sue", ← field: value
    age: 26, ← field: value
    status: "A" ← field: value
  }
)
```

} document

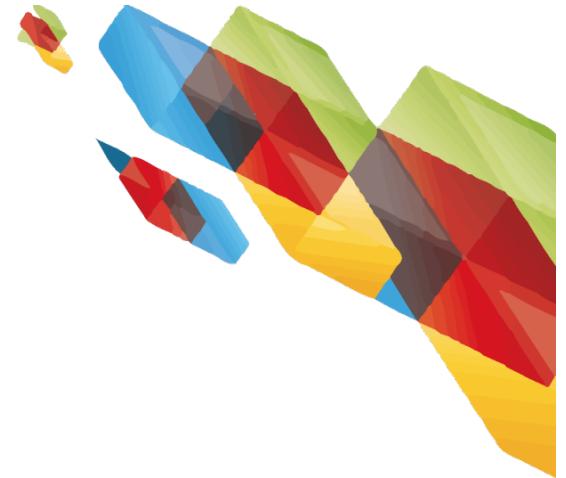
- Update

```
db.users.update( ← collection
  { age: { $gt: 18 } }, ← update criteria
  { $set: { status: "A" } }, ← update action
  { multi: true } ← update option
)
```

- Remove

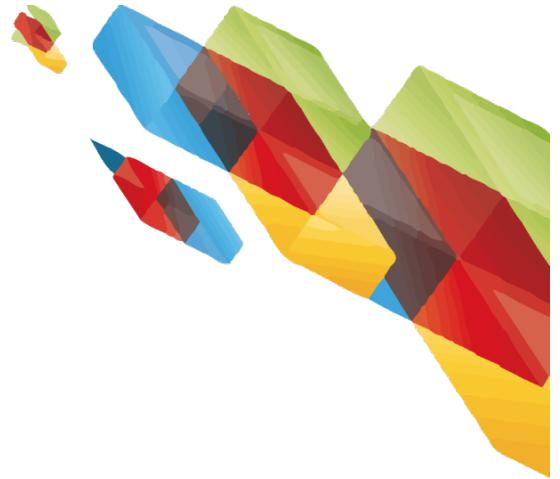
```
db.users.remove( ← collection
  { status: "D" } ← remove criteria
)
```

Escritura en MongoDB



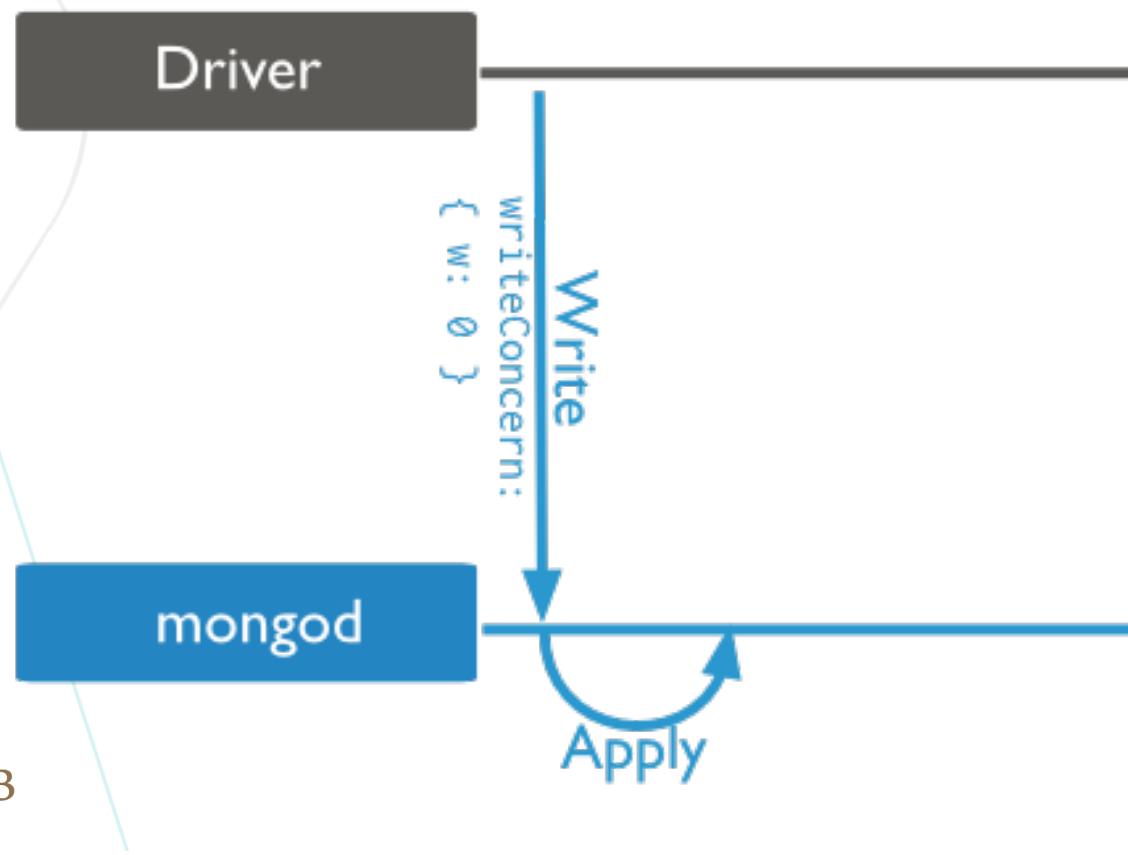
- WriteConcern Levels
- Establecen el modo en que MongoDB garantiza y reporta el éxito de una operación de escritura.
- Cuando se inserta, actualiza o borra con un nivel bajo de WriteConcern, las operaciones retornan más rápido. Pero en casos de fallas estas operaciones podrían retornar sin haber quedado almacenadas de manera persistente.
- Con WriteConcerns más estrictos o fuertes los clientes deben esperar a que MongoDB confirme las operaciones de escritura.

Escritura en MongoDB

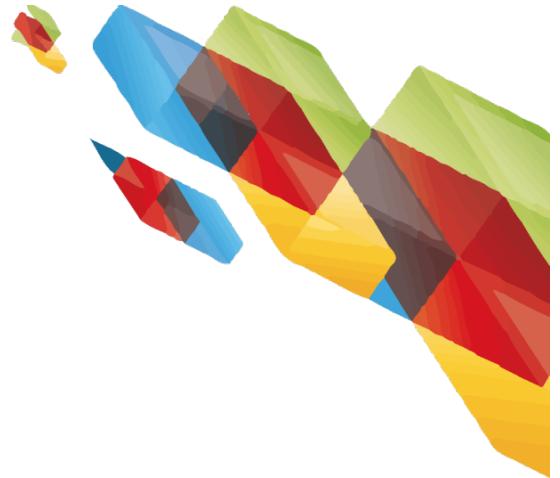


- WriteConcern Unacknowledged

- Solía ser el comportamiento por omisión

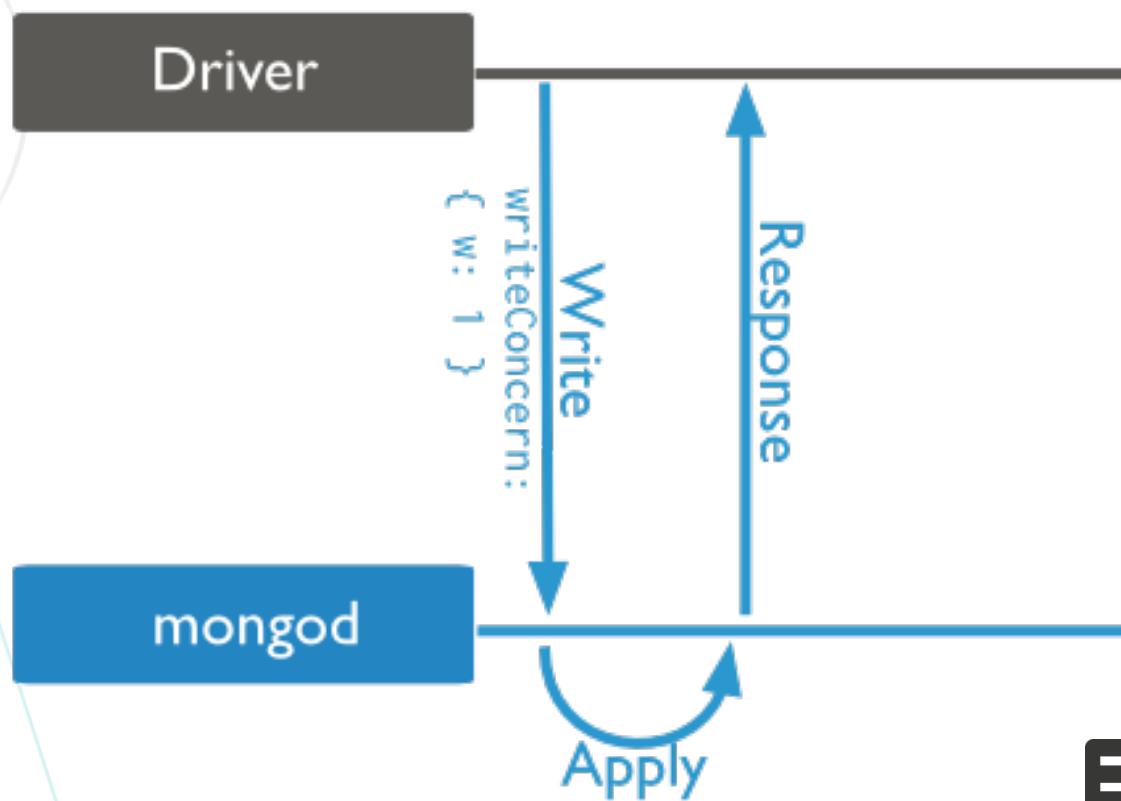


Escritura en MongoDB

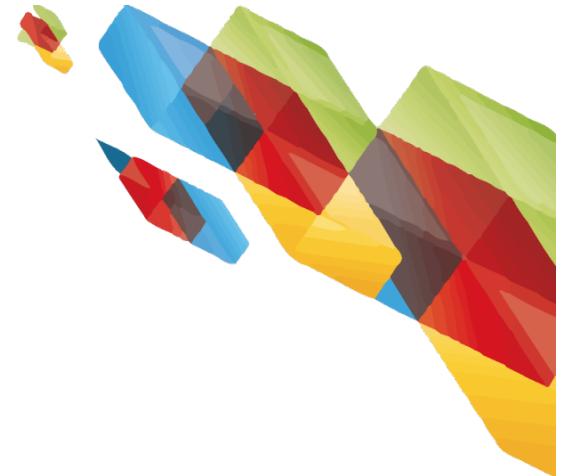


- WriteConcern Acknowledged

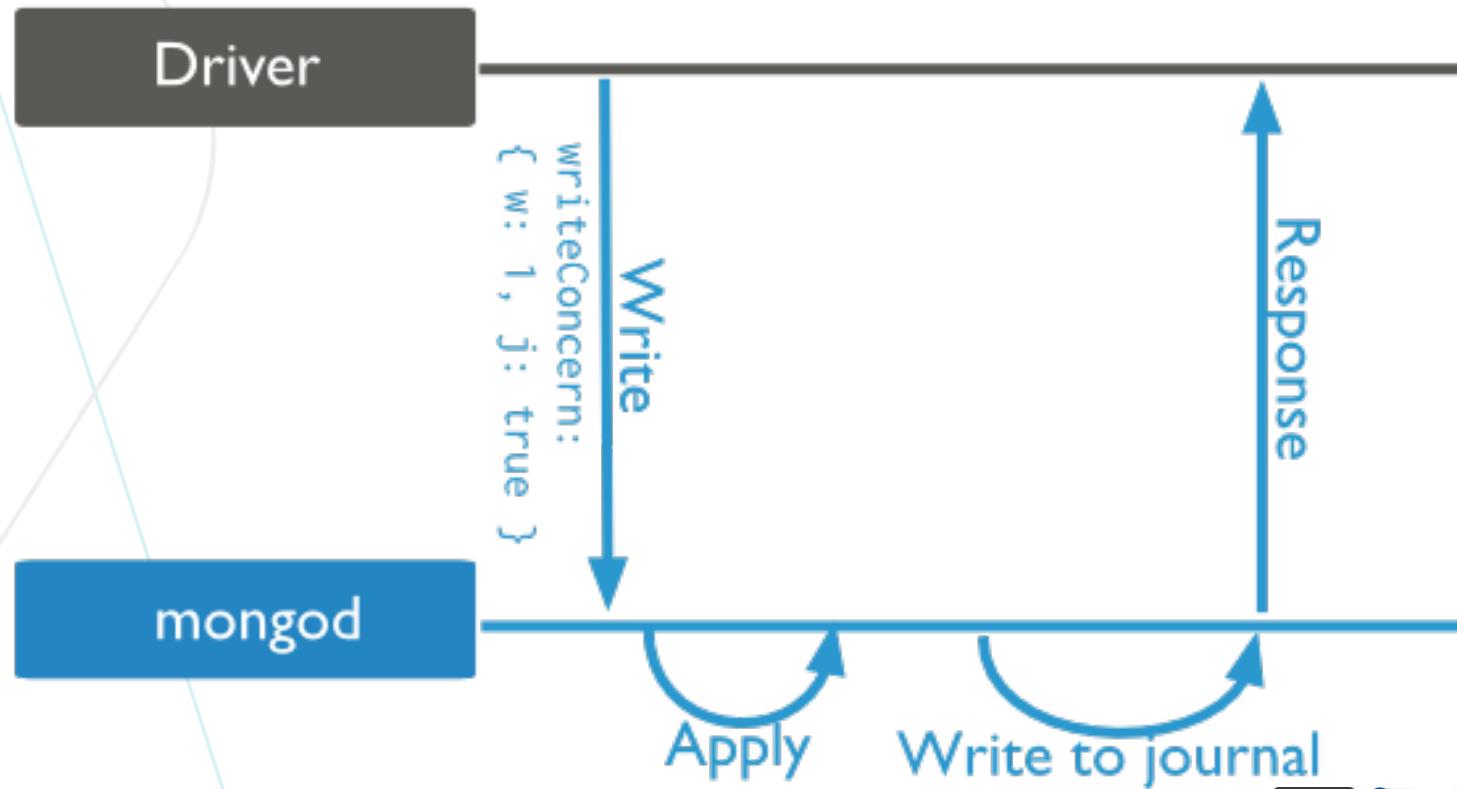
- Comportamiento por omisión



Escritura en MongoDB

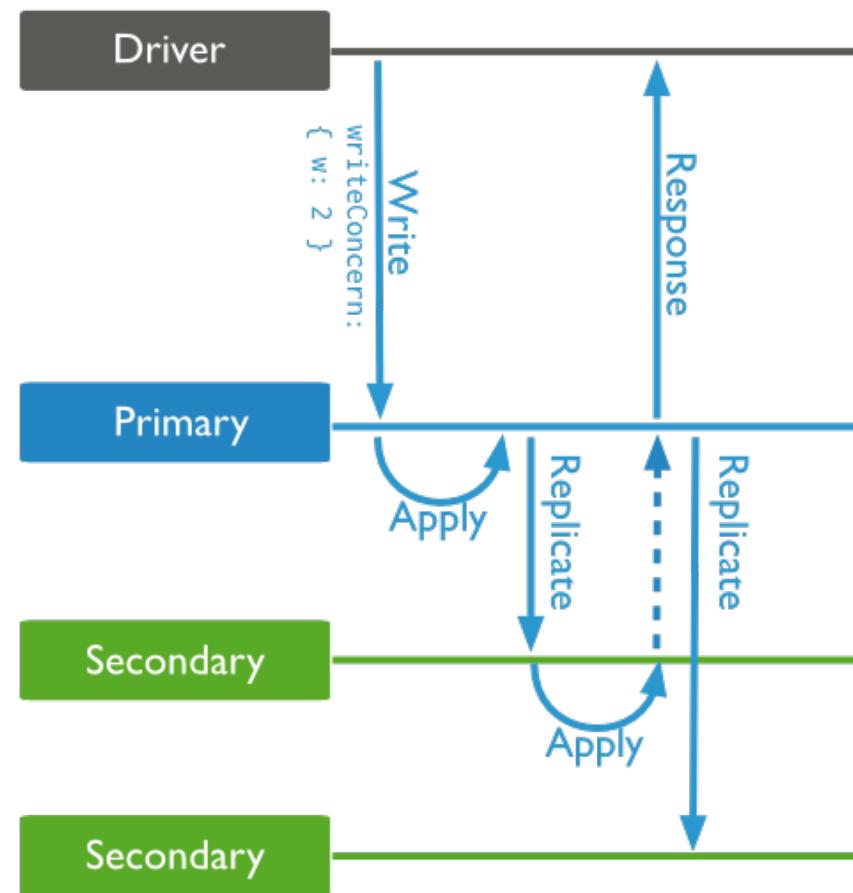


- WriteConcern Jounaled



Escritura en MongoDB

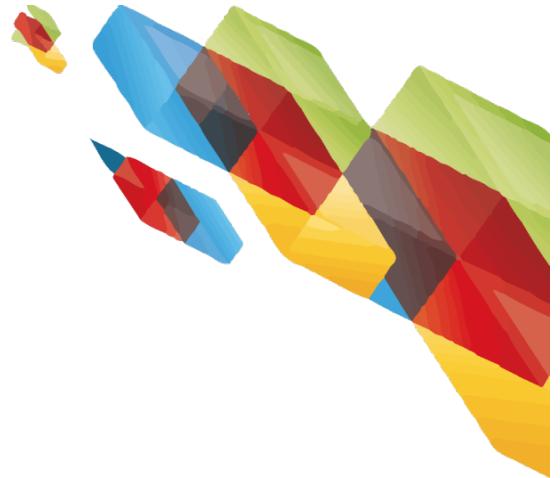
- WriteConcern Replica Acknowledged



Escritura en MongoDB



- Solid state drives (SSDs) registran en MongodB un comportamiento de hasta 100 veces mejor que los discos tradicionales para cargas de trabajo aleatorias



Escritura en MongoDB

- Bulk

- Bulk.insert()
- Bulk.find()
- Bulk.find.upsert()
- Bulk.find.update()
- Bulk.find.updateOne()
- Bulk.find.replaceOne()
- Bulk.find.remove()
- Bulk.find.removeOne()

