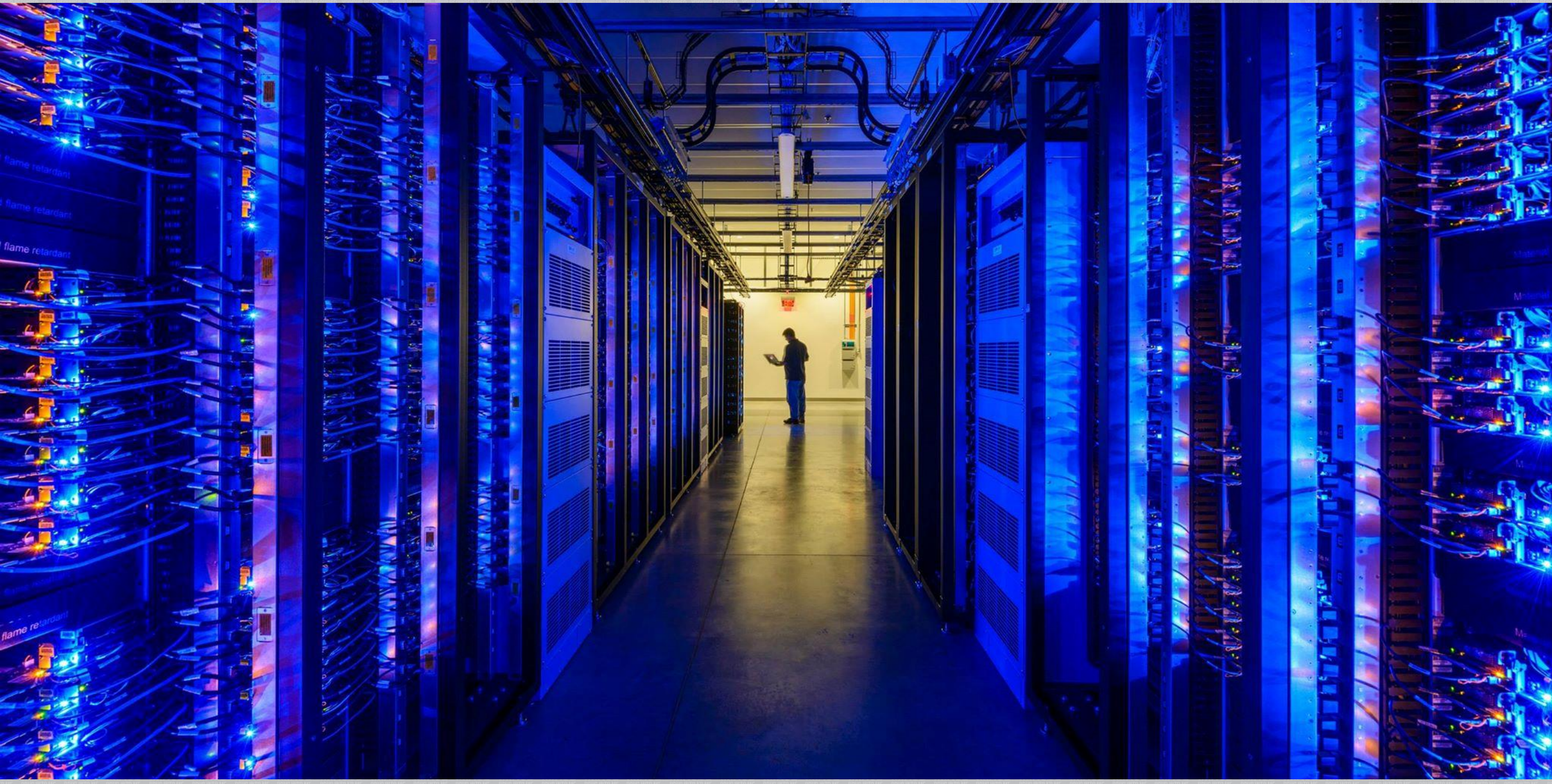


Tratamiento de datos masivos

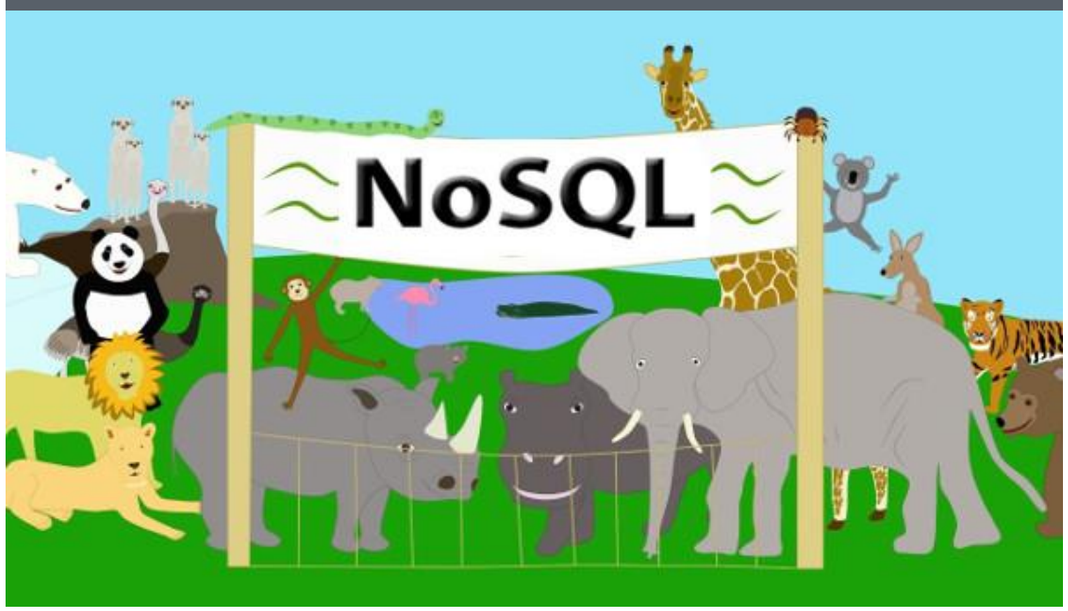
Rafael Caballero Roldán – rafacr@ucm.es

+ **Big Data**

MongoDB

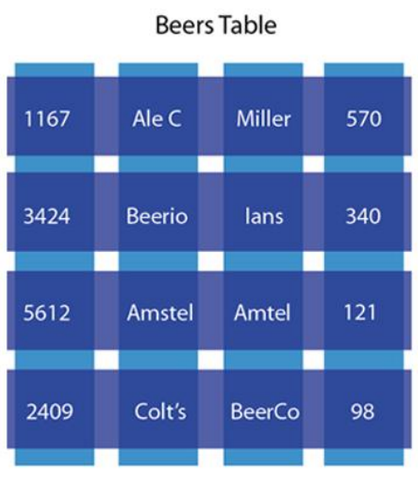
Una base de datos orientada a documentos que almacena datos en formato JSON

- Introducción
- Instalación y arranque
- Json
- Consultas simples
- Agregaciones



Bases de datos orientadas a documento

Almacenan documento complejos, muchos más que en bases de datos relacionales, documentos que además se pueden consultar y manipular desde el propio lenguaje. Su punto débil son las consultas complicadas incluyendo Map/Reduce



Orientada a Web, muy escalable pensada para **funcionar** "en el móvil y en un clúster"

CouchDB

Todas las ideas de NoSQL en una sola base de datos. Robusta y a la vez fácil. No muy eficiente.

