
UD8: BASES DE DATOS NOSQL

Parte 2. MongoDB: DML, Consultas Avanzadas e IDEs

Tema Extendido

Bases de Datos (BD)
CFGs DAM/DAW

Abelardo Martínez y Pau Miñana.
Basado y modificado de Sergio Badal.
Curso 2023-2024

ÍNDICE

- [1. Modificación de datos \(DML\)](#)
 - [1.1. Poblando la base de datos](#)
 - [1.2. Inserción de documentos](#)
 - [1.3. Actualización de documentos](#)
 - [1.4. Borrado de documentos](#)
- [2. Consultas avanzadas](#)
 - [2.1. Documentos embebidos y arrays](#)
 - [2.2. Agregaciones](#)
 - [2.2.1. Stages con equivalencia en el find](#)
 - [2.2.2. Stage para descomponer arrays](#)
 - [2.2.3. Enlazar colecciones. "JOIN"](#)
 - [2.2.4. Agrupamiento y funciones agregadas](#)
- [3. Instalación y uso de entornos gráficos \(IDE\)](#)
 - [3.1. MongoDB Compass](#)
 - [3.2. Robo 3T y Studio 3T](#)
- [4. Bibliografía](#)

1. Modificación de datos (DML)

1.1. Poblando la base de datos

Creamos una colección para poder ejecutar los ejemplos que vendrán después.

```
// limpiar pantalla (buena praxis)
cls
// conectar con la BD y borrarla (la crea si no existe)
use pruebas
db.dropDatabase()
use pruebas
// crear una nueva colección
db.createCollection("estudiantes")
// crea varios documentos
var j_est1 = {_id: 100, nombre:"Sergio", apellidos:"Gracia Merinda", email:"sergio@gmail.com", edad:19, nota:5}
var j_est2 = {_id: 200, nombre:"Pablo", apellidos:"Concar López", email:"graciamerinda@gmail.com", nota:6}
var j_est3 = {_id: 300, nombre:"Eva", apellidos:"Gredos Martínez", email:"gredoseva@gmail.com", edad:13, nota:7}
var j_est4 = {_id: 400, nombre:"Miranda", apellidos:"Aranda Bada", email:"indo@arandabada.com", edad:49, nota:8}
var j_est5 = {_id: 500, nombre:"Gabriel", apellidos:"Muro Menéndez", edad:12, nota:2}
var j_est6 = {_id: 600, nombre:"Ana", apellidos:"Gracia Merinda", email:"ana@gmail.com", edad:19, nota:4}
var j_est7 = {_id: 700, nombre:"Tania", apellidos:"Concar López", email:"tania@gmail.com", nota:6}
var j_est8 = {_id: 800, nombre:"Miguel", apellidos:"Gredos Martínez", email:"miguel@gmail.com", edad:39, nota:9}
var j_est9 = {_id: 900, nombre:"Álvaro", apellidos:"Aranda Bada", email:"diana@genial.com", edad:49, nota:1}
var j_est10 = {_id: 1000, nombre:"Diana", apellidos:"Muro Menéndez", edad:29, nota:7}

// inserta esos documentos en la colección
db.estudiantes.insertMany([j_est1, j_est2, j_est3, j_est4, j_est5, j_est6, j_est7, j_est8, j_est9, j_est10 ])
```

1.2. Inserción de documentos

⚠ MongoDB es **atómico por documento**, lo que quiere decir que si una operación falla, las anteriores sí que se ejecutan (**insertan, actualizan o borran**), el resto **no**.

Verás en la red que algunos autores usan los comandos **insert/update/delete** en vez de las opciones con **One/Many**. Estos comandos están desaconsejados, puesto que son obsoletos (deprecated).

Para insertar documentos en una colección debemos conectarnos a la base de datos que la contiene (comando **use**) y ejecutar los comandos **insertOne** o **insertMany**, que nos permiten insertar un único documento o varios al mismo tiempo, dentro de un *array*.

```
db.<coleccion>.insertOne(<json>)  
db.<coleccion>.insertMany([<json1>, ..., <jsonX>])
```

```
var j_doc1 = {nombre:"Juan"}  
var j_doc2 =  
{  
  nombre:"Eva",  
  apellidos:"García"  
}  
// Insertar documentos de uno en uno  
db.estudiantes.insertOne(j_doc1)  
db.estudiantes.insertOne(j_doc2)  
  
// Insertar varios documentos a la vez  
db.estudiantes.insertMany([j_doc1, j_doc2])  
  
// Equivalentes sin usar variables  
db.estudiantes.insertOne({nombre:"Juan"})  
db.estudiantes.insertOne({nombre:"Eva", apellidos:"García"})  
db.estudiantes.insertMany  
([ {nombre:"Juan"}, {nombre:"Eva", apellidos:"García"} ])
```

En los ejemplos anteriores estamos insertando varios documentos sobre la misma colección llamada *estudiantes*. Fíjate que primero estamos creando variables Javascript para hacer más legible el código, pero que no es estrictamente necesario.

Podemos crear e insertar documentos más complejos, pudiendo incluir, en una misma orden insertMany referida a una misma colección, documentos JSON de

varias páginas u otros de solo un campo. La manera de insertar ambos JSON será idéntica usando InsertOne o InsertMany según si queremos insertarlos manera individual o todos a la vez.