```
var bulk = db.items.initializeUnorderedBulkOp();
bulk.insert( { _id: 1, item: "abc123", status: "A", soldQty: 5000 } );
bulk.insert( { _id: 2, item: "abc456", status: "A", soldQty: 150 } );
bulk.insert( { _id: 3, item: "abc789", status: "P", soldQty: 0 } );
bulk.execute( { w: "majority", wtimeout: 5000 } );
```

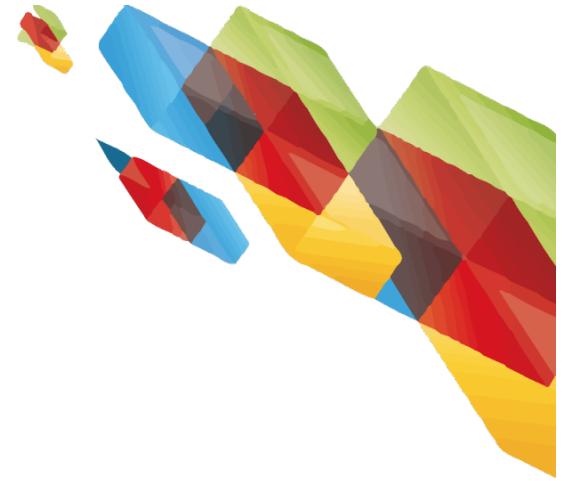
Javascript básico



Javascript en MongoDB



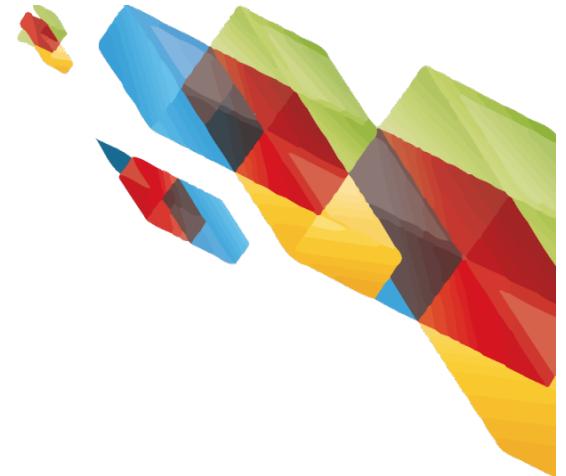
- El motor es el de Google V8 JavaScript a partir de la versión 2.4
- Permite múltiple operaciones javascript (Antes no)
- Se puede deshabilitar



JavaScript, Historia

- Inicialmente se llamaba LiveScript
- Lenguaje de guiones (script)
- Lo toma SUN en 1995 y le da su nombre JavaScript

Características generales

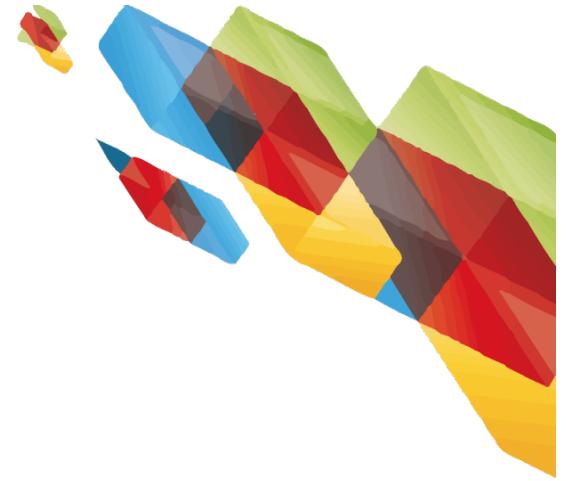


- Es un lenguaje interpretado
- Parte de un conjunto limitado de objetos y clases, pero es extensible
- Posee administradores de paquetes para ser extensible

Aplicaciones de JavaScript



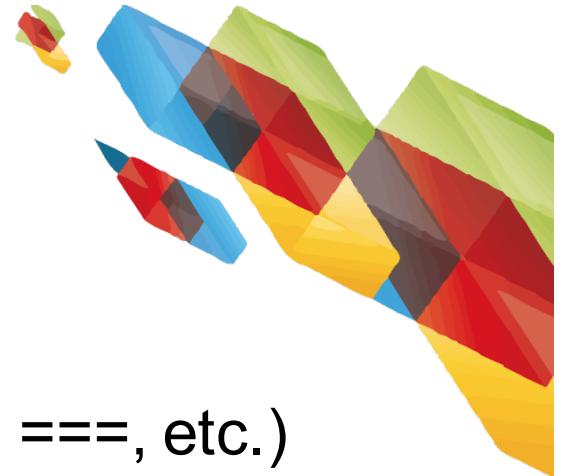
- Diseñado para programas sencillos y pequeños
- Realiza tareas repetitivas
- Diseñado para programar eventos de usuario
- Partió siendo utilizado para validación de datos en el browser
- Hoy funciona en el cliente y en el servidor
 - Node.js
 - MongoDB



Ventajas de JavaScript

- Desarrollo rápido
- (Relativamente) fácil de aprender
- Independencia de plataforma
- Prototype
- Estándar EcmaScript

Desventajas de JavaScript



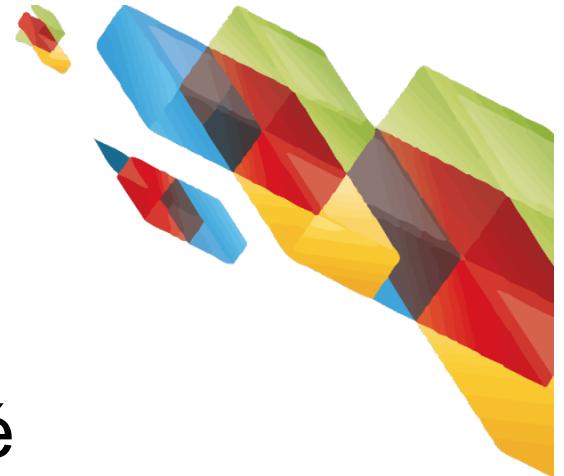
- Es fácil cometer errores (operadores ==, ===, etc.)
- No cuenta con encapsulación de código
- No obliga a declarar variables
- Callback hell

Objetos



- La utilización de "objetos" es un medio de organizar la información
- Se utilizan para describir entidades reales o imaginarias
- En su descripción se especifican:
 - Propiedades
 - Características que distinguen objetos del mismo tipo o clase
 - Métodos
 - Tareas que se pueden efectuar con las propiedades de un tipo de objeto

Clases de objetos



- Una clase de objeto especifica qué propiedades y métodos caracterizan a sus objetos, pero no asigna valores
 - automóvil
 - marca
 - modelo
 - persona
 - nombre
 - edad
 - estudiante
 - nombre
 - edad
 - carrera



mongoDB



Instancias de objetos



- Una instancia de objeto es un objeto de alguna clase, con valores en su propiedades
 - persona
 - Nombre: Luis
 - Edad: 36
 - alumno
 - Nombre: Juan
 - Edad: 19
 - Carrera: Diseño Gráfico

Jerarquías de objetos



- Los objetos pueden estar referidos a atributos de otros objetos
 - mediosdetransporte.bicicleta.manubrio



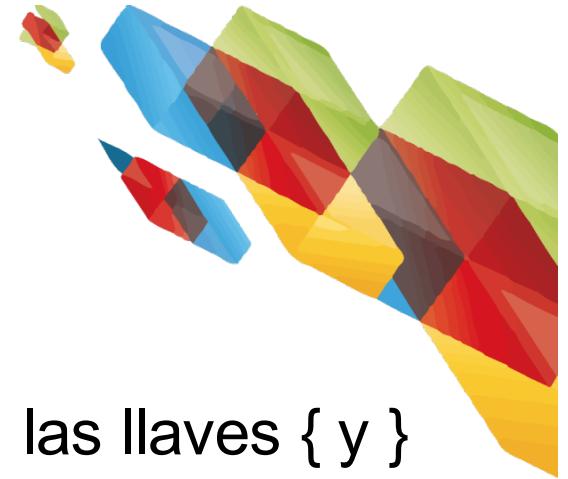
Creación de objetos propios

- Se pueden definir y crear objetos propios para describir información
 - Se definen, estableciendo su propiedades y sus métodos
 - Casa
 - # de cuartos
 - metros cuadrados
 - precio
 - define_precio()
 - Se crean, generando instancias y asignando valores

Bloques de comandos

- Se pueden agrupar comandos utilizando las llaves { y }

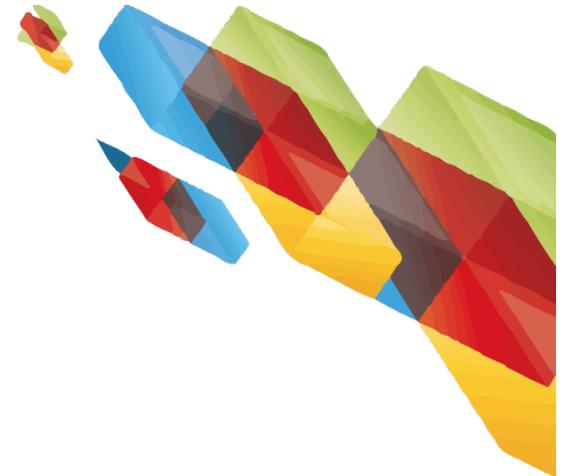
```
{  
  comando  
  comando  
}
```



Bloques de comandos

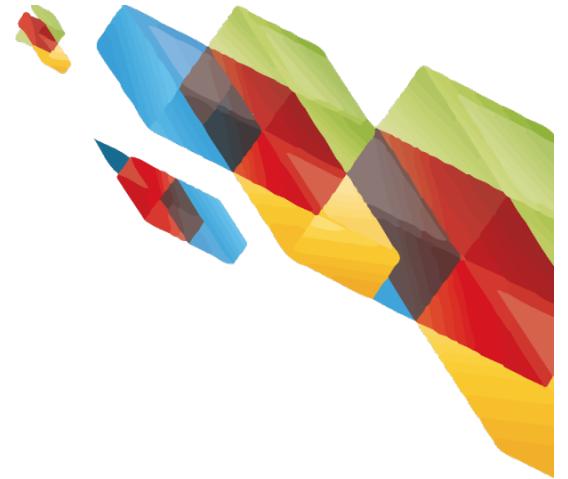
- Los bloques se pueden anidar