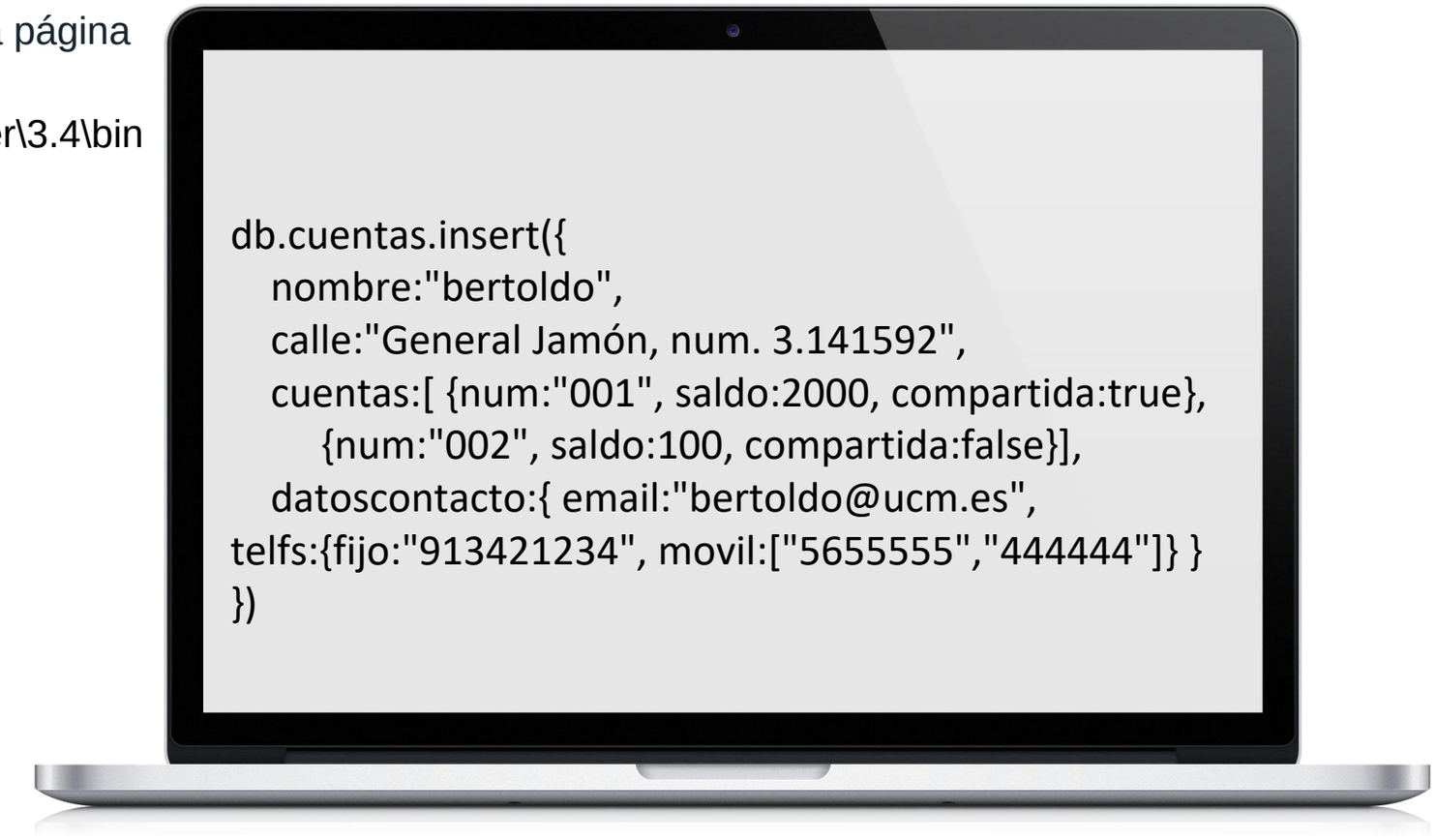
MongoDB

mongoDB

{ name: mongo, type: DB }

1 Instalación

Windows: Instalar el .msi desde la página
Añadir al path del sistema:
C:\Program Files\MongoDB\Server\3.4\bin



2 Arrancar el servidor

En un terminal
arrancar el servidor
mongod –dbpath datos

3 Arrancar el cliente

mongo

- MongoDB almacena los datos en *colecciones*.
- Una colección almacena diferentes documentos JSON, cada uno con una clave única dentro de la colección.

- Los objetos (documentos en lenguaje Mongo) se encierran entre { }
- Son secuencia cualquiera de elementos separados por comas.
- Los elementos se escriben de la forma **clave:valor**
- En cada pareja clave:valor, el valor puede ser:

  Un número

  Una cadena (entre comillas dobles, en el caso de Mongo y Python también se admiten comillas simples)

  Un booleano: **true** o **false**.

  Un array de valores, encerrados por [ ]

  Un objeto (entre { })

  null (no admitido en Mongo)

- MongoDB almacena los datos en *colecciones*.
- Una colección almacena diferentes documentos JSON, cada uno con una clave única dentro de la colección.

find,skip,limit

Para mostrar por pantalla todos los documentos de una colección:

db.coord.find()

Inicialmente solo muestra los 10 primer documentos. Para ver más teclear it.

Se puede mejorar diciendo al shell que lo imprima bien indentado:

db.cuentas.find().pretty()

En el shell es muy normal encadenar funciones con "." . En cualquier momento podemos añadir help() al final para ver las opciones disponibles. Por ejemplo: db.cuentas.find().pretty().help()

También se pueden "saltar" saltar unos cuantos resultados. Queremos los documentos que ocupan las posiciones 2 y 3, ambas incluidas:

db.cuentas.find().limit(2).skip(1).pretty()

Si solo queremos el primer elemento, podemos usar findOne:

db.cuentas.findOne()

sort

Para ordenar usaremos la estructura

db.coleccion.find(.....).sort(...)

Dentro de sort pondremos un criterio de ordenación que en general serán de la forma {clave:1} (ascendente) o {clave:-1} (descendente)

Proyección

La forma general de find es:
find(consulta, proyección)

La proyección indica qué información se mostrará de los documentos obtenidos por la consulta.

Las reglas básicas de la proyección son las siguientes:

{ } muestra todos los pares clave:valor
{clave1:true,...,claven:true}: muestra solo las claves 1...n (y además la clave _id)
{clave1:false,...,claven:false}: muestra todas las claves salvo la 1..n (y además la clave _id)

No se pueden mezclar valores true y false, con una excepción; en el caso de documento con valores true se puede añadir "_id":false para indicar que no queremos que se muestre el identificador.

mongoDB

{ name: mongo, type: DB }

Consultas Básicas

Las consultas toman la forma de un documento JSON.

- Se pueden comparar a la sección where de una consulta SQL: irven para seleccionar un subconjunto de documentos de una colección (aquellos que cumplen la condición expresada por la consulta).

- El documento vacío { } visto como consulta equivale a aceptar todos los miembros de la colección. Por ejemplo, db.restaurants.findOne({ }) es equivalente a db.restaurants.findOne() .

- {a:b} se entiende como "selecciona los documentos en los que la clave 'a' toma el valor 'b'". En SQL sería "where a=b".

- En caso de que "a" sea un array busca que contenga el valor "b".

Operadores

En lugar de la igualdad se pueden usar operadores como precio:{$gte:50} , que buscan documentos incluyendo claves de la forma precio:v tales que v>=50.
La lista completa de operadores incluye de comparación, lógicos, de evaluación y muchos otros.