1.3. Actualización de documentos

Para actualizar documentos en una colección debemos conectarnos a la base de datos que la contiene (comando `use`) y ejecutar los comandos `updateOne` o `updateMany`, que nos permiten actualizar un único documento o todos los que cumplan la condición.

```
db.<coleccion>.updateOne(<filtro>,<acciones>,<opciones>)  
db.<coleccion>.updateMany(<filtro>,<acciones>,<opciones>)
```

- El filtro es equivalente al de la operación `find`. Con `updateOne` solo se actualiza el primer documento que cumpla la condición, ordenado por `_id`.
- Las acciones son las modificaciones que se quieren hacer en los documentos que cumplan la condición. Es un documento JSON con cada una de las acciones a realizar. Usa los siguientes operadores (empiezan por `$`):
 - `$set:<json>`: Actualiza campos, especificados con un JSON. Si algún campo no existe, lo crea. `$set:{campo1:"sergio", campo2: 5}`
 - `$unset:<json>`: Elimina campos, el JSON no necesita valores, solo los campos a eliminar. `$unset:{campo1:""}`
 - `$inc:<json>`: Incrementa/decrementa el valor de campo numéricos en la cantidad especificada. `$inc:{campo1:5, campo2:-3}`
 - `$push:<json>`: Añade un valor a un campo de tipo array. `$push:{campoArray:5}`
 - `$rename:<json>`: Cambia el nombre de campos. `$rename:{campo1:"nuevo_nombre"}`
- Las opciones pueden omitirse. **No llevan \$**. En este caso solo estudiaremos una:
 - `{upsert:true}`: Si no se encuentra ningún documento que cumpla la condición (filtro), *inserta un nuevo documento* con los campos especificados en el filtro y las acciones. `$set` ya añadía campos, en este caso se trata de añadir nuevos documentos a la colección. Los campos en el filtro también se incluyen en el nuevo documento, para que se cumpla la condición, es decir, que esta opción crea el documento que cumpliría la actualización.

El filtro y las acciones siempre son obligatorios, (aunque el filtro se puede dejar vacío `{}` para aplicar a todos los documentos).

```
var j_filtro    = {nombre:"Pedro"}
var j_set       = {nota:5}
var j_accion    = {$set:j_set}

// Poner un 5 de nota a todos los estudiantes llamados Pedro:
db.estudiantes.updateMany(j_filtro, j_accion)

// Lo mismo pero solo al primer Pedro que se encuentre:
db.estudiantes.updateOne(j_filtro, j_accion)
// Con updateOne se recomienda usar el _id en el filtro para
// asegurar que se actualiza el documento deseado

// Poner la nota a 5 a todos los estudiantes (sin filtro)
db.estudiantes.updateMany({}, j_accion)
// Se debe dejar un documento vacío como filtro, como en el find

// Si no hubiera ningún Pedro, añadirlo con nota 5
var j_opciones = {upsert:true}
db.estudiantes.updateMany(j_filtro, j_accion, j_opciones)
// Como no teníamos ningún Pedro, se crea ahora.

// Borrado de los VALORES de campos, con $set
// Borra la nota de Pedro, ahora está a ""
var j_set       = {nota:""}
var j_accion    = {$set:j_set}
db.estudiantes.updateMany(j_filtro, j_accion)

// Borrado de CAMPOS, con $unset
// Elimina el campo nota de Pedro
var j_accion    = {$unset:j_set}
db.estudiantes.updateMany(j_filtro, j_accion)
```

Ten en cuenta que aunque estos ejemplos usen un solo campo en el \$set/\$unset, se pueden añadir a los mismos tantos campos como se necesite.

Ejemplos

Sube un punto la nota de todos los estudiantes mayores de 30 años.
Muestra el antes y el después (solo apellidos y nota).
Devuelve la base de datos a su estado original.

```
// Consultamos antes de actualizar
var j_proyeccion = {apellidos: 1, nota:1, _id: 0}
var j_valor      = {$gt:30}
var j_filtro     = {edad: j_valor}
db.estudiantes.find(j_filtro,j_proyeccion)
// Actualizamos y consultamos de nuevo
var j_valor      = {nota: 1}
```

```
var j_accion      = {$inc: j_valor}
db.estudiantes.updateMany(j_filtro, j_accion)
db.estudiantes.find(j_filtro,j_proyeccion)
// Bajamos la nota un punto para deshacer los cambios
var j_valor      = {nota: -1}
var j_accion      = {$inc: j_valor}
db.estudiantes.updateMany(j_filtro, j_accion)
db.estudiantes.find(j_filtro,j_proyeccion)
```

```
pruebas> var j_proyeccion = {apellidos: 1, nota:1, _id: 0}
pruebas> var j_valor      = {$gt:30}
pruebas> var j_filtro     = {edad: j_valor}
pruebas> db.estudiantes.find(j_filtro,j_proyeccion)
[
  { apellidos: 'Aranda Bada', nota: 8 },
  { apellidos: 'Gredos Martínez', nota: 9 },
  { apellidos: 'Aranda Bada', nota: 1 }
]
pruebas> var j_valor      = {nota: 1}
pruebas> var j_accion     = {$inc: j_valor}
pruebas> db.estudiantes.updateMany(j_filtro, j_accion)
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 3,
  modifiedCount: 3,
  upsertedCount: 0
}
pruebas> db.estudiantes.find(j_filtro,j_proyeccion)
[
  { apellidos: 'Aranda Bada', nota: 9 },
  { apellidos: 'Gredos Martínez', nota: 10 },
  { apellidos: 'Aranda Bada', nota: 2 }
]
pruebas> var j_valor      = {nota: -1}
pruebas> var j_accion     = {$inc: j_valor}
pruebas> db.estudiantes.updateMany(j_filtro, j_accion)
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 3,
  modifiedCount: 3,
  upsertedCount: 0
}
pruebas> db.estudiantes.find(j_filtro,j_proyeccion)
[
  { apellidos: 'Aranda Bada', nota: 8 },
  { apellidos: 'Gredos Martínez', nota: 9 },
  { apellidos: 'Aranda Bada', nota: 1 }
]
```

Cambia el email *graciamerinda@gmail.com* por *gracia@gmail.com*. Muestra el antes y el después (solo los emails).