```
{  
    un comando  
    {  
        otro comando  
        otro comando  
    }  
}
```



Salida de texto en MongoDB

- print
- printjson





Tipos de datos

- Para representar la información, se utilizan cuatro tipos de dato

- Números

- 17, 21.3, 34e3 (decimales)
 - 056 (octal)
 - 0x5F (hexadecimal)

- Cadenas

- "Hola!"

- Booleanos

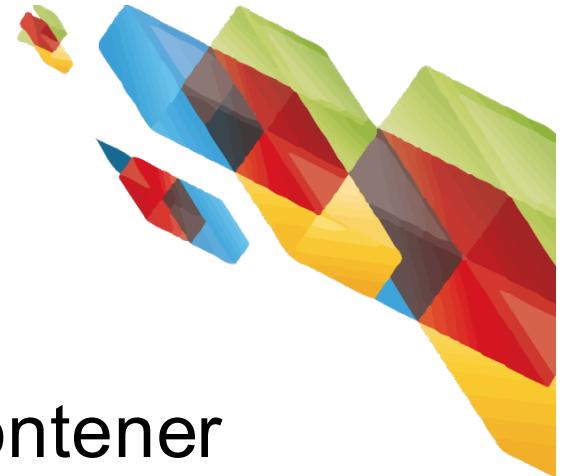
- true
 - false

- Nulos

 mongoDB • null



Variables



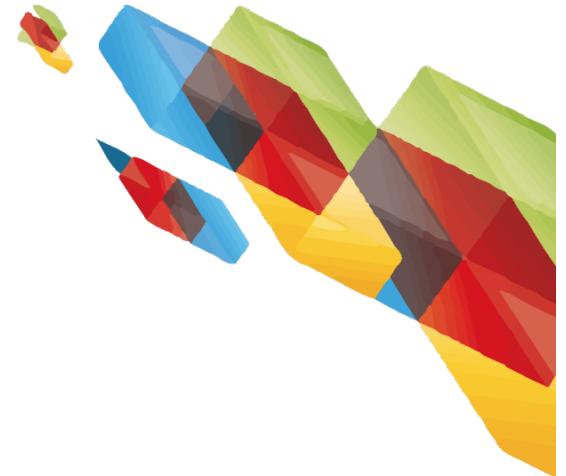
- Son identificadores que pueden contener valores, y éstos pueden variar su valor
- No es preciso declararlas, pero se recomienda
 - Declaración
 - var ejemplo;
 - Declaración y asignación
 - var ejemplo = "Hola";
 - Asignación
 - ejemplo = "Hola";
 - Invocación
 - documento.funcion(ejemplo);

Expresiones



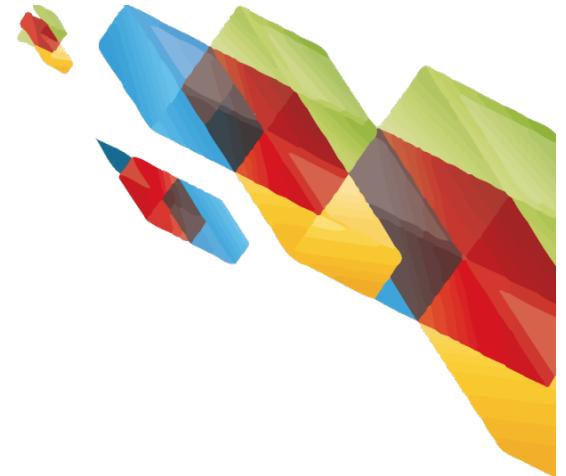
- Colección de variables, operadores y otras expresiones que se evalúan a un solo valor
 - Asignación
 - asigna un valor a una variable
 - Aritméticas
 - evalúan un número
 - Cadenas
 - evalúan una cadena
 - Lógicas
 - evalúan un valor booleano

Expresiones de asignación

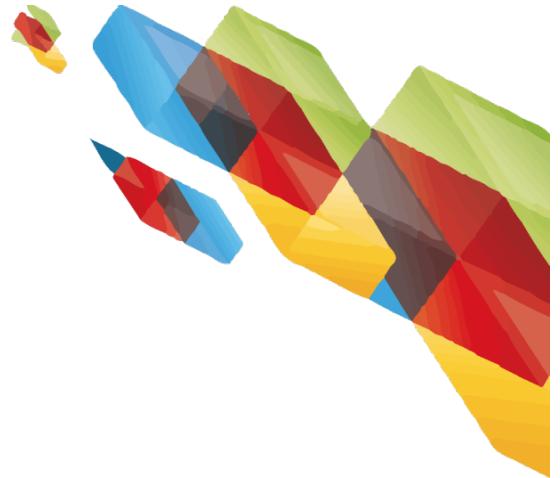


- **variable = expresión**
 - Evalúa la expresión, y el resultado se asigna a la variable
- **variable += expresión**
 - Evalúa la expresión, y el resultado mas el valor de la variable se asigna a la variable

Operadores aritméticos



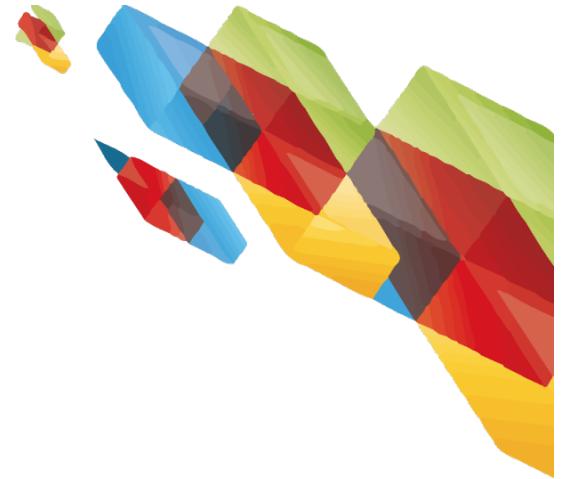
- + Suma
- - Resta
- * Multiplicación
- / División
- % Módulo (residuo de una división)
- Ejemplo
 - meses = edad * 12;
- Combinados
 - suma += dato;



Operadores lógicos

- **&&**
 - "y" lógico, devuelve true cuando ambos operandos son verdaderos y falso en otro caso
- **||**
 - "o" lógico, devuelve true cuando alguno de los operandos es verdadero
- **!**
 - "no" lógico, devuelve true si el operando es falso, y false si el operando es verdadero

Operadores de comparación



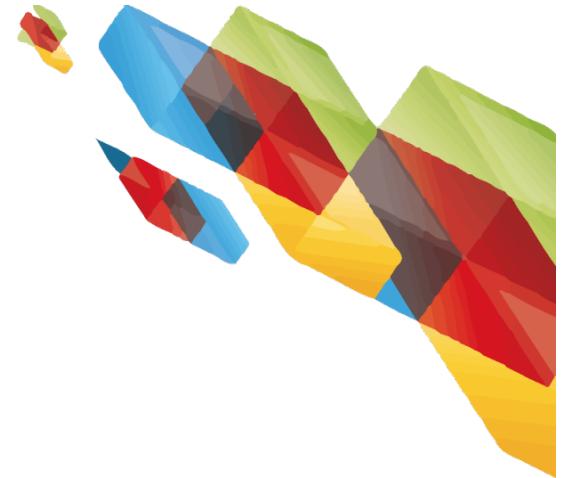
- ==
 - iguales, devuelve verdadero si los operandos son iguales
- !=
 - diferentes
- >
- <
- >=
- <=



Operador condicional

- Permite evaluar una expresión lógica, y devolver valores diferentes en consecuencia
 - (condición) ? valor1 : valor2
- Ejemplo
 - (nombre == "Luis") ? "Hola, Jefe" : "Hola, extraño"
 - Dependiendo del nombre, la expresión se evaluará como "Hola, Jefe" o "Hola, extraño"
 - respuesta = (edad<18) ? "Eres menor" : "Ya estás grandecito" ;

Operadores de cadena



- Permiten la unión de cadenas
 - "Hola, " + "qué tal!"
 - bienvenida += ", se bienvenido"



si (control de flujo)

- Se utilizan para que el programa tome decisiones de qué instrucciones ejecutar