Ejemplos:

1. En la colección de coordenadas creada más arriba (donde cada documento incluye valores para x e y) encontrar valores de x<5 tales que y>6

```
db.coord.find({x:{$gt:5}, y:{$lt:6}},{"_id":false})
```

2. En la colección de coordenadas, encontrar valores tales que 3<x<5:

```
db.coord.find({x:{$gt:3, $lt:5}},{"_id":false})
{ "x" : 4, "y" : 2 }
{ "x" : 4, "y" : 2 }
```

3. En la base de datos de bancos, buscar los clientes que empiezan por "B".

```
db.cuentas.find({nombre:{$gte:"B", $lt:"C"}},{"nombre":true,"_id":false})
```

Operadores. Preguntas

Pregunta. Cual de las siguientes consultas devolver los documentos con 'score' entre 50 y 60, ambos inclusive?

a) db.scores.find({ score : { $gt : 50 , $lt : 60 } });
b) db.scores.find({ score : { $gte : 50 , $lte : 60 } });
c) db.scores.find({ score : { $gt : 50 , $lte : 60 } });
d) db.scores.find({ score : { $gte : 50 , $lt : 60 } });
e) db.scores.find({ score : { $gt : 50 } });

Operadores. Preguntas

Pregunta. Cual de las siguientes consultas devolver los documentos con 'score' entre 50 y 60, ambos inclusive?

a) db.scores.find({ score : { $gt : 50 , $lt : 60 } });
b) db.scores.find({ score : { $gte : 50 , $lte : 60 } });
c) db.scores.find({ score : { $gt : 50 , $lte : 60 } });
d) db.scores.find({ score : { $gte : 50 , $lt : 60 } });
e) db.scores.find({ score : { $gt : 50 } });

Respuesta: b)

Operadores. Preguntas

Pregunta. Qué hace esta consulta:
db.scores.find({ score : { $gt : 50 }, score : { $lt : 60 } }); ?

a) Encuentra todos los documentos con 'score’ entre 50 and 60

b) Provoca el pánico el servidor, que huye despavorido

c) Encuentra todos los documentos con 'score' superior a 50

d) Encuentra todos los documentos con 'score' menor que 60

Operadores. Preguntas

Pregunta. Qué hace esta consulta:
db.scores.find({ score : { $gt : 50 }, score : { $lt : 60 } }); ?

a) Encuentra todos los documentos con 'score' entre 50 and 60

b) Provoca el pánico el servidor, que huye despavorido

c) Encuentra todos los documentos con 'score' superior a 50

d) Encuentra todos los documentos con 'score' menor que 60

Respuesta: d)

$or, $and, $exists

$or, $and

Permiten hacer la disyunción/conjunción de varias consultas. Se usan de forma prefija seguidos por un array de documentos (consultas).

Buscar coordenadas que tengan o bien 4<x<6 o 10<x<12

```
db.coord.find({$or:[{x:{$gt:4, $lt:6}}, {x:{$gt:10, $lt:12}}]},{"_id":false})
{ "x" : 5, "y" : 2.5 }
{ "x" : 11, "y" : 5.5 }
{ "x" : 5, "y" : 2.5 }
{ "x" : 11, "y" : 5.5 }
```

$exists

Útil para comprobar si una clave existe en un documento.

listado de los clientes del banco que no tienen "cuentas":

```
db.cuentas.find({cuentas:{$exists:false}},{"nombre":true,"_id":false})
{ "nombre" : "Herminia" }
```

Arrays

Para consultar si un elemento está en un array, en el shell basta con indicar el elemento como valor asociado a la clave (el nombre del array).

Ejemplo: Nombre de personas a las que les gusta correr:

```
db.gustos.find({aficiones:"correr"},{nombre:true,"_id":false})
{ "nombre" : "Herminia" }
{ "nombre" : "Godofredo" }
```

Esto nos hace ver que la expresión 'clave:valor' está sobrecargada;

- Busca elementos del documento con la forma 'clave:valor'
- Busca elementos del documento con la forma 'clave:[....,valor,....]'

Pregunta: Nombre de las personas a las que les gusta la ensalada 

```
db.gustos.find({comida:"ensalada"},{nombre:true,"_id":false})
```

Arrays: operadores

$all

Supongamos que queremos que queremos saber si alguien le gusta las lentejas y el cocido. Sería un error escribir:

```
db.gustos.find({comida:["ensalada","cocido"]})
```

porque estaríamos pidiendo que solo le gusten las dos cosas (y ninguna más). Para pedir que un array contenga al menos los elementos que indicamos se usa el operador $all

```
db.gustos.find({comida:{$all:["lentejas","cocido"]}},{nombre:true,"_id":false})
{ "nombre" : "Bertoldo" }
```

$in

¿A qué personas les gusta el champán o la ensalada (o las dos cosas)?

```
db.gustos.find({comida:{$in:["champán","ensalada"]}},{nombre:true,"_id":false})
{ "nombre" : "Herminia" }
{ "nombre" : "Aniceta" }
{ "nombre" : "Godofredo" }
```

Arrays: posiciones

También se puede preguntar por el valor de una posición determinada del array.

Por ejemplo, suponiendo que el orden en la lista de aficiones es relevante (la primera es la preferida), queremos saber el nombre de las personas cuya afición preferida es correr:

```
db.gustos.find({"aficiones.0":"correr"},{nombre:1, _id:0})
```

```
{ "nombre" : "Herminia" }
{ "nombre" : "Godofredo" }
```

Pregunta difícil: Nombre de las personas a las que les gustan al menos 3 comidas

Arrays: subcomponentes

Para buscar valores dentro de un array o dentro de un subdocumento se usa la notación ".". Para ello volvemos a la tabla de cuentas.

Ejemplo: personas que tienen un saldo menor de 400 euros en alguna de sus cuentas.

```
db.cuentas.find({"cuentas.saldo":{$lt : 400}}, {nombre:true, _id:false})
{ "nombre" : "Bertoldo" }
{ "nombre" : "toribia" }
```

La notación "." también puede utilizarse para referirse a una posición de un array (con 0 la primera posición).

Ejemplo: nombres de personas que tienen un saldo mayor a 1000 euros en la primera cuenta

```
db.cuentas.find({"cuentas.0.saldo":{$gt :1000}}, {nombre:true, _id:false})
{ "nombre" : "Bertoldo" }
{ "nombre" : "Aniceto" }
```

Update

En general las modificaciones se hacen mediante:

db.collection.update(query, update, options)

query selecciona los elementos que se van a actualizar.
update indica los cambios que se van a hacer
options es una parte opcional. Aquí veremos las opciones upsert y multi.

Update total

Una primera forma fácil de hacer update es simplemente reemplazar un documento por otro; es decir un update total.
En este tipo de update el segundo parámetro es simplemente el valor que queremos poner en el documento (excepto el _id que será el dado por el primer parámetro query).

Por ejemplo, en la colección estelar queremos cambiar el tipo de Plutón por el de "planeta enano". Supongamos que probamos a hacer:

```
db.estelar.update({_id:3}, {tipo:"planeta enano"})
```

¿Qué ocurre?

Y ahora

```
db.estelar.update({_id:3},{nombre:"Plutón", tipo:"planeta enano"  })
```

Update Parcial

El segundo parámetro indica con algún operador como $set los cambios que se quieren hacer sobre el documento recuperado por la query.
Por ejemplo, supongamos que queremos cambiar el nombre "aniceto" a "Aniceto"

```
db.cuentas.update( {nombre:"aniceto"}, {$set : { nombre: "Aniceto"}})
```

Si la clave en $set no existe se añade, lo que hace de update una herramienta útil para insertar nueva información.

```
db.cuentas.update( {nombre:"Aniceto"}, {$set : {edad: 21}})
```

Si no hay nadie con nombre "Aniceto" entonces la instrucción no hará nada. Sin embargo, en ocasiones queremos que en caso de que esté lo modifique, y sino está que lo añada. Esta mezcla de "insert" y "update" es conocida como "upsert" y se añader como una opción para update. Las opciones se ponen como tercer parámetro:

```
db.cuentas.update( {nombre:"Aniceto"}, {$set : {edad: 21}}, {upsert:true})
```

remove({query })

Para borrar elementos de una colección se puede usar el método remove que recibe como parámetro una query y elimina todos los elementos de la colección que encajan con la consulta.