Devuelve la base de datos a su estado original.

```
var j_proyeccion = {email: 1, _id: 0}
db.estudiantes.find({}, j_proyeccion)
var j_filtro      = {email: "graciamerinda@gmail.com"}
var j_valor       = {email: "gracia@gmail.com"}
var j_accion      = {$set: j_valor}
db.estudiantes.updateMany(j_filtro, j_accion)
db.estudiantes.find({}, j_proyeccion)
var j_filtro      = {email: "gracia@gmail.com"}
var j_valor       = {email: "graciamerinda@gmail.com"}
var j_accion      = {$set: j_valor}
db.estudiantes.updateMany(j_filtro, j_accion)
db.estudiantes.find({}, j_proyeccion)
```

```
pruebas> var j_proyeccion = {email: 1, _id: 0}
pruebas> db.estudiantes.find({}, j_proyeccion)
[
  { email: 'sergio@gmail.com' },
  { email: 'graciamerinda@gmail.com' },
  { email: 'gredoseva@gmail.com' },
  { email: 'indo@arandabada.com' },
  {},
  { email: 'ana@gmail.com' },
  { email: 'tania@gmail.com' },
  { email: 'miguel@gmail.com' },
  { email: 'diana@genial.com' },
  {}
]
pruebas> var j_filtro      = {email: "graciamerinda@gmail.com"}
pruebas> var j_valor       = {email: "gracia@gmail.com"}
pruebas> var j_accion      = {$set: j_valor}
pruebas> db.estudiantes.updateMany(j_filtro, j_accion)
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
pruebas> db.estudiantes.find({}, j_proyeccion)
[
  { email: 'sergio@gmail.com' },
  { email: 'gracia@gmail.com' },
  { email: 'gredoseva@gmail.com' },
  { email: 'indo@arandabada.com' },
  {},
  { email: 'ana@gmail.com' },
  { email: 'tania@gmail.com' },
  { email: 'miguel@gmail.com' },
  { email: 'diana@genial.com' },
  {}
]
```


Baja un punto a todos los estudiantes. Muestra el antes y el después (solo apellidos y nota). Devuelve la base de datos a su estado original.

```
var j_proyeccion = {apellidos: 1, nota:1, _id: 0}
db.estudiantes.find({},j_proyeccion)
var j_valor      = {nota: -1}
var j_accion     = {$inc: j_valor}
db.estudiantes.updateMany({}, j_accion)
db.estudiantes.find({},j_proyeccion)
var j_valor      = {nota: 1}
var j_accion     = {$inc: j_valor}
db.estudiantes.updateMany({}, j_accion)
db.estudiantes.find({},j_proyeccion)
```

```
pruebas> var j_proyeccion = {apellidos: 1, nota:1, _id: 0}
pruebas> db.estudiantes.find({},j_proyeccion)
[
  { apellidos: 'Gracia Merinda', nota: 5 },
  { apellidos: 'Concar López', nota: 6 },
  { apellidos: 'Gredos Martínez', nota: 7 },
  { apellidos: 'Aranda Bada', nota: 8 },
  { apellidos: 'Muro Menéndez', nota: 2 },
  { apellidos: 'Gracia Merinda', nota: 4 },
  { apellidos: 'Concar López', nota: 6 },
  { apellidos: 'Gredos Martínez', nota: 9 },
  { apellidos: 'Aranda Bada', nota: 1 },
  { apellidos: 'Muro Menéndez', nota: 7 }
]
pruebas> var j_valor      = {nota: -1}
pruebas> var j_accion     = {$inc: j_valor}
pruebas> db.estudiantes.updateMany({}, j_accion)
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 10,
  modifiedCount: 10,
  upsertedCount: 0
}
pruebas> db.estudiantes.find({},j_proyeccion)
[
  { apellidos: 'Gracia Merinda', nota: 4 },
  { apellidos: 'Concar López', nota: 5 },
  { apellidos: 'Gredos Martínez', nota: 6 },
  { apellidos: 'Aranda Bada', nota: 7 },
  { apellidos: 'Muro Menéndez', nota: 1 },
  { apellidos: 'Gracia Merinda', nota: 3 },
  { apellidos: 'Concar López', nota: 5 },
  { apellidos: 'Gredos Martínez', nota: 8 },
  { apellidos: 'Aranda Bada', nota: 0 },
  { apellidos: 'Muro Menéndez', nota: 6 }
]
pruebas> var j_valor      = {nota: 1}
pruebas> var j_accion     = {$inc: j_valor}
pruebas> db.estudiantes.updateMany({}, j_accion)
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 10,
  modifiedCount: 10,
  upsertedCount: 0
}
```

Sube un punto al primer estudiante que encuentres, borra su email y sus apellidos, y elimina su campo edad.

Muestra el antes y el después (solo `_id`, apellidos, nota y edad).

```
var j_proyeccion = {apellidos: 1, nota:1, edad:1}
db.estudiantes.find({},j_proyeccion).limit(1)
var j_valor1      = {apellidos: "", email:""}
var j_valor2      = {edad: ""}
var j_valor3      = {nota: 1}
var j_accion      = {$set: j_valor1,
                    $unset: j_valor2, $inc: j_valor3}
db.estudiantes.updateOne({}, j_accion)
db.estudiantes.find({},j_proyeccion).limit(1)
```

```
pruebas> var j_proyeccion = {apellidos: 1, nota:1, edad:1}

pruebas> db.estudiantes.find({},j_proyeccion).limit(1)
[ { _id: 100, apellidos: 'Gracia Merinda', edad: 19, nota: 5 } ]
pruebas> var j_valor1      = {apellidos: "", email:""}

pruebas> var j_valor2      = {edad: ""}

pruebas> var j_valor3      = {nota: 1}

pruebas> var j_accion      = {$set: j_valor1, $unset: j_valor2, $inc: j_valor3}

pruebas> db.estudiantes.updateOne({}, j_accion)
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
pruebas> db.estudiantes.find({},j_proyeccion).limit(1)
[ { _id: 100, apellidos: '', nota: 6 } ]
```

1.4. Borrado de documentos

Para borrar documentos en una colección debemos conectarnos a la base de datos que la contiene (comando `use`) y ejecutar los comandos `deleteOne` o `deleteMany`, que nos permiten borrar un único documento (el primero que encuentre ordenado por `_id`) o todos los que cumplan la condición.

```
db.<coleccion>.deleteOne(<filtro>)
db.<coleccion>.deleteMany(<filtro>)
```

```
// Borrado de todos los documentos que tengan como nombre "Pedro"
var j_filtro      = {nombre:"Pedro"}
db.estudiantes.deleteMany(j_filtro)
// o también
var j_valor       = {$eq:"Pedro"}
var j_filtro      = {nombre: j_valor}
db.estudiantes.deleteMany(j_filtro)

// Borrado del primer documento que encuentre con nota > 5
var j_valor       = {$gt:5}
var j_filtro      = {nota: j_valor}
db.estudiantes.deleteOne(j_filtro)

// Borrado de todos los documentos de una colección (truncate)
db.estudiantes.deleteMany({})
```

Recuerda que se necesita incluir siempre un filtro, aunque sea vacío `{}` (esto borrará todos los documentos de la colección).