```
if condicion  
    comando;
```

- Ejemplo

```
if( día == "domingo" )  
    document.write( "Hoy es domingo" );
```



si-sino

- Se considera el caso de que no se cumpla la condición

```
if condicion  
    comando;  
else  
    otro comando;
```

Utilización de bloques con el if



- Si se desean ejecutar varios comandos en lugar de sólo uno, se utilizan bloques

```
if condicion {  
    comando  
    comando  
} else {  
    comando  
    comando  
}
```

Funciones

- Permiten agrupar código para desempeñar una tarea o función específica y que puede usarse muchas veces en un programa



Funciones: definición



- La definición establece qué es lo que hace la función

```
function nombre_de_funcion( parámetros,  
    argumentos ) {  
    bloque de comandos  
}
```

Ejemplo

```
function despNombre( nombre ) {  
    document.write( "<hr>tu nombre es<b>" );  
    document.write( nombre )  
    document.write( "</b><hr>" );  
}
```

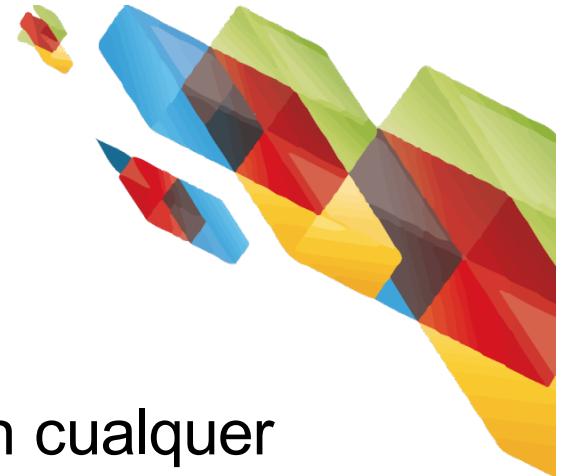


Retorno de resultados

- En los entornos de programación, a las funciones que no retornan resultados se les llama "procedimientos"
- Si retornan resultados son "funciones"

```
function cubo( numero ) {  
    return numero * numero * numero;  
}
```

Funciones: definición en el encabezado

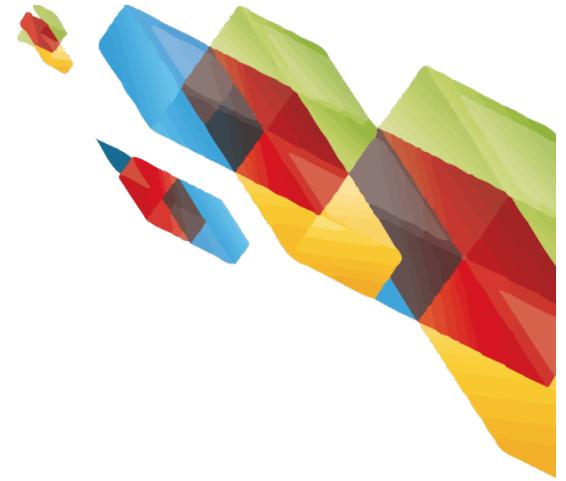


- En general el código de Java puede ir en cualquier parte
- Si va a generar texto para la página, el código debe estar donde debe generar el texto
- Se recomienda poner la definición de las funciones en el encabezado

Funciones: invocación (ponerlas a trabajar)



- Para hacer que las funciones hagan su trabajo, se les invoca desde el programa
 - despNombre("Luis");
- Si devuelven resultado, pueden formar parte de una expresión
 - a=cubo(4)+1;



Ciclos

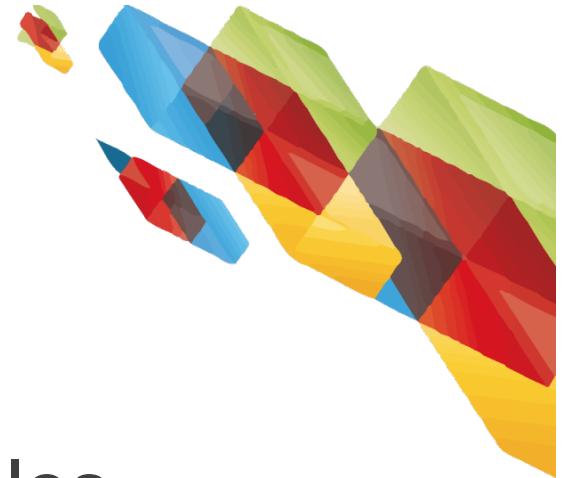
- Cuenta con diferentes ciclos
 - `for(i=0; i < 10;i++){...}`
 - `while`
 - `for(indice in arreglo)`
 - Este último se utiliza para arreglos que no están indexados de manera correlativa

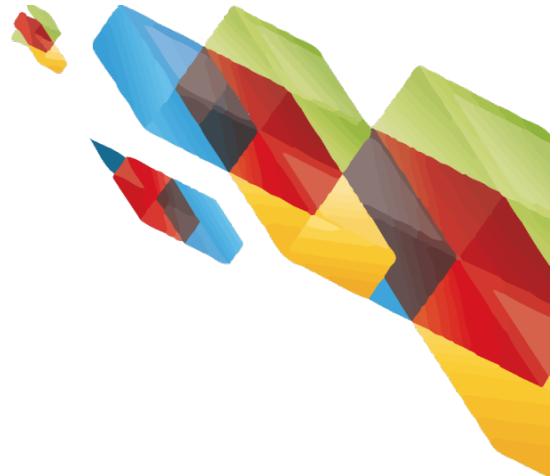
Agregación



Agregación

- Son operaciones que procesan los documentos y realizan un cálculo que devuelven como resultado.
- MongoDB provee varias operaciones de agregación
- El uso de estos operadores simplifica el código de las aplicaciones y permite gestionar de mejor manera el uso de recursos.





Tuberías / Pipelines

- ▶ Está basado en el concepto de pipelines
- ▶ Provee:
 - ▶ *filtros* que funcionan como una consulta
 - ▶ *Transformaciones al documento* que modifican el documento que se obtiene como resultado.
- ▶ Provee herramientas para :
 - ▶ Agrupar y ordenar por campos
 - ▶ Agregar contenido en arreglos, esto incluye arreglos de documentos
- ▶ El uso de operadores puede ser para por ejemplo calcular un promedio o concatenar strings.