db.telef.remove({}) : borra todos los elementos de la colección telef. Si se quiere hacer esto es mejor db.people.drop()

db.cuentas.remove({nombre:"Bertoldo"}): borra a Bertoldo de la colección

mongoDB

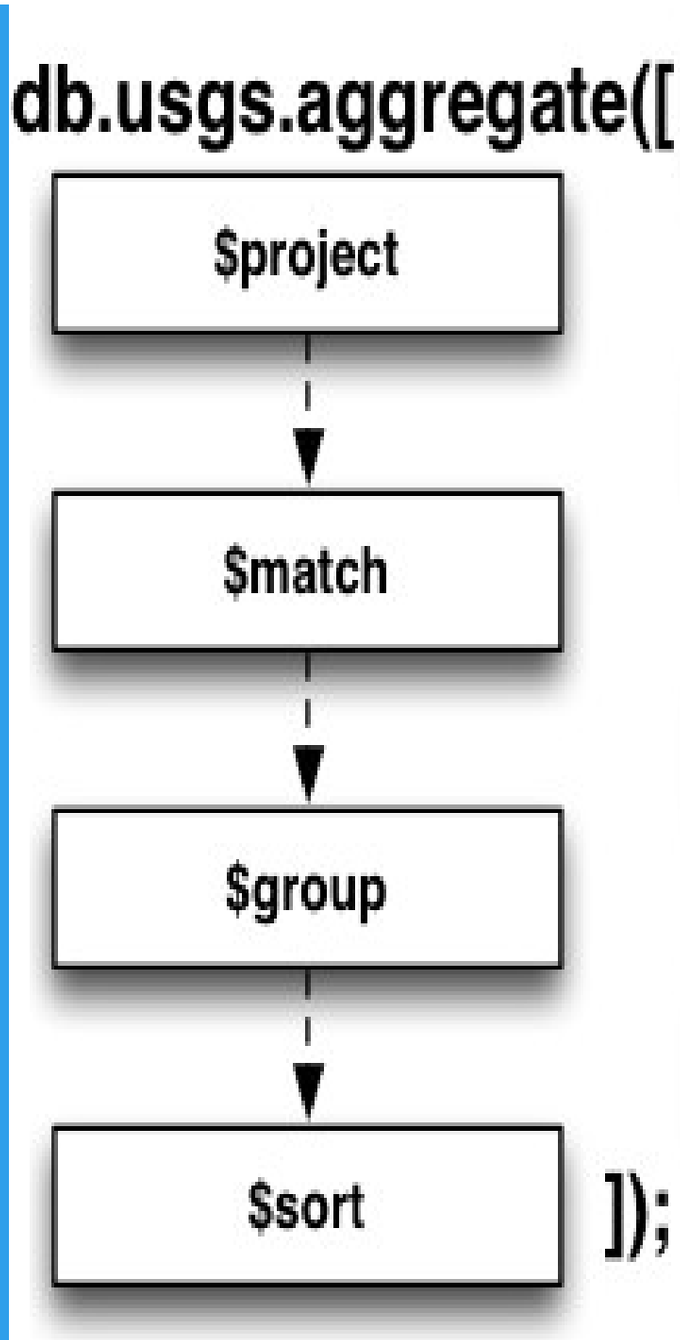
{ name: mongo, type: DB }

Agrupaciones

Propósito

En las bases de datos es muy importante disponer de consultas que permitan combinar diferentes elementos; por ejemplo el total vendido; o incluso los subtotales logrados por cada comercial.

A esta misma familia de consultas también pertenece la obtención de un valor que se calcula a partir de toda la colección, por ejemplo el máximo valor de una clave, o la media de todos los valores



Estructura

En Mongo las agrupaciones se hacen por etapas, donde cada elemento toma como entrada la salida del anterior.

Agrupaciones

Posibles componentes de la Tubería de agrupaciones

1. $project: Su función es "recolocar" el documento. Selecciona las claves que se usarán, y puede "elevar" claves que están en subdocumentos al nivel superior. Tras un paso $project habrá tantos documentos como inicialmente; pero con un formato que puede haber variado. (1:1)
2. $match: Filtra documentos, dejando solo los que vamos a utilizar. (n:1)
3. $group: Realiza la agregación (n:1)
4. $sort: Ordena. 1:1
5. $skip: Saltarse algunos a elementos n:1
6. $limit: Número de elementos máximo. n:1
7. $unwind: "aplana" datos de arrays, produciendo tantos elementos como elementos tenga el array. 1:n
8. $out: crea una colección nueva a partir de los datos. 1:1
9. $redact: Seguridad. Impide que algunos usuarios vean algunos documentos. n:1
10. $geonear: Se utiliza para búsquedas por posición (ver índices geoespaciales). n:1
11. $sample: permite elegir al azar unos cuantos documentos a modo de muestra. n:1
12. $lookup: join the varias colecciones

mongoDB
{ name: mongo, type: DB }

Agrupaciones: ejemplo

Vamos a usar el siguiente ejemplo para ver las distintas posibilidades de la agregación.

```
use running
db.sesiones.drop()
db.sesiones.insert({nombre:"Bertoldo", mes:"Marzo", distKm:6, tiempoMin:42})
db.sesiones.insert({nombre:"Herminia", mes:"Marzo", distKm:10, tiempoMin:60})
db.sesiones.insert({nombre:"Bertoldo", mes:"Marzo", distKm:2, tiempoMin:12})
db.sesiones.insert({nombre:"Herminia", mes:"Marzo", distKm:10, tiempoMin:61})
db.sesiones.insert({nombre:"Bertoldo", mes:"Abril", distKm:5, tiempoMin:33})
db.sesiones.insert({nombre:"Herminia", mes:"Abril", distKm:42, tiempoMin:285})
db.sesiones.insert({nombre:"Aniceto", mes:"Abril", distKm:5, tiempoMin:33})
```

mongoDB
{ name: mongo, type: DB }

Agrupaciones: ejemplo

Numero de sesiones que ha realizado cada persona

```
db.sesiones.aggregate(                    // aggregate significa que vamos a agrupar
                [                         // lista de operaciones, a realizar en secuencia
                 {$group:                 // en este caso solo una operación, agrupar
                       { _id:"$nombre",   // agrupamos por nombre
                         num_sesiones:    // nueva clave, num_sesiones
                              {$sum:1}    // cuenta el num.elementos en el grupo
                       }
                 }
                ]
               )
```

mongoDB
{ name: mongo, type: DB }

Agrupaciones: Pregunta

Consideramos la siguiente colección:

```
db.coord.drop()
 for (var i=0; i<5; i++) { for(j=0; j<4; j++){ db.coord.insert({x:i,y:j+i});  }  }
```

¿Cuántos elementos mostrará en pantalla la siguiente consulta?

```
db.coord.aggregate([ {$group:{_id:'$x'}} ])
```

Agrupaciones: ejemplo

Contar sesiones por nombre y mes

```
db.sesiones.aggregate(
 [
  {$group:
    { _id:{nombre:"$nombre", mes: "$mes"},
      num_sesiones: {$sum:1}
    }
  }
 ]
)
```

Agrupaciones: Pregunta

Consideramos la siguiente colección:

```
db.coord.drop()
  for (var i=0; i<5; i++) { for(j=0; j<4; j++){db.coord.insert({x:i,y:j+i});} }
```

¿Cuántos elementos mostrará en pantalla la siguiente consulta?

```
db.coord.aggregate([ {$group:{_id:{lax:'$x', la:'$y'}}} ])
```

mongoDB
{ name: mongo, type: DB }

Agrupaciones (funciones de agregación)

$sum es una de ellas; pero hay muchas más

1. $sum: suma (o incrementa)
2. $avg : calcula la media
3. $min: mínimo de los valores
4. $max: máximo
5. $push: recolecta valores en un array
6. $addToSet: Mete en un array los valorex que digamos, pero solo una vez
7. $first: obtiene el primer elemento del grupo, a menudo junto con sort
8. $last: obtiene el último elemento, a menudo junto con sort

Agrupaciones: $sum

Ya hemos visto cómo usarla para sumar, pero su propósito general es suma:

```
db.sesiones.aggregate(
 [
        {$group:
                { _id:{nombre:"$nombre"},
                  num_km: {$sum:'$distKm'}
                }
        }
 ]
)
```

Agrupaciones: $avg

Media en kilómetros por mes:

```
db.sesiones.aggregate(
 [
        {$group:
                { _id:{nombre:"$nombre", mes: "$mes"},
                  media: {$avg:'$distKm'} }
        }
 ]
)
```

Agrupaciones: $addToSet

$addToSet crea arrays agrupando elementos

Ejemplo: Supongamos que queremos saber qué distancias ha corrido cada persona. Agrupamos por el nombre y "coleccionamos" las distancias distintas:

```
db.sesiones.aggregate(
                    [
                     {$group:
                    { _id:{nombre:"$nombre"},
                      distancias: {$addToSet:'$distKm'}
                    }
                  }
                  ]
                )
```

Agrupaciones: $push

Análogo a $addToSet pero incluye las repeticiones

Ejemplo: queremos saber en cada mes qué distancias se han hecho en alguna sesión. Si una distancia se ha corrido varias veces en ese mes debe aparecer varias veces:

```
db.sesiones.aggregate(
                    [
                     {$group:
                    { _id:{mes:"$mes"},
                      distancias:{$push:'$distKm'}
                    }
                  }
                  ]
                )
```

Agrupaciones: $unwind

Es el reverso de $push; cuando tenemos documentos que contienen un array y queremos agrupar por valores del array, a veces conviene eliminar los arrays y convertirlos en múltiples documentos. En realidad estamos "normalizando" (primera forma normal).