Ejemplos

Borra los estudiantes mayores de 30 años.

Muestra el antes y el después (solo apellidos y edad).

```
var j_proyeccion = {apellidos: 1, edad:1, _id: 0}
var j_valor      = {$gt:30}
var j_filtro     = {edad: j_valor}
db.estudiantes.find(j_filtro,j_proyeccion)
db.estudiantes.deleteMany(j_filtro)
db.estudiantes.find(j_filtro,j_proyeccion)
```

```
pruebas> var j_proyeccion = {apellidos: 1, edad:1, _id: 0}
pruebas> var j_valor      = {$gt:30}
pruebas> var j_filtro     = {edad: j_valor}
pruebas> db.estudiantes.find(j_filtro,j_proyeccion)
[
  { apellidos: 'Aranda Bada', edad: 49 },
  { apellidos: 'Gredos Martínez', edad: 39 },
  { apellidos: 'Aranda Bada', edad: 49 }
]
pruebas> db.estudiantes.deleteMany(j_filtro)
{ acknowledged: true, deletedCount: 3 }
pruebas> db.estudiantes.find(j_filtro,j_proyeccion)
pruebas> █
```

Borra el primer estudiante que tenga nota <5.

Muestra el antes y el después (solo apellidos y nota).

```
var j_proyeccion = {apellidos: 1, nota:1, _id: 0}
var j_valor      = {$lt:5}
var j_filtro     = {nota: j_valor}
db.estudiantes.find(j_filtro, j_proyeccion).limit(1)
db.estudiantes.deleteOne(j_filtro)
db.estudiantes.find(j_filtro, j_proyeccion)
```

```
pruebas> var j_proyeccion = {apellidos: 1, nota:1, _id: 0}
pruebas> var j_valor      = {$lt:5}
pruebas> var j_filtro     = {nota: j_valor}
pruebas> db.estudiantes.find(j_filtro, j_proyeccion).limit(1)
[ { apellidos: 'Muro Menéndez', nota: 2 } ]
pruebas> db.estudiantes.deleteOne(j_filtro)
{ acknowledged: true, deletedCount: 1 }
pruebas> db.estudiantes.find(j_filtro, j_proyeccion)
[ { apellidos: 'Gracia Merinda', nota: 4 } ]
```

Borra el primer estudiante que encuentres.

Muestra el antes y el después (solo apellidos y _id).

```
var j_proyeccion = {apellidos: 1}
db.estudiantes.find({}, j_proyeccion).limit(1)
db.estudiantes.deleteOne({})
db.estudiantes.find({}, j_proyeccion).limit(1)
```

```
pruebas> var j_proyeccion = {apellidos: 1}
pruebas> db.estudiantes.find({}, j_proyeccion).limit(1)
[ { _id: 100, apellidos: '' } ]
pruebas> db.estudiantes.deleteOne({})
{ acknowledged: true, deletedCount: 1 }
pruebas> db.estudiantes.find({}, j_proyeccion).limit(1)
[ { _id: 200, apellidos: 'Concar López' } ]
```

2. Consultas avanzadas


2.1. Documentos embebidos y arrays

Los documentos embebidos son documentos que están dentro de otros documentos como si fuesen un campo más. Los estudiantes anteriores podrían tener un campo *asignatura* que fuese un documento embebido o incluso un array, tanto de valores simples como de documentos embebidos:

```
// Campo notas como array de valores, se desconoce la asignatura
notas: [ 6, 8]
// Campo asignatura como objeto, nombre+nota
asignatura: {Nombre:"Bases de Datos", Nota:6}
// Campo asignatura como array de objetos
asignatura: [
    {nombre:"Bases de Datos", nota:6},
    {nombre:"Programación", nota:8}
]
```

Recuerda que en el mismo Array no tienen por qué coincidir los tipos de datos de cada elemento y que se pueden anidar arrays y documentos embebidos.

Consultas en documentos embebidos

 Para hacer referencia a los campos dentro de un documento embebido en un filtro/proyección se necesita usar el **nombre del documento seguido de un punto y el nombre del campo** (y así sucesivamente si hubiese más documentos embebidos) **todo entre comillas** o los puntos provocarán un error en la expresión.

Actualizamos la base de datos de estudiantes, suponiendo que partimos de los datos originales del punto 1.1. Ponemos algunas notas (*asignatura*) en modo objeto, otras como array de objetos y finalmente borramos el campo *nota* original.

```
var j_filtro      = {nombre:"Sergio"}
var j_set         = {asignatura:{nombre:"BD", nota:6 }}
var j_accion      = {$set:j_set}
var j_opciones    = {upsert:true}
db.estudiantes.updateOne(j_filtro, j_accion, j_opciones)
var j_filtro      = {nombre:"Pablo"}
var j_set         = {asignatura:{nombre:"Prog", nota:4 }}
var j_accion      = {$set:j_set}
db.estudiantes.updateOne(j_filtro, j_accion, j_opciones)
var j_filtro      = {nombre:"Eva"}
```

```

var j_set          = {asignatura:{nombre:"BD", nota:8 }}
var j_accion       = {$set:j_set}
db.estudiantes.updateOne(j_filtro, j_accion, j_opciones)
var j_filtro       = {nombre:"Miranda"}
var j_set          = {asignatura:[{nombre:"BD", nota:6 },
{nombre:"Prog", nota:8 }]}
var j_accion       = {$set:j_set}
db.estudiantes.updateOne(j_filtro, j_accion, j_opciones)
var j_filtro       = {nombre:"Gabriel"}
var j_set          = {asignatura:[{nombre:"BD", nota:2 },
{nombre:"Prog", nota:5 }]}
var j_accion       = {$set:j_set}
db.estudiantes.updateOne(j_filtro, j_accion, j_opciones)
var j_set          = {nota:""}
var j_accion       = {$unset:j_set}
db.estudiantes.updateMany({}, j_accion)