```
db.zips.aggregate(  
    { $match: { state: "TN" } },  
    { $group: { _id: "TN", pop: { $sum: "$pop" } } }  
);
```

```
{  
    city: "LOS ANGELES",  
    loc: [-118.247896, 33.973093],  
    pop: 51841,  
    state: "CA",  
    _id: 90001  
}  
  
{  
    city: "NEW YORK",  
    loc: [-73.996705, 40.74838],  
    pop: 18913,  
    state: "NY",  
    _id: 10001  
}  
  
{  
    city: "NASHVILLE",  
    loc: [-86.778441, 36.167028],  
    pop: 1579,  
    state: "TN",  
    _id: 37201  
}  
  
{  
    city: "MEMPHIS",  
    loc: [-90.047995, 35.144001],  
    pop: 4144,  
    state: "TN",  
    _id: 38103  
}
```

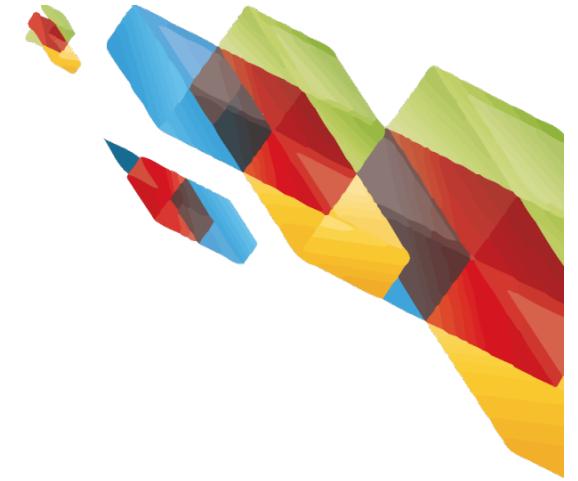
\$match

```
{  
    city: "NASHVILLE",  
    loc: [-86.778441, 36.167028],  
    pop: 1579,  
    state: "TN",  
    _id: 37201  
}  
  
{  
    city: "MEMPHIS",  
    loc: [-90.047995, 35.144001],  
    pop: 4144,  
    state: "TN",  
    _id: 38103  
}
```

\$group

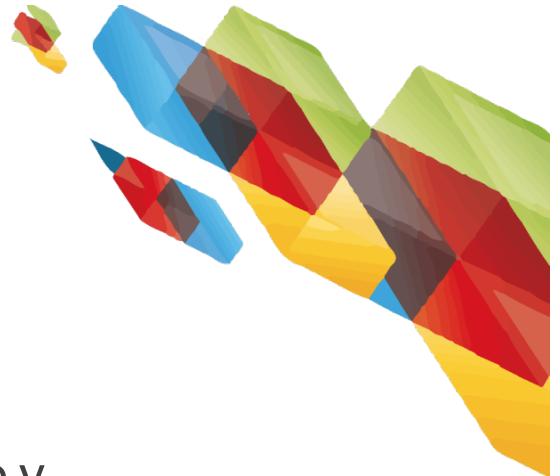
```
{  
    _id: "TN"  
    pop: 5723  
}
```





Tuberías / Pipelines

- \$limit
- \$skip
- \$sort



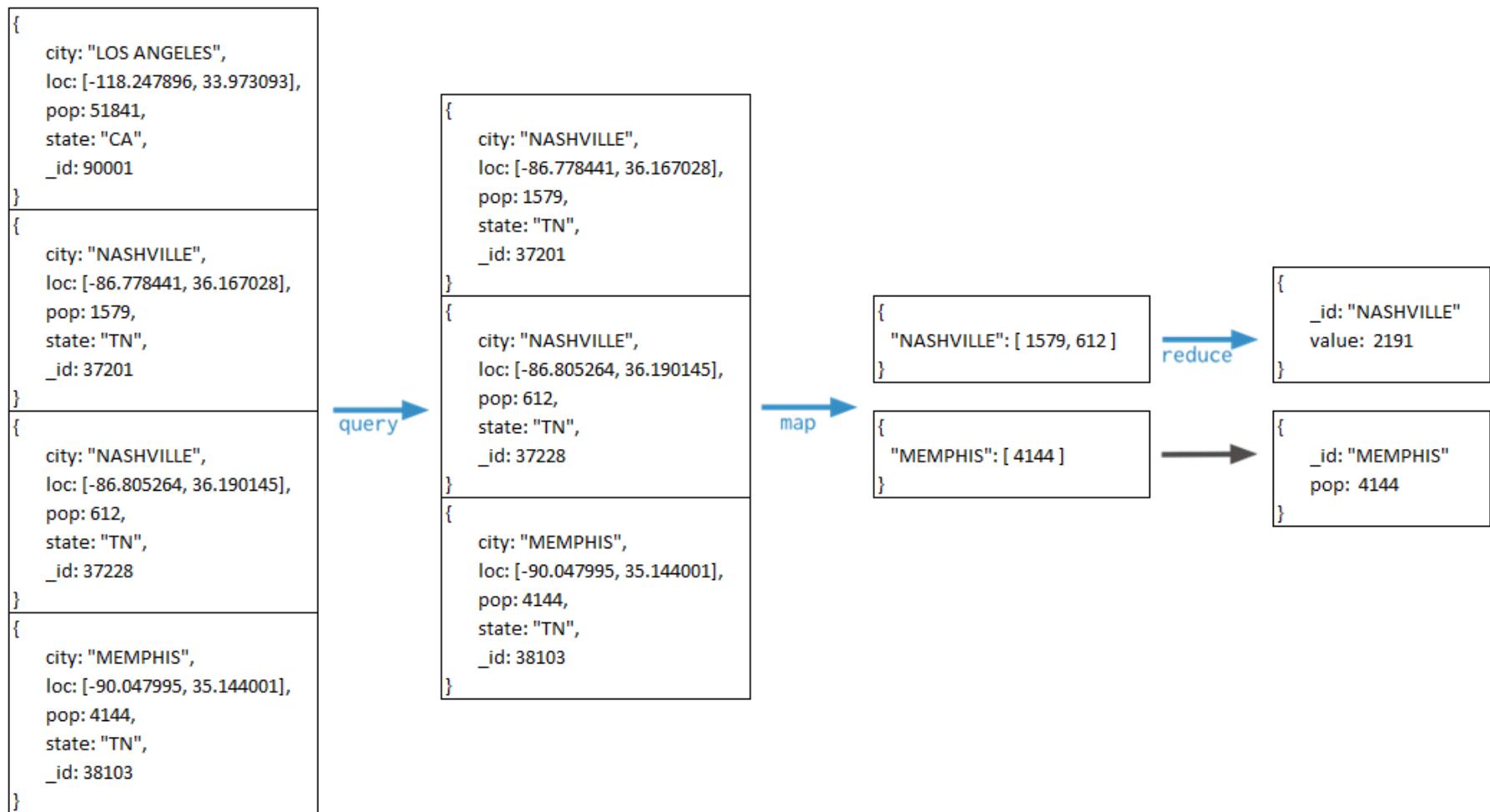
Map-Reduce

- Tiene dos fases:
 - Una fase *map* que procesa cada documento y emite uno o más objetos para esa entrada.
 - Una fase *reduce* que combina la salida de la fase *map*.
 - Una fase opcional *finalize* que permite realizar modificaciones al resultado final.
- Utiliza funciones JavaScript
 - Es mucho más flexible que la agregación, pero es menos eficiente y es más complejo.
 - Puede generar resultados que excedan los 16MBm que es el límite de las agregaciones.

```

db.zips.mapReduce(
    function() { emit( this.city, this.pop ); },
    function(key, values) { return Array.sum( values ) },
    {
        query: { state: "TN" },
        out: "city_pop_totals"
    }
);

```



Operaciones de agregación de propósito simple

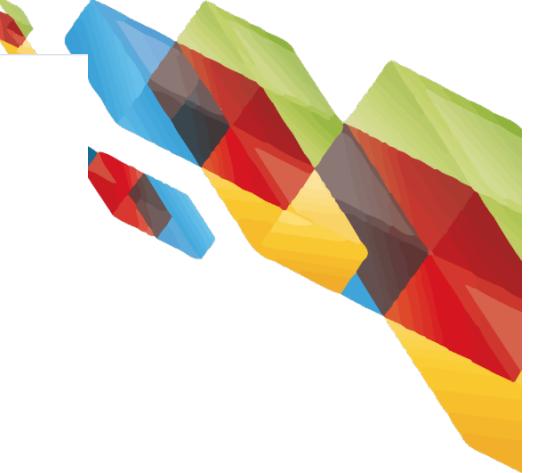


- Comandos de agregación de propósito simple:
 - ▶ Contar los documentos que calzan con una consulta
 - ▶ Retornar los distintos valores de un campo
 - ▶ Agrupar datos de acuerdo al valor de un campo.
- Operan a nivel de colección
- Carecen de la flexibilidad y las capacidades de las agregaciones y de map-reduce

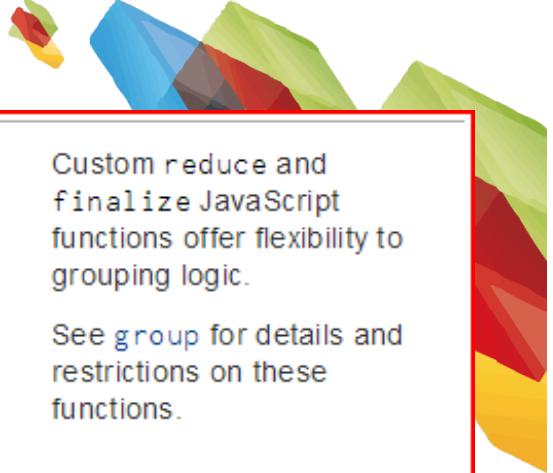
```
db.zips.distinct( "state" );
```

```
{  
  city: "LOS ANGELES",  
  loc: [-118.247896, 33.973093],  
  pop: 51841,  
  state: "CA",  
  _id: 90001  
}  
  
{  
  city: "NEW YORK",  
  loc: [-73.996705, 40.74838],  
  pop: 18913,  
  state: "NY",  
  _id: 10001  
}  
  
{  
  city: "NASHVILLE",  
  loc: [-86.778441, 36.167028],  
  pop: 1579,  
  state: "TN",  
  _id: 37201  
}  
  