Ejemplo: Queremos saber el número de personas con el que cuenta cada afición. ¿Cómo hacerlo?

```
db.gustos.aggregate([   {$unwind:'$aficiones'}   ] )
```

```
{"nombre" : "Bertoldo", "aficiones" : "siesta" }
{"nombre" : "Bertoldo", "aficiones" : "cine" }
{"nombre" : "Herminia", "aficiones" : "correr" }
{"nombre" : "Herminia", "aficiones" : "cine" }
{"nombre" : "Aniceta", "aficiones" : "viajar" }
{"nombre" : "Aniceta", "aficiones" : "cine" }
{"nombre" : "Godofredo", "aficiones" : "correr" }
{"nombre" : "Godofredo", "aficiones" : "montaña" }
{"nombre" : "Godofredo", "aficiones" : "cine" }
```

Agrupaciones: $unwind

Ahora es fácil pensar en la siguiente etapa: agrupar por aficiones

```
db.gustos.aggregate([
                {$unwind:'$aficiones'},
                {$group:
            {_id:'$aficiones',
             total:{$sum:1} } }
          ] )
```

Agrupaciones: $max, $min

Las típicas funciones para calcular máximos y mínimos. También en MongoDB

Ejemplo: queremos saber para cada persona el max y el min

```
db.sesiones.aggregate(
                [
                 {$group:
                       { _id:{nombre:"$nombre"},
                         maxdist:{$max:'$distKm'},
                         mindist:{$min:'$distKm'}
                       }
                 }
                ]
               )
```

mongoDB

{ name: mongo, type: DB }

Proyecciones: $project

Project está al nivel de $group, es una de las etapas permitidas en aggregate. Resulta muy útil para cambiar nombres de claves, introducir nuevas claves, etc. Es decir, su objetivo es "preparar" la agregación. En ocasiones se utiliza para crear nuevas colecciones.
Se suele usar con los operadores

1. booleanos: $and, $or, $not
2. strings :$concat, $toUpper, $toLower, $substr, $strcasecmp
3. operadores aritméticos: $abs, $add, $ceil, $divide, $exp, $floor, $ln, $log, $log10, $mod, $multiply, $pow, $sqrt, $substract, $trunc
4. conjuntos (arrays vistos como):$setEquals, $setInsersection, $setUnion, $setDifference, $setIsSubset, $anyElementTrue, $allElementsTrue
5. Arrays: $arrayElementAt, $concatArrays, $filter, $isArray, $size, $slice
6. Fechas: $datOfYear, $dayOfMonth, $dayOfWeek, $yrear, $month, $week, $hour, $minute, $second, $millisecond, $dateToString
7. Condicionales: cond, ifnull
8. otros: map, let

Proyecciones: ejemplo

Ejemplo: Queremos disponer de los datos de distancias recorridas en millas, sabiendo que una milla = 1,60934 km

```
db.sesiones.aggregate(
                    [
                     {$project:
                           {
                      distMillas:{'$multiply':['$distKm',1.60934]}
                     }
                }                    ]                )
```

Observación: $project solo incluye las claves que se indiquen, con excepción del _id que se incluye siempre pero se puede eliminar explícitamente el id con _id:0 . Si se quiere que una clave aparezca tal cual se puede poner clave:1 en la proyección

Aviso: A partir de la observación anterior, si ponemos clave:1 incluirá el valor original, pero ¿y si queremos que tenga el valor 1? En este caso y similares es útil $literal; se pondría clave:{'$literal':1}

Proyecciones: pregunta

Pregunta: Escribir una consulta para obtener a partir de documentos de la forma:

{_id:….,nombre:"Bertoldo", mes:"Marzo", distKm:6, tiempoMin:42}

Otros de la forma

{_id:….., name:"BERTOLDO", distKm:6}

Solución

```
db.sesiones.aggregate( [
  {$project: {
              nombre:{$toUpper:"$nombre"}, distKm:1
              }
  }
])
```

mongoDB

{ name: mongo, type: DB }

$match

Filtra elementos. Se puede usar tanto antes de la agregación (sería el where de SQL) como después (sería el having).

Ejemplo: Queremos obtener la media en kilómetros mensuales de cada corredor, pero solo para aquellos valores medios sobre 5km,

```
db.sesiones.aggregate( [
  {$group: { _id:{nombre:"$nombre", mes: "$mes"}, media: $avg:'$distKm'} } },
  {$match: {media:{$gt:5}} } ]
)
```

mongoDB
{ name: mongo, type: DB }

Agrupaciones: $first, $last

El primero y el último de la colección

Pregunta. Supongamos la colección

```
> > db.fun.find()
> { "_id" : 0, "a" : 0, "b" : 0, "c" : 21 }
> { "_id" : 1, "a" : 0, "b" : 0, "c" : 54 }
> { "_id" : 2, "a" : 0, "b" : 1, "c" : 52 }
> { "_id" : 3, "a" : 0, "b" : 1, "c" : 17 }
> { "_id" : 4, "a" : 1, "b" : 0, "c" : 22 }
> { "_id" : 5, "a" : 1, "b" : 0, "c" : 5 }
> { "_id" : 6, "a" : 1, "b" : 1, "c" : 87 }
> { "_id" : 7, "a" : 1, "b" : 1, "c" : 97 }
```

```
db.fun.aggregate([
{$match:{a:0}},
{$sort:{c:-1}},
{$group:{_id:"$a", c:{$first:"$c"}}}
])
```

Pregunta. ¿Cuánto valdrá c?

$sort

Ordena los resultados. Hay dos formas de ordenar:

- En memoria: es el método por defecto. Es el más rápido pero tiene como límite 100 Mb en la colección a ordenar
- Disco: más lento, pero sin límites; se obtiene añadiendo una etapa con forma {allowDiskUse:true}

Ejemplo: en el ejemplo de media de kilómetros por corredor y mes, ordenar por mes.

```
db.sesiones.aggregate(
[
  {$group:
            { _id:{nombre:"$nombre",
             mes: "$mes"},
              media: {$avg:'$distKm'}
             }
      },
  {$sort: {'_id.mes':1} }
]
)
```

Agrupaciones: $skip, $limit

Análogos al caso de find, y se usan siempre en combinación con sort. Útiles para obtener el mayor , el primero, etc

```
db.sesiones.aggregate(
[
{$group:
{ _id:{nombre:"$nombre"},
media: {$avg:'$distKm'}
}
},
{$sort: {media:-1} },
{$limit:1}
]
)
```

Etapas

¿Cómo calcular el número medio de sesiones por persona al mes? (es decir, se cuenta el número de sesiones por persona y mes y a continuación se hace la media de este dato)
Idea:
- Sabemos contar el número de sesiones por mes.
- Sabemos hacer la media de unos valores
¡usemos 2 etapas!

```
db.sesiones.aggregate(
 [
  {$group:
        { _id:{nombre:"$nombre", mes: "$mes"},
          sesiones:{$sum:1}
        }
  } ,

  {$group:
        {_id:'$_id.nombre',
         media:{$avg:'$sesiones'}
        }
  }
 ]
)
```