```

Listar los datos de las asignaturas de los estudiantes es fácil, solo hay que añadir *asignatura* a la proyección, pero si se quiere mostrar/filtrar usando solo el campo *nota* o *nombre* embebidos en *asignatura*, se debe hacer referencia a ellos con el nombre del objeto padre (asignatura) seguido de un punto y el nombre del campo hijo, **todo entre comillas**.

```

// Consulta las notas de los estudiantes con asignatura BD
var j_filtro       = {"asignatura.nombre": "BD"}
var j_proyeccion   = {nombre: 1,"asignatura.nota": 1, _id: 0}
db.estudiantes.find(j_filtro,j_proyeccion)

```

```

pruebas> var j_filtro       = {"asignatura.nombre": "BD"}
pruebas> var j_proyeccion   = {nombre: 1,"asignatura.nota": 1, _id: 0}
pruebas> db.estudiantes.find(j_filtro,j_proyeccion)
[
  { nombre: 'Sergio', asignatura: { nota: 6 } },
  { nombre: 'Eva', asignatura: { nota: 8 } },
  { nombre: 'Miranda', asignatura: [ { nota: 6 }, { nota: 8 } ] },
  { nombre: 'Gabriel', asignatura: [ { nota: 2 }, { nota: 5 } ] }
]

```

Seguramente este resultado te haya sorprendido, ya que las notas mostradas no son las de "BD" sino todas las notas de los estudiantes que tienen "BD" en *asignatura*.

El comando *find* ejecuta la búsqueda tanto en los documentos embebidos simples como en los de dentro del Array. Esto permite filtrar de golpe con condiciones de campos que estén en un array o no mientras el nombre sea similar. Pero por otro

lado hay un inconveniente para mostrar los resultados, ya que *find* filtra documentos enteros que tengan algún *"asignatura.nombre":"BD"* y muestra todos los *"asignatura.nota"* de ese documento, ni el filtro ni la proyección pueden cambiar esto, con lo que si está en un array con otras asignaturas no da la nota de BD sino todas las notas del Array. Veremos más adelante cómo tratar estos casos.

Otra opción, aunque menos habitual, es **filtrar elementos usando directamente el índice del Array**, con *"NombreArray.X"* siendo X el índice del Array. Por ejemplo:

- *"asignatura.0"* se refiere al primer documento o valor en el Array asignatura.
- *"asignatura.1"* se refiere al segundo documento o valor en el Array asignatura
- *"asignatura.1.nota"* busca la nota del segundo elemento del Array, si es un documento.

⚠ La referencia por índice sólo sirve para filtrar, no para las proyecciones, es decir, se puede usar por ejemplo {"asignatura.0.nota:1"} como filtro y devuelve todos los documentos que tengan un 1 como nota de la primera asignatura del Array, pero si se usa como proyección no muestra las notas de la primera asignatura de los documentos, sale vacío.

2.2. Agregaciones

Las agregaciones permiten transformar y combinar documentos en una colección. Esto nos ayuda a realizar consultas más complejas, entre otras cosas, aplicar agrupaciones, usar funciones agregadas, unir colecciones como si de un JOIN de SQL se tratara, o superar las limitaciones que nos hemos encontrado para separar los elementos de un Array y mostrar sólo los que nos interesan.

Para las agregaciones se construye un *pipeline* (secuencia de comandos) que procesa un conjunto de varias fases o *stages*. Cada una de las fases se determina con un documento y se van ejecutando sucesivamente desde la primera a la última. Se supone que todos los documentos (*stages*) se deberían pasar dentro de un único Array pero lo cierto es que el comando admite igualmente una lista de documentos sin integrarlos dentro de un Array.

Se debe tener presente que *stages* del mismo tipo se pueden aplicar varias veces, es decir que se podría, por ejemplo, aplicar una fase de filtrado, una para separar un Array y volver a filtrar.

```
db.<coleccion>.aggregate([<fase1>, <fase2>, ..., <faseX>])
```


La agregación es una herramienta muy poderosa con multitud de posibilidades. Por motivos de tiempo nos limitaremos a algunas de las fases más comunes que ofrece y con opciones limitadas, para un uso básico. Los nombres de las fases siempre empiezan por \$.

2.2.1. Stages con equivalencia en el find

- `$match:<filtro>`: Similar al filtro de la función *find* o al *WHERE* de SQL.
- `$project:<proyección>`: Parecido a la proyección del find o al SELECT de SQL. Pero ahora no son solo los campos a mostrar, son los que pasan a la siguiente fase, por tanto si no se incluye un campo en la proyección las siguientes fases no podrán usarlo.
- `$sort:<orden>`: Similar a la operación *sort* o al *ORDER BY* de SQL.
- `$limit: X`: Similar a la operación *limit* del *find*, restringe el resultado a los primeros X documentos.
- `$count: "campo"`: Devuelve un documento con un *campo* que tiene como valor el número de documentos. Los campos de la consulta no pasan a la siguiente fase.

Ejemplos

Filtrado: Mostrar los estudiantes con *edad* = 19

```
var j_filtro      = {edad:19}
var stage_filtro  = {$match:j_filtro}
db.estudiantes.aggregate([stage_filtro])
// Equivalente a db.estudiantes.find(j_filtro)
```

```
pruebas> db.estudiantes.aggregate([stage_filtro])
[
  {
    _id: 100,
    nombre: 'Sergio',
    apellidos: 'Gracia Merinda',
    email: 'sergio@gmail.com',
    edad: 19,
    asignatura: { nombre: 'BD', nota: 6 }
  },
  {
    _id: 600,
    nombre: 'Ana',
    apellidos: 'Gracia Merinda',
    email: 'ana@gmail.com',
    edad: 19
  }
]
```


Proyección: Mostrar el *nombre* de los estudiantes con *edad* = 19

```
var j_filtro          = {edad:19}
var stage_filtro      = {$match:j_filtro}
var j_proy            = {nombre:1,_id:0}
var stage_proy        = {$project:j_proy}
db.estudiantes.aggregate([stage_filtro, stage_proy])
```

```
pruebas> db.estudiantes.aggregate([stage_filtro, stage_proy])
[ { nombre: 'Sergio' }, { nombre: 'Ana' } ]
```

Si aplicásemos primero la proyección que el filtro

`db.estudiantes.aggregate([stage_proy, stage_filtro])` no se mostraría nada, ya que cuando la consulta llega al filtro el único campo que tiene es el *nombre*.