{  
  city: "MEMPHIS",  
  loc: [-90.047995, 35.144001],  
  pop: 4144,  
  state: "TN",  
  _id: 38103  
}
```

distinct → ["CA", "NY", "TN"]



	<code>aggregate</code>	<code>mapReduce</code>	<code>group</code>
Description	<p><i>New in version 2.2.</i></p> <p>Designed with specific goals of improving performance and usability for aggregation tasks.</p> <p>Uses a “pipeline” approach where objects are transformed as they pass through a series of pipeline operators such as <code>\$group</code>, <code>\$match</code>, and <code>\$sort</code>.</p> <p>See Aggregation Reference for more information on the pipeline operators.</p>	<p>Implements the Map-Reduce aggregation for processing large data sets.</p>	<p>Provides grouping functionality.</p> <p>Is slower than the <code>aggregate</code> command and has less functionality than the <code>mapReduce</code> command.</p>
Key Features	<p>Pipeline operators can be repeated as needed.</p> <p>Pipeline operators need not produce one output document for every input document.</p> <p>Can also generate new documents or filter out documents.</p>	<p>In addition to grouping operations, can perform complex aggregation tasks as well as perform incremental aggregation on continuously growing datasets.</p> <p>See Map-Reduce Examples and Perform Incremental Map-Reduce.</p>	<p>Can either group by existing fields or with a custom key if JavaScript function, can group by calculated fields.</p> <p>See <code>group</code> for information and example using the <code>keyf</code> function.</p>



Flexibility	<p>Limited to the operators and expressions supported by the aggregation pipeline.</p> <p>However, can add computed fields, create new virtual sub-objects, and extract sub-fields into the top-level of results by using the <code>\$project</code> pipeline operator.</p> <p>See <code>\$project</code> for more information as well as Aggregation Reference for more information on all the available pipeline operators.</p>	<p>Custom map, reduce and finalize JavaScript functions offer flexibility to aggregation logic.</p> <p>See <code>mapReduce</code> for details and restrictions on the functions.</p>	<p>Custom reduce and finalize JavaScript functions offer flexibility to grouping logic.</p> <p>See <code>group</code> for details and restrictions on these functions.</p>
Output Results	<p>Returns results inline.</p> <p>The result is subject to the BSON Document size limit.</p>	<p>Returns results in various options (inline, new collection, merge, replace, reduce). See <code>mapReduce</code> for details on the output options.</p> <p><i>Changed in version 2.2:</i> Provides much better support for sharded map-reduce output than previous versions.</p>	<p>Returns results inline as an array of grouped items.</p> <p>The result set must fit within the maximum BSON document size limit.</p> <p><i>Changed in version 2.2:</i> The returned array can contain at most 20,000 elements; i.e. at most 20,000 unique groupings. Previous versions had a limit of 10,000 elements.</p>
Sharding	Supports non-sharded and sharded input collections.	Supports non-sharded and sharded input collections.	Does not support sharded collection.

```
C:\mongodb\bin>mongo.exe
MongoDB shell version: 2.4.9
connecting to: test
> show dbs
blog    0.203125GB
local   0.078125GB
test    0.203125GB
> use blog
switched to db blog
> db.zips.aggregate( {$match: {state: "TN"}}, {$group: {_id: "TN", pop: {$sum: "$pop"}}})
{
  "result" : [ { "_id" : "TN", "pop" : 4876457 } ],
  "ok" : 1
}
> db.zips.mapReduce(
...   function() { emit( this.city, this.pop ); },
...   function(key, values) { return Array.sum( values ); },
...   {
...     query: { state: "TN" },
...     out: "city_pop_totals"
...   }
... );
{
  "result" : "city_pop_totals",
  "timeMillis" : 198,
  "counts" : {
    "input" : 582,
    "emit" : 582,
    "reduce" : 13,
    "output" : 505
  },
  "ok" : 1
}
>
> db.city_pop_totals.find({_id: "NASHVILLE"})
{
  "_id" : "NASHVILLE",
  "value" : 349822
}
```

```
> db.zips.distinct( "state" )
[
```

AL
AK
AZ
AR
CA
CO
GT
DE
DC
FL
GA
HI
ID
IL
IN
IA
KS
KY
LA
ME
MD
MA
MI
MN
MS
MO
MT
NE
NU
NH
NJ
NM
NY
NC
ND
OH
OK
OR
PA
RI
SC
SD
TN
TX
UT
VT
VA
WA
WU
WI
WY

Contenido Binario



GridFS



- GridFS es el módulo para almacenar y recuperar archivos que exceden el límite de 16MB de un documento BSON
- En vez de almacenar el contenido en un solo documento, GridFS divide el archivo en partes o chunks y almacena cada uno de esos chunks como documentos.
- Por omisión GridFS limita el tamaños de los chunk a 255k.
- GridFS utiliza dos colecciones. Una colección para los chunks de los archivos y otra para almacenar la metadata del archivo.

GridFS



- Cuando se hace una consulta a GridFS por un archivo, el driver o el cliente reensamblan los chunks a medida que se requieren.
- Es factible hacer consultas de rango o acceder a secciones arbitrarias de los archivos (Por ej. “saltar” a la mitad del video)..
- GridFS es útil no solo para almacenar archivos de más de 16MB, sino que también para almacenar todos aquellos archivos que no se desean cargar completamente en memoria.

¿Cuándo usar GridFS?



- Si existen documentos en una colección mayores a 16MB se debe usar GridFS
- Si el filesystem limita el número de archivos por directorio, es factible usar GridFS para almacenar esos archivos.
- Cuando se requiere mantener archivo y su metada sincronizado a través de múltiples sistemas o ubicaciones. Utilizando réplicas distribuidas, MongoDB puede distribuir los archivos y su metada automáticamente a esas instancias.
- Cuando se desea acceder a porciones de la información contenida en archivos grandes, sin tener que cargarlos completamente en memoria.



¿Cuándo no usar GridFS?

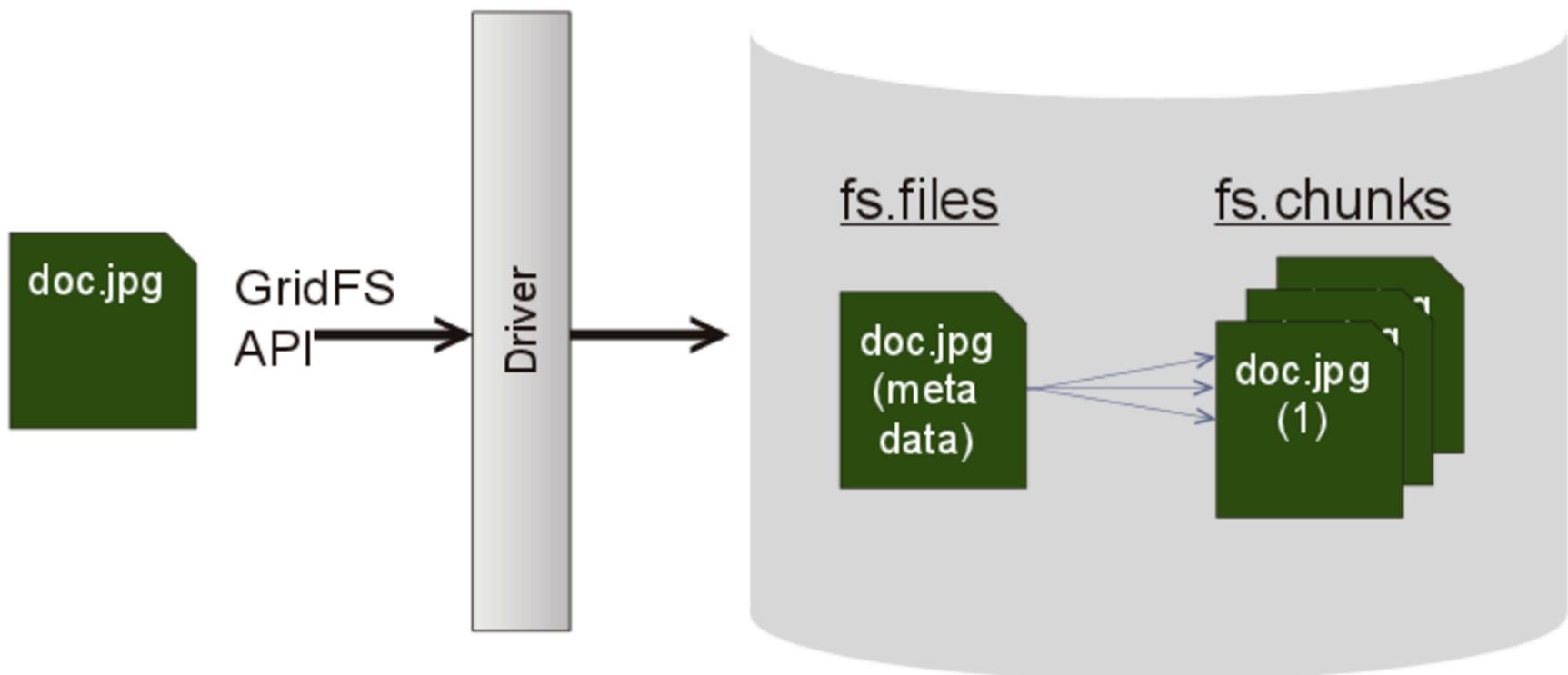
- Si se requiere actualizar el contenido de manera atómica.
- Si los archivos son de un tamaño menor a 16 MB, es una mejor alternativa guardar todo el archivo en un solo documento.
- Para guardar contenido binario se utiliza el tipo BinData.



GridFS

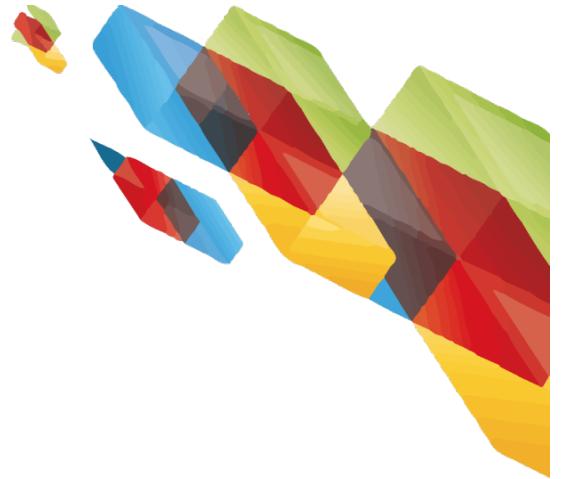
- GridFS almacena los archivos en dos colecciones:
 - chunks que almacena los pedazos binarios.
 - files, que almacena la metada de los archivos.
- GridFS almacena estas colecciones en un canasto (bucket)
- El bucket actúa como prefijo de las colecciones
- El nombre por omisión del bucket es “fs” :
 - fs.files
 - fs.chunks
- Para crear un bucket
 - new GridFS(db, "buck")
 - new GridFS(db, "fs")
 - new GridFS(db, "bucket"),

GridFS



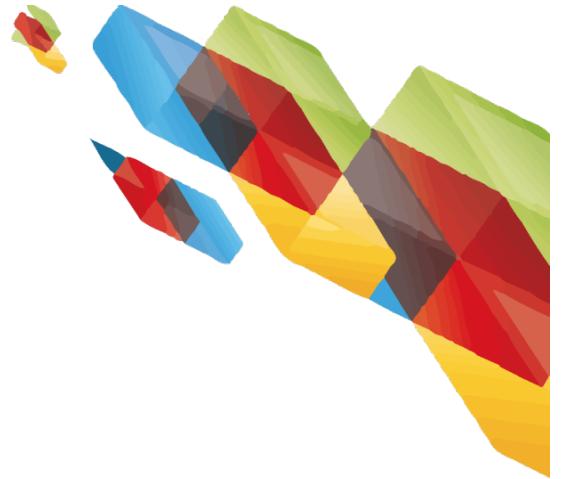
 mongoDB

 EXE



La colección Files