Agrupaciones: $out

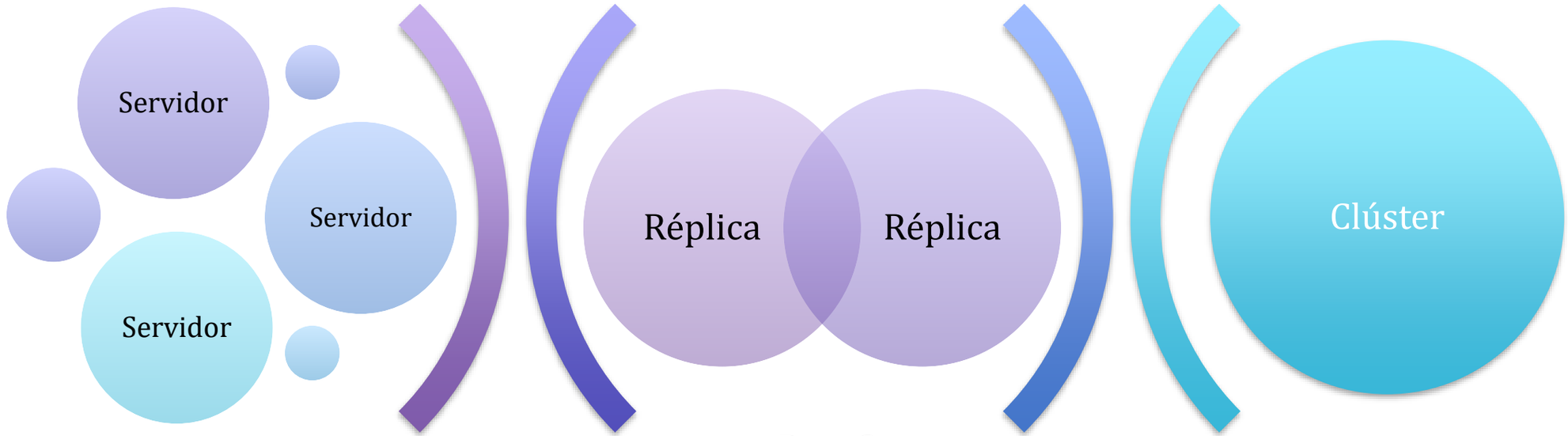
Crear una nueva colección con la salida. Es muy fácil de usar y cómo para ver lo que está produciendo una consulta en detalle

```
db.sesiones.aggregate(
                    [
                     {$group:
                           { _id:{nombre:"$nombre", mes: "$mes"},
                            sesiones:{$sum:1}
                           }
                    } ,

                    {$group:
                           {_id:'$_id.nombre',
                            media:{$avg:'$sesiones'}
                           }
                    },

                    {$out: 'sesiones_persona_mes'}
                   ]
                  )
```

mongoDB

{ name: mongo, type: DB }

Réplicas y Sharding



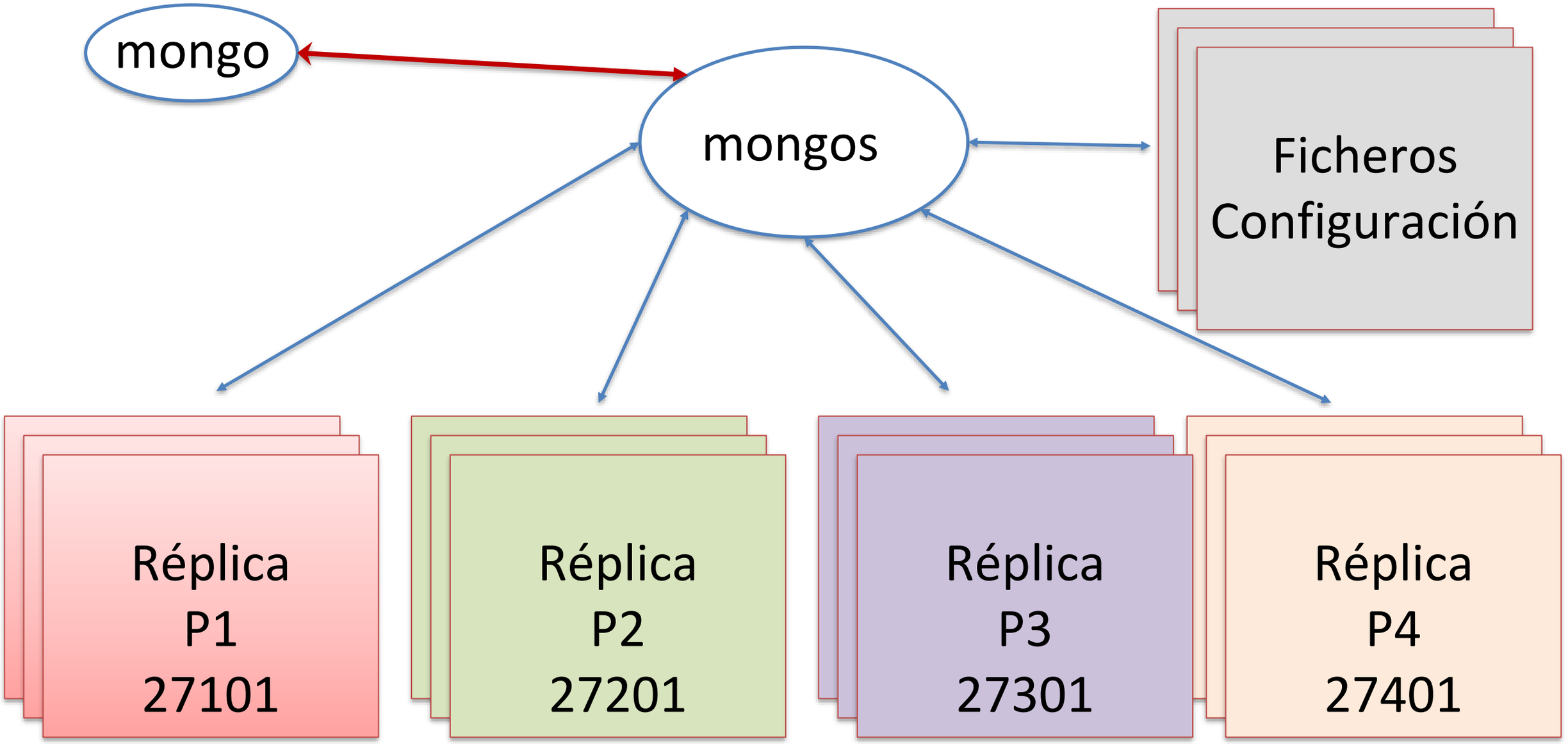
Paso 1
servidores individuales

Paso 2
Agrupación en
conjuntos de réplica

Paso 3
Formación del clúster

Réplicas y Sharding

Vamos a crear (simular en local) nuestro propio clúster



Réplicas y Sharding: pasos a seguir

1.- Servidores de configuración

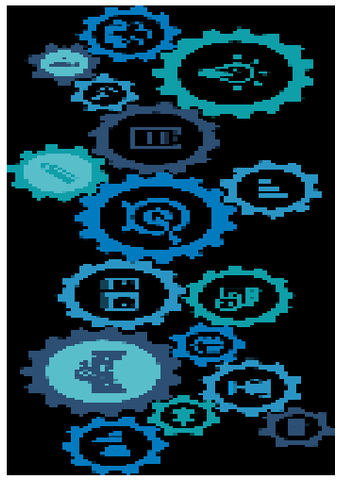
Los 3 servidores contendrán la misma información . Son los que indican qué datos están en qué réplica