Conteo: Contar los estudiantes con *edad* = 19

```
var j_filtro          = {edad:19}
var stage_filtro      = {$match:j_filtro}
var stage_cuenta      = {$count:"Estudiantes_19"}
db.estudiantes.aggregate([stage_filtro, stage_cuenta])
// Parecido a db.estudiantes.find(j_filtro).count()
// o db.estudiantes.countDocuments(j_filtro)
// pero devuelve un documento no un número
```

```
pruebas> db.estudiantes.aggregate([stage_filtro, stage_cuenta])
[ { Estudiantes_19: 2 } ]
pruebas> db.estudiantes.find(j_filtro).count()
2
```

Ordenación y Límite: Mostrar el *nombre* y la *edad* de los 3 estudiantes con más *edad* (obviamos que puede haber gente con la misma edad y no ser realmente 3 estudiantes, ya que aquí no veremos subconsultas ni otras opciones)

```
var j_orden           = {edad:-1}
var stage_orden       = {$sort:j_orden}
var j_proy            = {nombre:1,edad:1,_id:0}
var stage_proy        = {$project:j_proy}
var stage_limit       = {$limit:3}
db.estudiantes.aggregate([stage_proy, stage_orden, stage_limit])
// Similar a db.estudiantes.find({}, j_proy).sort(j_orden).limit(3)
```

```
pruebas> db.estudiantes.aggregate([stage_proy,stage_orden,stage_limit])
[
  { nombre: 'Álvaro', edad: 49 },
  { nombre: 'Miranda', edad: 49 },
  { nombre: 'Miguel', edad: 39 }
]
```

2.2.2. Stage para descomponer arrays

La agregación permite aplicar una fase para separar los elementos de un array, de modo que en la salida se obtiene una copia del documento por cada elemento del array, que pasa a ser un campo con un único dato/documento.

- `$unwind: "$NombreArray"` Con esta stage se separan los elementos del Array "NombreArray". No olvidar que **el nombre debe ir entre comillas y precedido de \$**. Se pueden aplicar otras stages antes y después de la separación.

Ejemplos

Mostrar los estudiantes con su *_id* y el array *asignatura* descompuesto.

```
var j_proy          = {asignatura:1}
var stage_proy      = {$project:j_proy}
var stage_separar   = {$unwind:"$asignatura"}
db.estudiantes.aggregate([stage_proy,stage_separar])
```

```
pruebas> db.estudiantes.aggregate([stage_proy,stage_separar])
[
  { _id: 100, asignatura: { nombre: 'BD', nota: 6 } },
  { _id: 200, asignatura: { nombre: 'Prog', nota: 4 } },
  { _id: 300, asignatura: { nombre: 'BD', nota: 8 } },
  { _id: 400, asignatura: { nombre: 'BD', nota: 6 } },
  { _id: 400, asignatura: { nombre: 'Prog', nota: 8 } },
  { _id: 500, asignatura: { nombre: 'BD', nota: 2 } },
  { _id: 500, asignatura: { nombre: 'Prog', nota: 5 } }
]
```

Se pueden observar 3 cosas; los documentos en que *asignatura* no es un array no se modifican, mientras que los documentos donde lo es se clonan, cada uno con una entrada del array *asignatura* y los documentos que no tienen el campo "asignatura" **se pierden**. (Esto último se puede evitar pasando a la stage unwind un objeto con algunas opciones en vez del nombre del Array directamente). Más información [aquí](#).

Mostramos ahora correctamente el listado de notas de BD, sin otras notas.

```
var stage_separar    = {$unwind:"$asignatura"}
var j_filtro         = {"asignatura.nombre": "BD"}
var stage_filtro     = {$match:j_filtro}
var j_proy           = {nombre: 1,"asignatura.nota": 1, _id: 0}
var stage_proy       = {$project:j_proy}
db.estudiantes.aggregate([stage_separar,stage_filtro,stage_proy])
```

```
pruebas> db.estudiantes.aggregate([stage_separar,stage_filtro,stage_proy])
[
  { nombre: 'Sergio', asignatura: { nota: 6 } },
  { nombre: 'Eva', asignatura: { nota: 8 } },
  { nombre: 'Miranda', asignatura: { nota: 6 } },
  { nombre: 'Gabriel', asignatura: { nota: 2 } }
]
```

2.2.3. Enlazar colecciones. "JOIN"

Siempre que se tenga un campo que enlace 2 colecciones, como si fuese una clave ajena, se puede usar una fase **lookup** para incluir los elementos de una colección como documentos embebidos en la otra, funcionando de manera parecida al *JOIN* de SQL.

- **\$lookup:<param>** Esta stage necesita un documento con 4 parámetros para realizar la unión:
 - **from**: nombre de la colección a incluir como objeto embebido
 - **localField**: nombre del campo en la colección del *aggregate* que se corresponde al *foreignField*.
 - **foreignField**: nombre del campo en la colección del *from* que se corresponde al *localField*.
 - **as**: nombre que le queremos dar al campo en el que quedan embebidos los documentos.