```
{  
  "_id" : <ObjectId>,  
  "length" : <num>,  
  "chunkSize" : <num>,  
  "uploadDate" : <timestamp>,  
  "md5" : <hash>,  
  "filename" : <string>,  
  "contentType" : <string>,  
  "aliases" : <string array>,  
  "metadata" : <dataObject>  
}
```



La colección chunks

```
{  
  "_id" : <ObjectId>,  
  "files_id" : <ObjectId>,  
  "n" : <num>,  
  "data" : <binary>  
}
```

Índices en GridFS



- GridFS utiliza un único índice compuesto para la colección chunks. Los campos indexados son el identificador del archivo, files_id y el campo n que es la secuencia del chunk.
- El primer chunk es el chunk 0.
- Ejemplo para recuperar los chunks:

```
cursor = db.fs.chunks.find({files_id: myFileID}).sort({n:1});
```



mongofiles

- Para incorporar archivos a una base de datos se usa el comando mongofiles
- mongofiles -d db list
- mongofiles -d db put archivo.pdf
- mongofiles -d db get archivo.pdf
- mongofiles -d db delete 32-corinth.lp
- mongofiles -d db search archivo
- mongofiles --help

Escalamiento



Replicación

- ¿Qué es replicación?
- Propósito de replicar o tener redundancia
 - Tolerancia a fallas
 - Disponibilidad
 - Incrementa la capacidad de lectura

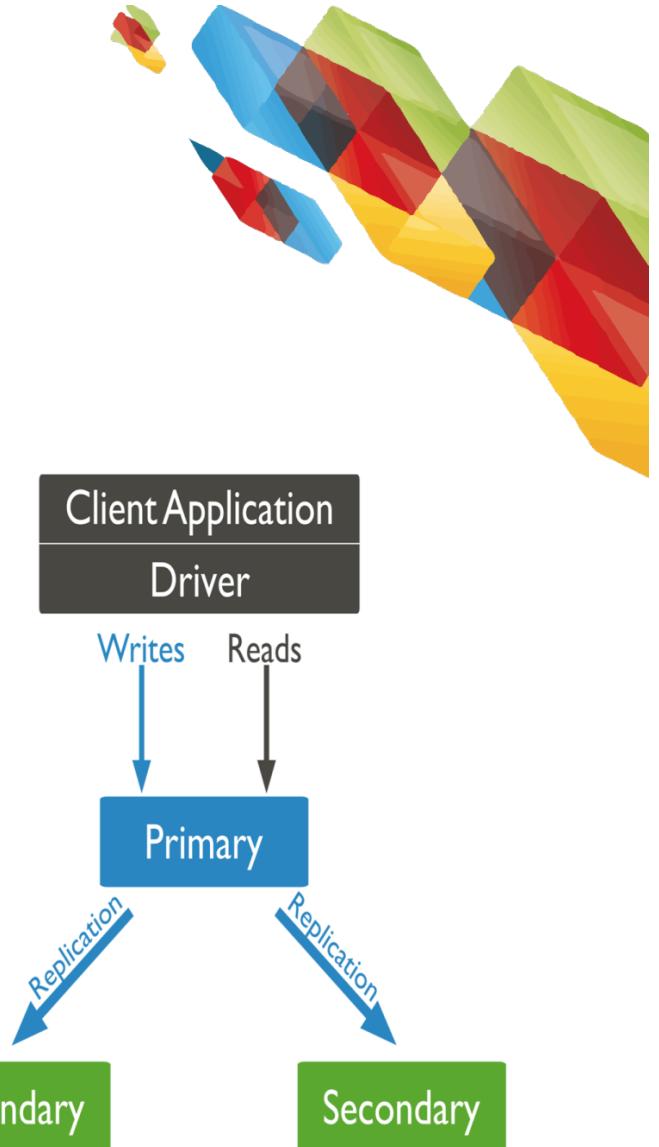