2.-Conjuntos de réplica

Se crean como conjuntos independientes. Un mínimo de 3 miembros por conjunto

3.- Servicios mongos

Uno o varios, son con los que conectará el cliente, Hará de intemediario



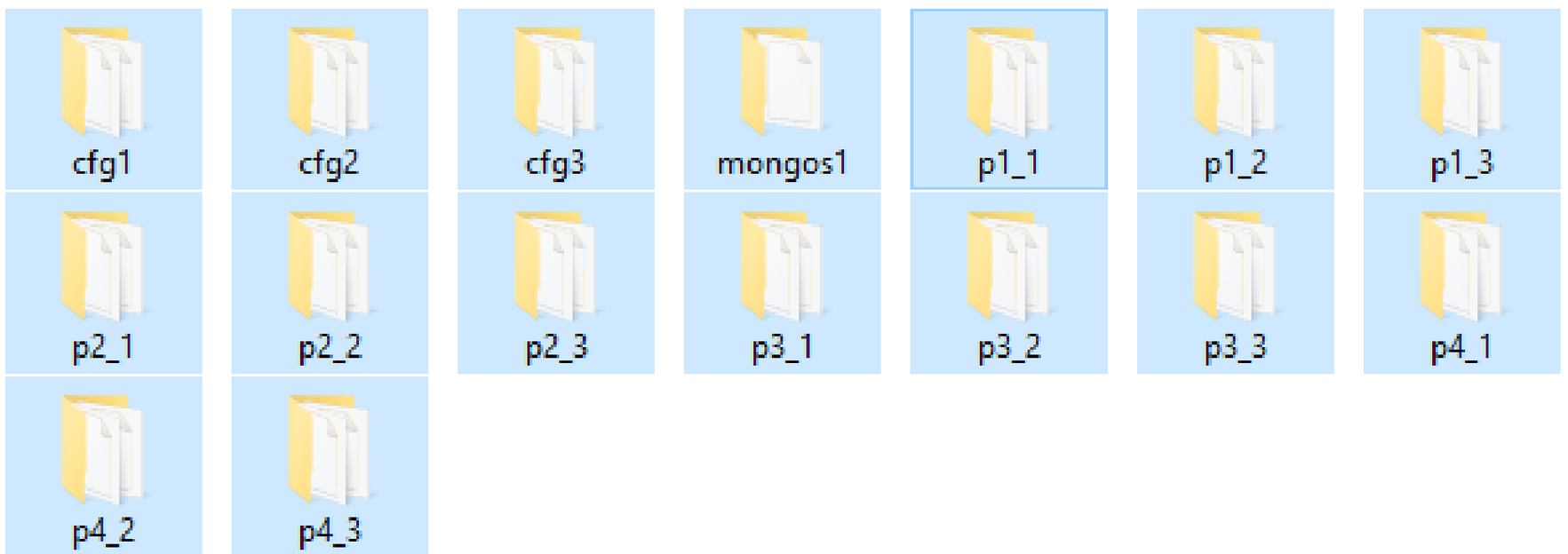
4.- Iniciar clúster

Se añaden los conjuntos de réplicas (uno de sus miembros) formando los shards, hasta tener un clúster como una unidad

Réplicas y Sharding: carpetas

Para simularlo en local empezamos por crear las carpetas de datos, una por servidor y otra más para el proceso *mongos*

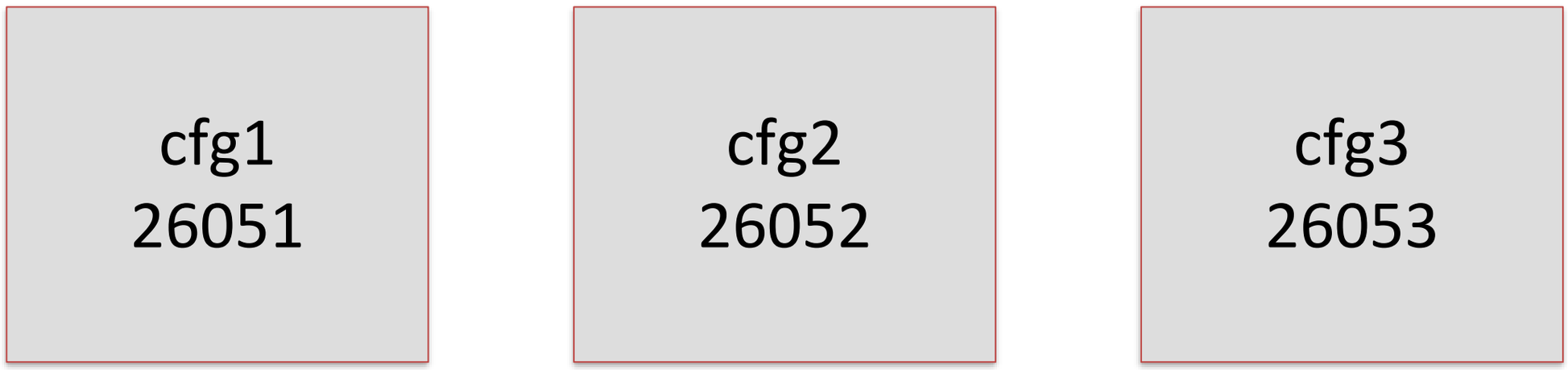
```
mkdir mongo
cd mongo
mkdir cfg1 cfg2 cfg3
mkdir p1_1 p1_2 p1_3 p2_1 p2_2 p2_3 p3_1 p3_2 p3_3 p4_1 p4_2 p4_3
mkdir mongos1
```



Réplicas y Sharding: servidores de configuración

Son instancias del proceso mongod con el parámetro --configsvr

```
start /b mongod --configsvr --dbpath cfg1 --port 26051 --logpath cfg1/log --logappend
start /b mongod --configsvr --dbpath cfg2 --port 26052 --logpath cfg2/log --logappend
start /b mongod --configsvr --dbpath cfg3 --port 26053 --logpath cfg3/log --logappend
```



Los puertos son arbitrarios; en un clúster real irán preferidos por el número IP del servidor o por su nombre en la red local

mongoDB
{ name: mongo, type: DB }

Réplicas y Sharding: servidores de datos

Los creamos de forma individual como instancias del proceso mongod indicando que serán parte de un clúster (--shardsvr) y de un conjunto réplica (--replSet *nombre*)

```
start /b mongod --shardsvr --replSet p1 --dbpath p1_1 --logpath p1_1/log --port 27101  --logappend --oplogSize 50
start /b mongod --shardsvr --replSet p1 --dbpath p1_2 --logpath p1_2/log --port 27102  --logappend --oplogSize 50
start /b mongod --shardsvr --replSet p1 --dbpath p1_3 --logpath p1_3/log --port 27103  --logappend --oplogSize 50
start /b mongod --shardsvr --replSet p2 --dbpath p2_1 --logpath p2_1/log --port 27201  --logappend --oplogSize 50
start /b mongod --shardsvr --replSet p2 --dbpath p2_2 --logpath p2_2/log --port 27202  --logappend --oplogSize 50
start /b mongod --shardsvr --replSet p2 --dbpath p2_3 --logpath p2_3/log --port 27203  --logappend --oplogSize 50
start /b mongod --shardsvr --replSet p3 --dbpath p3_1 --logpath p3_1/log --port 27301  --logappend --oplogSize 50
start /b mongod --shardsvr --replSet p3 --dbpath p3_2 --logpath p3_2/log --port 27302  --logappend --oplogSize 50
start /b mongod --shardsvr --replSet p3 --dbpath p3_3 --logpath p3_3/log --port 27303  --logappend --oplogSize 50
start /b mongod --shardsvr --replSet p4 --dbpath p4_1 --logpath p4_1/log --port 27401  --logappend --oplogSize 50
start /b mongod --shardsvr --replSet p4 --dbpath p4_2 --logpath p4_2/log --port 27402  --logappend --oplogSize 50
start /b mongod --shardsvr --replSet p4 --dbpath p4_3 --logpath p4_3/log --port 27403  --logappend --oplogSize 50
```

En Linux quitar start /b y añadir la opción --fork para iniciar en segundo plano

Réplicas y Sharding: mongos

El proceso *mongos* se indica indicándole cuáles son los ficheros de configuración. Se pueden añadir otras opciones como el tamaño del “chunk” la pieza de datos que se moverá de partición en partición (partición = conjunto réplica). El defecto es 64 Mb

```
start /b mongos --configdb 127.0.0.1:26051,127.0.0.1:26052,127.0.0.1:26053  --chunkSize 1
                        --logappend --logpath mongos1/log
```