Ejemplos

Creamos una nueva colección *profesores* que tenga un campo con la asignatura que imparten(*imparte*), esto puede funcionar como una "clave ajena" para unir con las asignaturas en la colección *estudiantes*.

```
db.createCollection("profesores")
var j_prof1 = {_id: 100, nombre:"Abelardo",
email:"abe1@gmail.com", imparte:"BD"}
var j_prof2 = {_id: 200, nombre:"Pau", email:"pau1@gmail.com",
```

```
imparte:"BD"}
var j_prof3 = {_id: 300, nombre:"Joan", email:"joan1@gmail.com",
imparte:"Prog"}
var j_prof4 = {_id: 400, nombre:"Admin", imparte:["Prog","BD"]}
db.profesores.insertMany([j_prof1, j_prof2, j_prof3, j_prof4])
```

Para mostrar el listado de *estudiantes* con las asignaturas que tienen y los nombres de los profesores de las mismas:

```
var st_unwin = {$unwind:"$asignatura"}
var j_join    = {from:"profesores", localField:"asignatura.nombre",
                 foreignField:"imparte", as:"asignatura.profesores"}
var st_join   = {$lookup:j_join}
var j_proy    = {nombre: 1, "asignatura.nombre": 1,
                 "asignatura.profesores.nombre": 1, _id: 0}
var st_proy   = {$project:j_proy}
db.estudiantes.aggregate([st_unwin,st_join,st_proy])
```

```
pruebas> db.estudiantes.aggregate([st_unwind,st_join,st_proy])
[
  {
    nombre: 'Sergio',
    asignatura: {
      nombre: 'BD',
      profesores: [
        { nombre: 'Abelardo' },
        { nombre: 'Pau' },
        { nombre: 'Admin' }
      ]
    }
  },
  {
    nombre: 'Pablo',
    asignatura: {
      nombre: 'Prog',
      profesores: [ { nombre: 'Joan' }, { nombre: 'Admin' } ]
    }
  },
  {
    nombre: 'Eva',
    asignatura: {
      nombre: 'BD',
      profesores: [
        { nombre: 'Abelardo' },
        { nombre: 'Pau' },
        { nombre: 'Admin' }
      ]
    }
  },
  {
    nombre: 'Miranda',
    asignatura: {
      nombre: 'BD',
      profesores: [
        { nombre: 'Abelardo' },
        { nombre: 'Pau' },
        { nombre: 'Admin' }
      ]
    }
  }
]

[
  {
    nombre: 'Miranda',
    asignatura: {
      nombre: 'Prog',
      profesores: [ { nombre: 'Joan' }, { nombre: 'Admin' } ]
    }
  },
  {
    nombre: 'Gabriel',
    asignatura: {
      nombre: 'BD',
      profesores: [
        { nombre: 'Abelardo' },
        { nombre: 'Pau' },
        { nombre: 'Admin' }
      ]
    }
  },
  {
    nombre: 'Gabriel',
    asignatura: {
      nombre: 'Prog',
      profesores: [ { nombre: 'Joan' }, { nombre: 'Admin' } ]
    }
  }
]
```

Como se puede observar, *lookup* es capaz incluso de hacer un join correctamente cuando los campos afectados están dentro de un documento e incluso formando parte de un array, añadiendo incluso los *profesores* que tienen varios módulos en

el módulo correcto. Esto tiene una limitación; cuando el *localField* forma parte de un array, como es el caso, se debe deconstruir o funciona como un conjunto.

Si se quita el *st_unwind* de la orden se puede observar que al añadir el campo *"asignatura.profesores"* no lo hace a cada elemento del array sino que lo fusiona en un único documento, el join seguiría siendo correcto en cierta manera pero ya no tenemos las asignaturas separadas, con lo que en los documentos que tienen *asignatura* como array es incapaz de mostrar el *nombre* de las asignaturas, y solo muestra el campo *profesores* con los profesores de todas las asignaturas que el alumno tiene.

⚠ Para unir campos que están dentro de un array en el origen del aggregate conviene aplicar antes un *\$unwind*.

2.2.4. Agrupamiento y funciones agregadas

- ***\$group:<agrupamiento>*** Esta fase se usa para realizar agrupamientos de forma parecida al *GROUP BY* de SQL y usar funciones agregadas. A esta fase se le pasa un documento con la clave de agrupación (*GROUP BY*) y los campos con las funciones agregadas, llamadas *acumuladores* en MongoDB.
 - ***_id:clave***: La fase agrupa los documentos en grupos según esta clave, de modo que la salida es un documento por cada valor único de esta clave, que puede ser un campo, varios o una expresión. Deben ser precedidos por *\$*. El *_id* es necesario, así que si no se desea agrupar por ninguna categoría se puede dejar en blanco o null. Ejemplos ***{_id:\$edad,...}***, ***{_id:"",...}***, ***{_id:null,...}***
 - ***nombre:{acumulador:exp}***: nombre que damos al campo, acumulador a aplicar y campo/expresión donde aplicar, con *\$*. Por ejemplo ***{_id:\$edad, Media:{ \$avg:"\$Nota"}}*** calcula la media de los valores del campo *Nota* y los agrupa según la *edad*.
La lista de acumuladores es algo más completa que la de funciones agregadas, pero aquí nos limitaremos a las típicas: *\$sum*, *\$avg*, *\$min*, *\$max*, *\$count*, esta última sin valor, con un documento vacío (*\$count:{}*).
- Existe otra opción que permite calcular sencillamente **funciones agregadas en elementos que están dentro de un array en un mismo documento**. Por ejemplo las notas de nuestros estudiantes con un array de asignaturas. Para ello se puede aplicar el acumulador como un campo de la fase *\$project*, sin necesidad de *\$group*. Esto se puede hacer incluso en la proyección del find, aunque no se recomienda.
Esta opción simplemente calcula la agregada, no agrupa realmente nada, con lo que los elementos del array siguen mostrándose todos, por ejemplo, si se aplica un *\$max* a *"asignatura.nota"* y se muestra también *"asignatura.nombre"* no

se muestra sólo el módulo con más nota, sino todos y el campo que hayamos definido con la nota máxima.