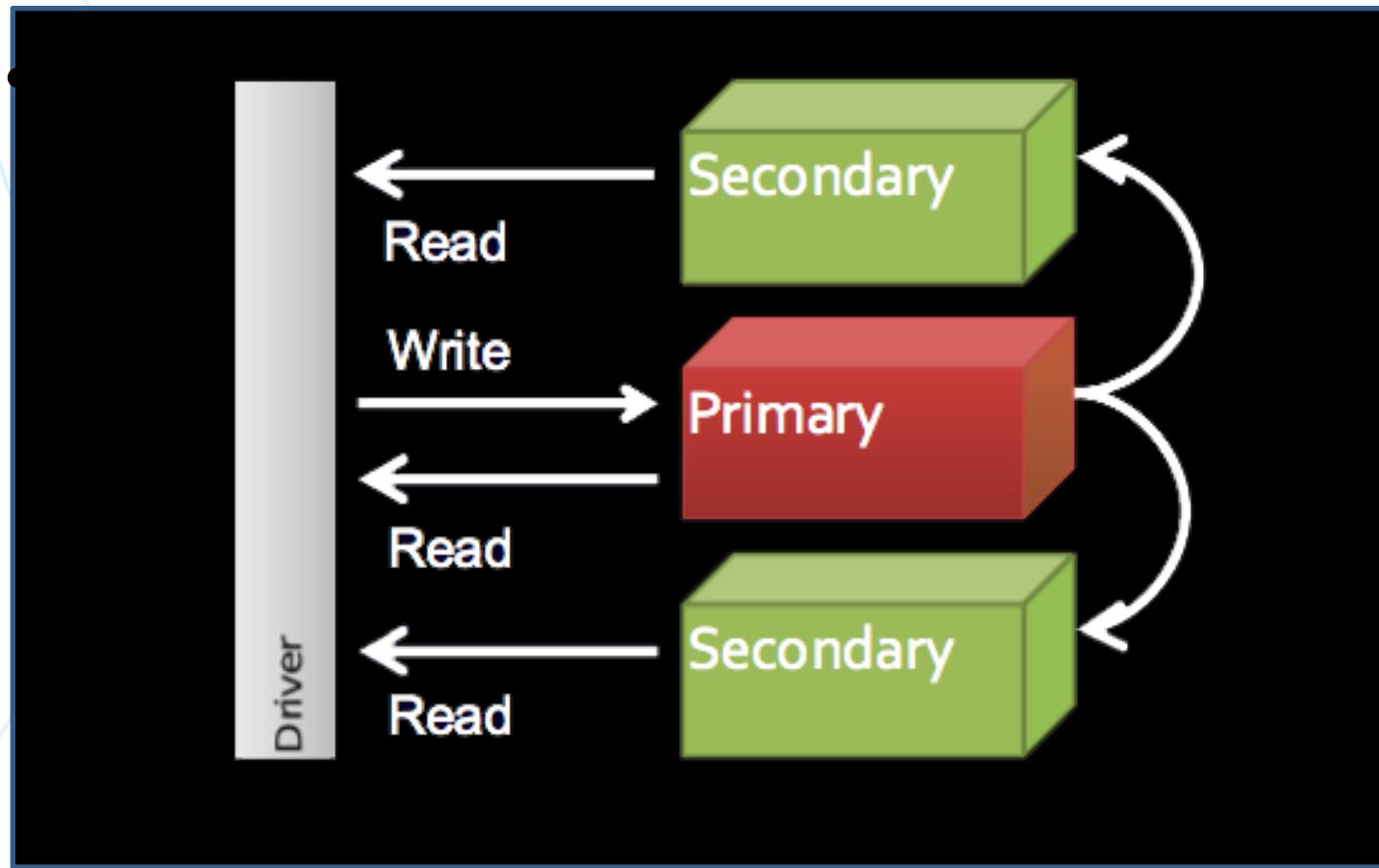
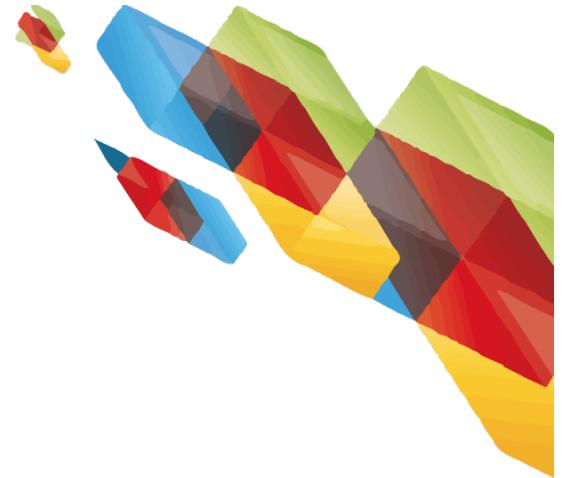


Figure 1: Diagram of default routing of reads and writes to the primary.

Replicación



Replicación



- Existen distintos tipos de miembros de un grupo replicado
 - Primario
 - Operaciones de lectura y escritura
 - Secundarios
 - Replicación asíncrona
 - Puede ser primario
 - Arbitro
 - Votación
 - No puede ser primario
 - Secundario en espera
 - No puede ser primario

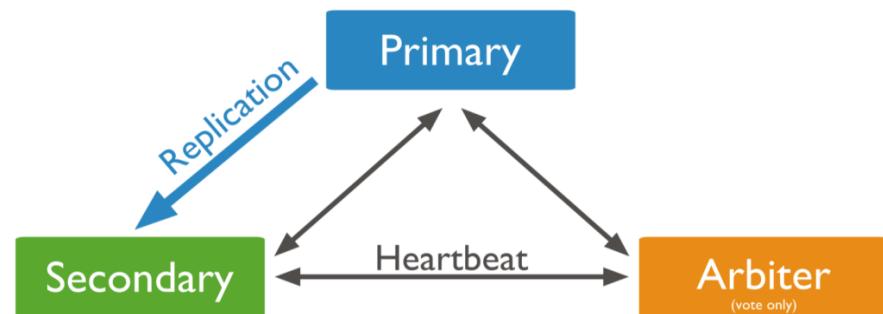
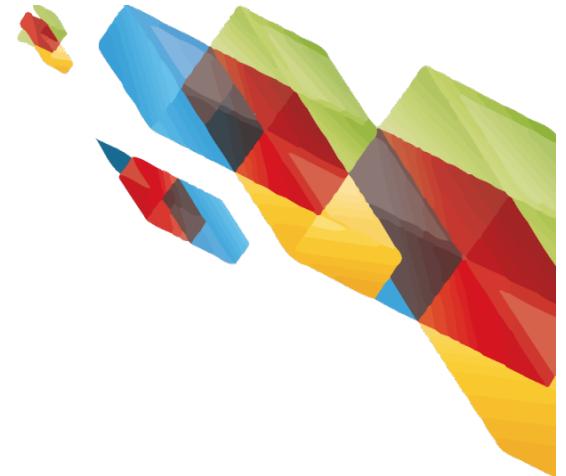
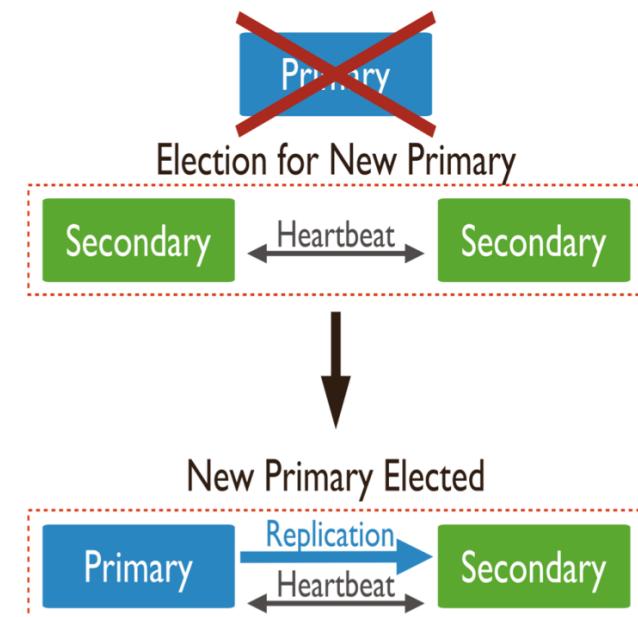


Figure 3: Diagram of a replica set that consists of a primary, a secondary, and an arbiter.

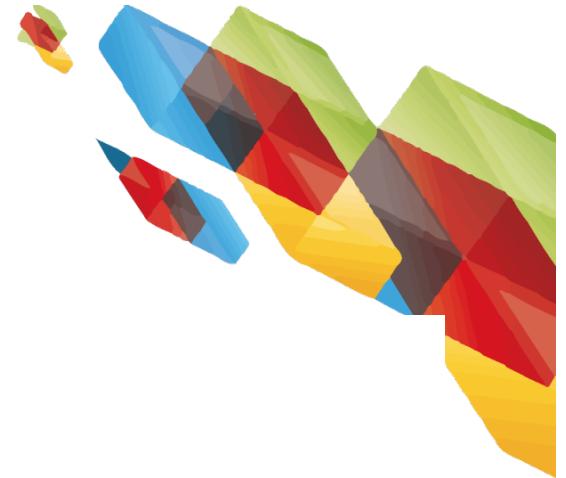
Replicación en MongoDB



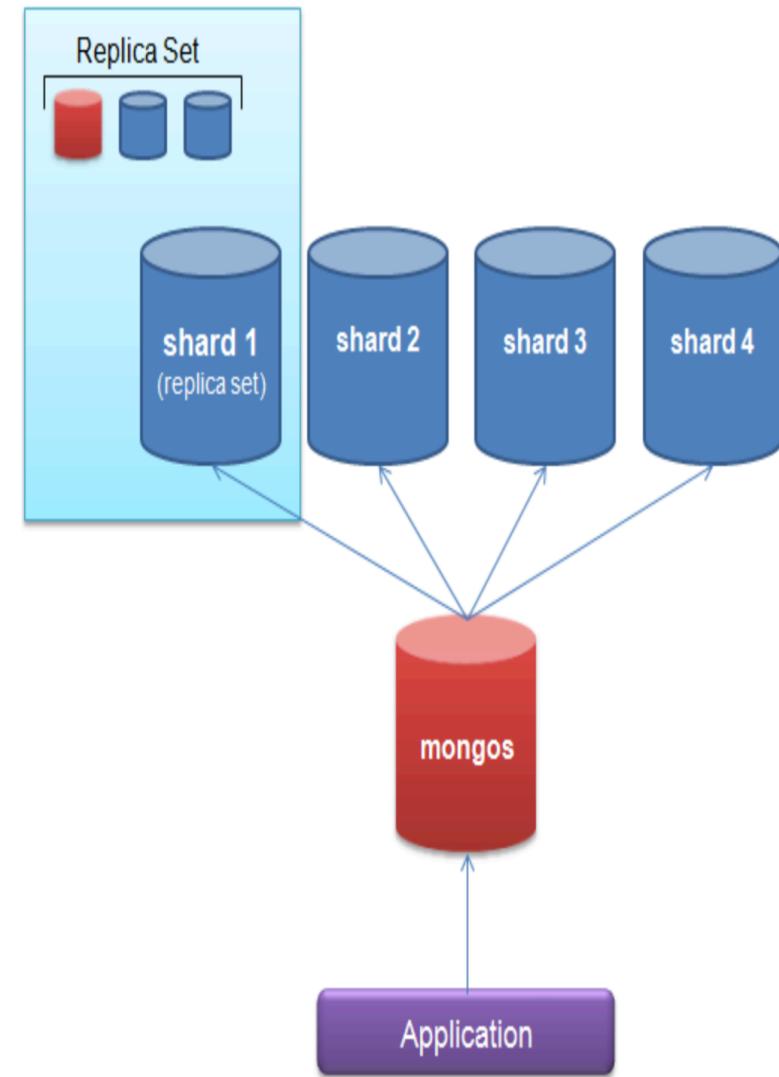
- Automatic Failover
 - Heartbeats
 - Elecciones
- The Standard Replica Set Deployment
- El número de miembros debe ser impar
- Rollback
- Security
 - SSL/TLS



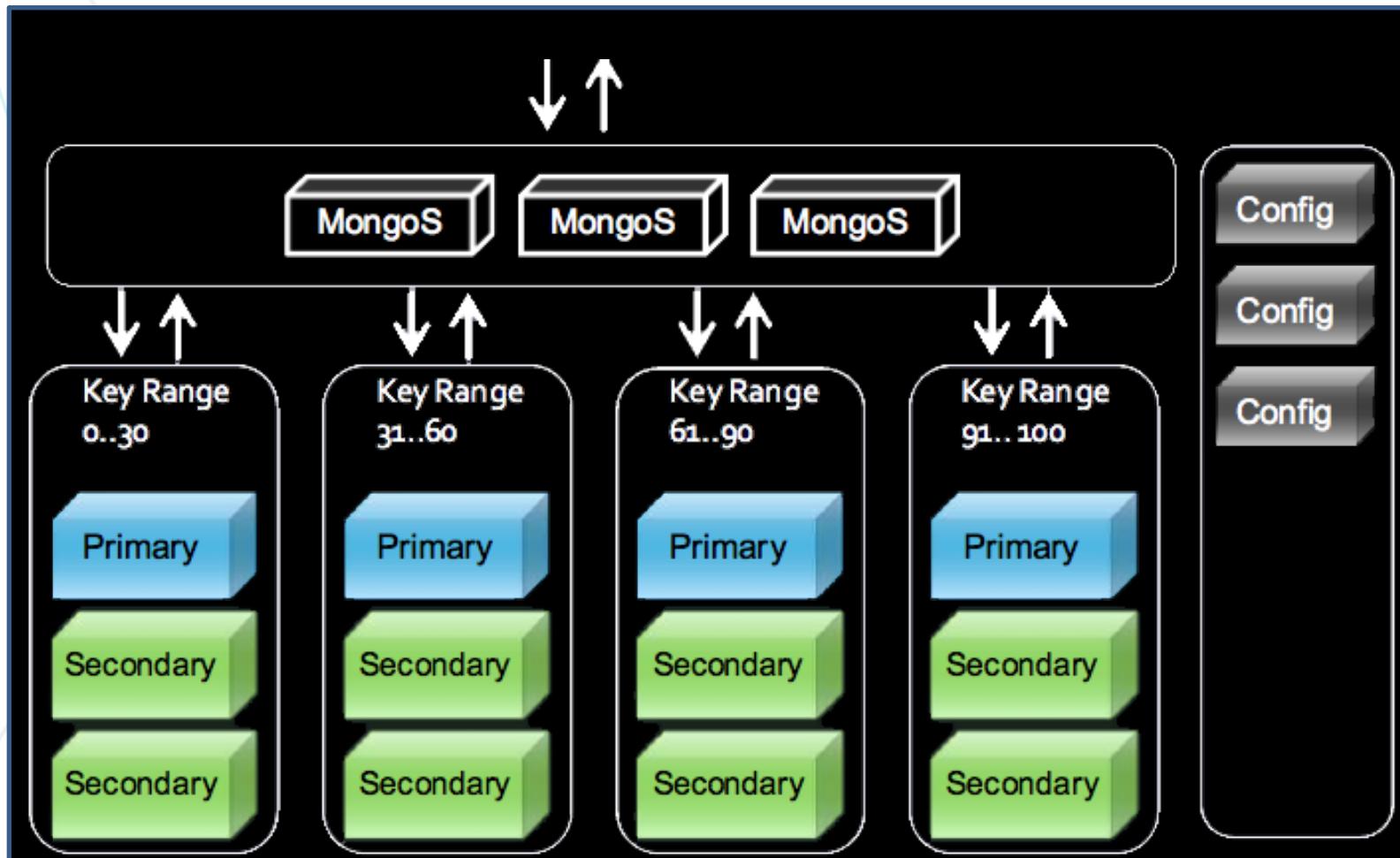
Sharding

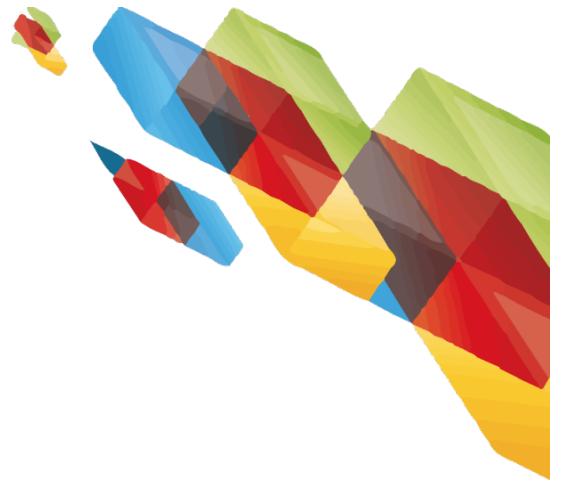


- ¿Qué es sharding?
- El propósito:
 - Escalar horizontalmente
- Query Routers
 - mongos
- Shard keys
 - Sharding basado en rangos
 - Cardinalidad



Shardig





Thanks