Al no indicar explícitamente un puerto, arrancará “escuchando” el 27017, que es el puerto por defecto al que intenta conectar el cliente *mongo*

Réplicas y Sharding: creando conjuntos de réplica

Los servidores existen, pero los diferentes miembros del conjunto de réplica no se conocen entre sí. Para crear el primero, desde la consola hacer mongo --port 27101 y teclear:

```
config = { _id: "p1",
            members: [ {_id:1, host:"127.0.0.1:27101" },
                 {_id:2, host:"127.0.0.1:27102" },
                 {_id:3, host:"127.0.0.1:27103", priority:0, slaveDelay:240 } ]   }
rs.initiate(config)
```

¡Ya se conocen entre sí! El servidor actual pasará por varios estados: *OTHER* (autonegación), *SECONDARY* (falta de confianza) y *PRIMARY* (megalomanía). Teclear Ctr-D y desde el terminal:

```
mongo --port 27201 --eval "rs.initiate({ _id:'p2', members: [ {_id:1, host:'127.0.0.1:27201' },{_id:2,
host:'127.0.0.1:27202' },{_id:3, host:'127.0.0.1:27203', priority:0, slaveDelay:240 } ] })“

mongo --port 27301 --eval "rs.initiate({ _id:'p3', members: [ {_id:1, host:'127.0.0.1:27301' },{_id:2,
host:'127.0.0.1:27302' },{_id:3, host:'127.0.0.1:27303', priority:0, slaveDelay:240} ] })“

mongo --port 27401 --eval "rs.initiate({ _id:'p4', members: [ {_id:1, host:'127.0.0.1:27401' },{_id:2,
host:'127.0.0.1:27402' },{_id:3, host:'127.0.0.1:27403', priority:0, slaveDelay:240 } ] })"
```

Réplicas y Sharding: configuración de conjuntos de réplica

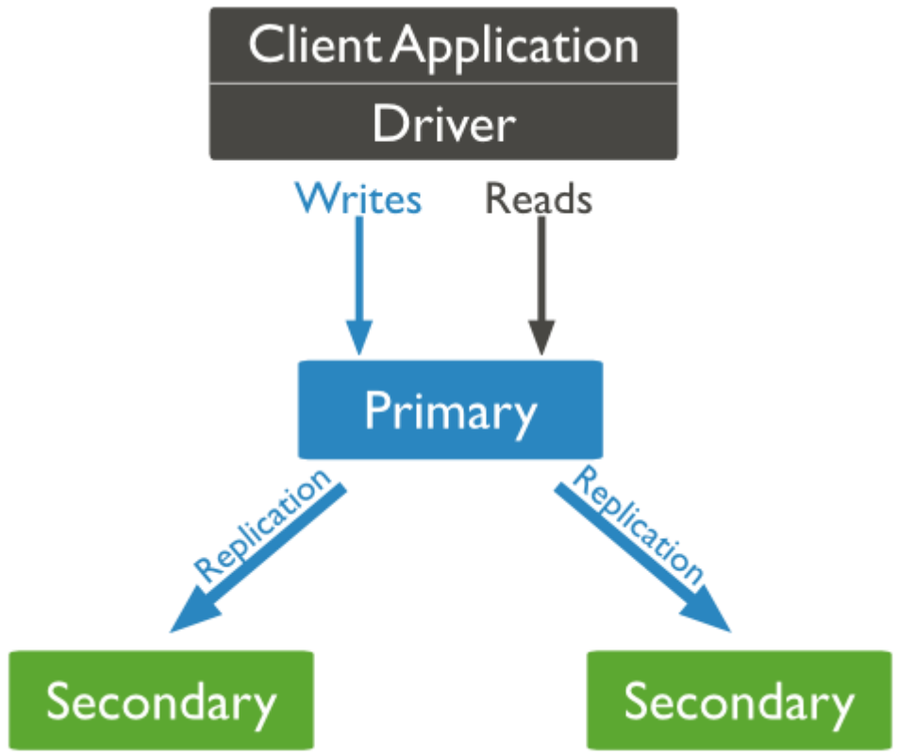
Vamos a tratar de entender menor los datos de configuración

```
config = { _id: "p1",
            members: [ {_id:1, host:"127.0.0.1:27101" },
                 {_id:2, host:"127.0.0.1:27102" },
                 {_id:3, host:"127.0.0.1:27103", priority:0, slaveDelay:240 } ]   }
```

- _id:”p1”: Se debe usar como identificador el mismo nombre que se uso junto con la opción --replSet al crear los servidores que componen la réplica
- Members: un array con los servidores. Cada servidor se especifica mediante un documento con la siguiente estructura:
  - _id: Un número único dentro del conjunto de réplica (sí puede existir en conjuntos réplica diferentes).
  - Host: Nombre del host y puerto en el que se creo el servidor
  - Priority: número real entre 0 y 1. 0 significa que nunca será servidor, 1 (por defecto) que se ofrece para el puesto.
  - slaveDelay: número de segundos de retraso al grabar escrituras
  - *arbiterOnly:true* indica que se trata de un servidor árbitro que no contendrá datos

Réplicas y Sharding: conjuntos de réplica

Hemos creado cuatro conjuntos de réplica. En cada caso habrá un primario y dos secundarios:



Réplicas y Sharding: creando el clúster

Ya tenemos las particiones funcionando de forma independiente entre sí. Para unificarlas en un solo clúster, desde la consola hacer mongo y teclear:

```
sh.addShard("p1/127.0.0.1:27101")
sh.addShard("p2/127.0.0.1:27201")
sh.addShard("p3/127.0.0.1:27301")
sh.addShard("p4/127.0.0.1:27401")
```

¡Ya tenemos clúster! Podemos aprovechar para crear alguna colección:

```
sh.enableSharding("piedras")
sh.shardCollection("piedras.minerales",{_id:1},true)
sh.enableBalancing("piedras.minerales")

db.minerales.insert({_id:"Calcita", color:"Jaspeada", brillo:"opaco", dureza:3, textura:"Arcillosa"})
db.minerales.insert({_id:"Plata", color:"Plateado",brillo:"Metálico", dureza:2.7, textura:"Lisa"})
db.minerales.insert({_id:"Cuarzo", color:"Incoloro",brillo:"Vítreo", dureza:7, textura:"Lisa"})
db.minerales.insert({_id:"Yeso", color:"Jaspeado", brillo:"Vítreo",dureza:[1.5,2],textura:"Arcillosa"})
db.minerales.insert({_id:"Halita", color:"Blanco",brillo:"Vítreo", dureza:2.5, textura:"Granulosa"})
db.minerales.insert({_id:"Grafito",color:"Gris",brillo:"Submetálico", dureza:1, textura:"Granulosa"})
```

Réplicas y Sharding: comprobación

Vamos a ver los datos accediendo directamente a los servidores.

```
mongo --port 27103 piedras

rs.slaveOk()
db.minerales.find()
```

¿Qué se obtiene? ¿Por qué?

```
mongo --port 27201 piedras

db.minerales.find()
```

¿Qué se obtiene? ¿Por qué?

mongoDB

{ name: mongo, type: DB }

Réplicas y Sharding: más datos

Metamos más datos, para ver qué ocurre:

```
mongo piedras

for(i=0;i<50000;i++){db.minerales.insert({_id:i,otros:"aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa"})}
```

Ahora hacer:

```
mongo --port 27201 piedras

db.minerales.find()
```

¿Qué se obtiene? ¿Por qué?

Experimentos

Probar a “tirar” el primario de p1 (27101) y volverlo a levantar
Antes de tirarlo insertar un elemento desde mongo (mongos)

¿Qué se obtiene? ¿Por qué?

Ahora probar a “levantarlo”

¿Qué papel ocupa? ¿Por qué?

Muchas Gracias