Ejemplos

Listado de estudiantes con su nota media (agrupación por array, sin *\$group*)

```
var j_filtro      = {"asignatura": {$exists:true}}
var stage_filtro  = {$match:j_filtro}
var j_proy        = {nombre: 1, media:{$avg:"$asignatura.nota"}}
var stage_proy    = {$project:j_proy}
db.estudiantes.aggregate([stage_filtro,stage_proy])
```

```
pruebas> db.estudiantes.aggregate([stage_filtro,stage_proy])
[
  { _id: 100, nombre: 'Sergio', media: 6 },
  { _id: 200, nombre: 'Pablo', media: 4 },
  { _id: 300, nombre: 'Eva', media: 8 },
  { _id: 400, nombre: 'Miranda', media: 7 },
  { _id: 500, nombre: 'Gabriel', media: 3.5 }
]
```

Si añadimos asignatura a la proyección, se puede observar que los datos siguen accesibles, solo se ha calculado la media de los datos de las notas de dentro pero el Array sigue sin agrupar.

```
var j_filtro      = {"asignatura": {$exists:true}}
var stage_filtro  = {$match:j_filtro}
var j_proy        = {nombre: 1, asignatura:1,
                    media:{$avg:"$asignatura.nota"}}
var stage_proy    = {$project:j_proy}
db.estudiantes.aggregate([stage_filtro,stage_proy])
```

```
pruebas> db.estudiantes.aggregate([stage_filtro,stage_proy])
[
  {
    _id: 100,
    nombre: 'Sergio',
    asignatura: { nombre: 'BD', nota: 6 },
    media: 6
  },
  {
    _id: 200,
    nombre: 'Pablo',
    asignatura: { nombre: 'Prog', nota: 4 },
    media: 4
  },
  {
    _id: 300,
    nombre: 'Eva',
    asignatura: { nombre: 'BD', nota: 8 },
    media: 8
  },
  {
    _id: 400,
    nombre: 'Miranda',
    asignatura: [ { nombre: 'BD', nota: 6 }, { nombre: 'Prog', nota: 8 } ],
    media: 7
  },
  {
    _id: 500,
    nombre: 'Gabriel',
    asignatura: [ { nombre: 'BD', nota: 2 }, { nombre: 'Prog', nota: 5 } ],
    media: 3.5
  }
]
```

Mostrar cuantos estudiantes hay con cada edad.

```
var j_cuenta      = {"_id":"$edad",Cantidad:{$count:{}}}
var stage_group   = {$group:j_cuenta}
db.estudiantes.aggregate([stage_group])
```

```
pruebas> db.estudiantes.aggregate([stage_group])
[
  { _id: 13, Cantidad: 1 },
  { _id: 12, Cantidad: 1 },
  { _id: 19, Cantidad: 2 },
  { _id: 49, Cantidad: 2 },
  { _id: 39, Cantidad: 1 },
  { _id: 29, Cantidad: 1 },
  { _id: null, Cantidad: 2 }
]
```

Se puede observar que el campo por el que se agrupa queda con nombre `_id`, lo que puede resultar confuso. Esto se puede arreglar en la proyección, modificando el nombre de ese campo; tan sencillo como usar el nombre deseado y como valor el campo original precedido de `$` entre comillas (`grupo_edad:"$_id"`), ese es el modo de aplicar alias en MongoDB.

```
var j_cuenta      = {"_id":"$edad",Cantidad:{$count:{}}}
var stage_group   = {$group:j_cuenta}
var j_proy        = {grupo_edad:"$_id",_id:0,Cantidad:1}
var stage_proy    = {$project:j_proy}
db.estudiantes.aggregate([stage_group,stage_proy])
```

Mostrar el número total de alumnos (sin agrupar, usando `$count`).

```
var j_cuenta      = {"_id":"",Cantidad:{$count:{}}}
var stage_group   = {$group:j_cuenta}
db.estudiantes.aggregate([stage_group])
```

```
pruebas> db.estudiantes.aggregate([stage_group])
[ { _id: '', Cantidad: 10 } ]
```

Calcular la nota máxima, mínima y media de cada asignatura. En este caso se debe descomponer primero el Array de notas o los grupos de asignaturas no serían simples. Habría un grupo para quien tenga solo "BD" otro para quien tenga "Prog" y otro para quien tenga "BD"+"Prog", evitando que se puedan calcular los datos.

```
var st_unwin = {$unwind:"$asignatura"}
var j_cuenta = {"_id":"$asignatura.nombre",
               Maxima:{$max:"$asignatura.nota"},
               Minima:{$min:"$asignatura.nota"},
               Media:{$avg:"$asignatura.nota"}}
var st_group = {$group:j_cuenta}
db.estudiantes.aggregate([st_unwin,st_group])
```

```
pruebas> db.estudiantes.aggregate([st_unwin,st_group])
[
  { _id: 'BD', Maxima: 8, Minima: 2, Media: 5.5 },
  { _id: 'Prog', Maxima: 8, Minima: 4, Media: 5.666666666666667 }
]
```


3. Instalación y uso de entornos gráficos (IDE)

3.1. MongoDB Compass

En el siguiente vídeo [#2 del Curso de MongoDB del canal RedesPlus](#) verás cómo instalar MongoDB Compass en Windows y cómo conectar con un cluster de Bases de datos en Atlas. Además, verás un poco por encima la interfaz gráfica de Compass.

Verás los siguientes pasos:

1. Instalar Compass
2. Conectar con Atlas
3. BD, colecciones y documentos
4. Explicar las vistas en Compass

3.2. Robo 3T y Studio 3T

En el siguiente vídeo [#9 del Curso de MongoDB del canal RedesPlus](#) verás cómo instalar y configurar dos herramientas muy útiles para utilizar con MongoDB: ROBO 3T y STUDIO 3T. Además, verás cómo conectarte con el Cluster de MongoDB University y con un Cluster propio que puedes crear con Mongo Atlas.

Verás los siguientes pasos:

1. Descargar e Instalar ROBO 3T
2. Conectar con los Cluster de Atlas desde ROBO 3T
3. Diferencias entre ROBO 3T y STUDIO 3T
4. Descargar e Instalar STUDIO 3T
5. Conectar con los Cluster de Atlas desde STUDIO3T

4. Bibliografía

- MongoDB Manual. MongoDB Documentation.
<https://www.mongodb.com/docs/manual/>
- Proyectos ágiles. Triángulo de hierro. <https://proyectosagiles.org/triangulo-hierro/>
- MongoDB: qué es, cómo funciona y cuándo podemos usarlo (o no).
<https://www.genbeta.com/desarrollo/mongodb-que-es-como-functiona-y-cuando-podemos-usarlo-o-no>