CENTRE ESPECÍFIC
D´EDUCACIÓ A DISTÀNCIA DE
LA COMUNITAT VALENCIANA

UNIDAD 8

BASES DE DATOS NOSQL: MONGODB

BASES DE DATOS 22/23
CFGS DAW

PARTE 2 DE 2. MONGODB: DML, CONSULTAS AVANZADAS E IDES

Revisado por:
Abelardo Martínez y Pau Miñana
Autor:
Sergio Badal

ÍNDICE DE CONTENIDO

UD8.2. MONGODB: DML, CONSULTAS AVANZADAS E IDES

1. MODIFICACIÓN DE DATOS (DML)

1.1 POBLADO DE LA BASE DE DATOS

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

CONSEJO

Para ejecutar el código en tu consola, copia, pega antes el código en un editor de textos de Linux (Kwrite, Pluma, Gedit) o Windows (Bloc de notas) y quítale las tabulaciones de la izquierda antes de pegarlo en la consola. Funciona perfectamente con las tabulaciones, pero la salida queda más elegante sin ellas.

Creamos una colección con datos de prueba para los ejemplos

```
// 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 CONSIDERACIONES PREVIAS

IMPORTANTE

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**. Estos comandos están desaconsejados, puesto que son obsoletos (deprecated).*

1.3 INSERCIÓN DE DOCUMENTOS

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.

La orden **insert** está desaconsejada (*deprecated*).

Insertar (insertOne / insertMany)

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

Ejemplo1:

```
var j_doc1 = {nombre:"Juan"}
var j_doc2 =
   {
      nombre:"Eva",
      apellidos:"García"
   }
db.estudiantes.insertOne(j_doc1)
db.estudiantes.insertOne(j_doc2)
```

Ejemplo2:

```
var j_doc1 = {nombre:"Juan"}
var j_doc2 =
   {
      nombre:"Eva",
      apellidos:"García"
   }
db.estudiantes.insertMany([j_doc1, j_doc2])
```

Ejemplo3:

```
db.estudiantes.insertOne({nombre:"juan"})
db.estudiantes.insertOne({nombre:"Eva", apellidos:"García"})
```

Ejemplo4:

```
db.estudiantes.insertMany([{nombre:"Juan"}, {nombre:"Eva" , apellidos:"García"}])
```

Variantes NO válidas:

```
db.<col>.insertOne()
```

```
db.<col>.insertMany()
```

En los ejemplos anteriores estamos insertando varios documentos sobre la misma colección llamada pruebas. Fíjate en que estamos creando variables Javascript para hacer más legible el código, pero realmente no son necesarias como puedes ver en el ejemplo3 y el ejemplo4.

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

1.4 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 (el primero que encuentre ordenado por _id) o todos los que cumplan la condición. La orden **update** está desaconsejada (*deprecated*).

Actualizar (updateOne / updateMany)

```
db.<coleccion>.update(<filtro>,<accion>,{opcion1:valor1,…})
db.<coleccion>.updateOne(<filtro>,<accion>, {opcion1:valor1, ...})
db.<coleccion>.updateMany(<filtro>,<accion>,{opcion1:valor1, ...})
```

OPERADORES para `<accion>` (CON SÍMBOLO DÓLAR):

```
$set:{doc} documento con los campos y valores a asignar $set:{campo1:"sergio", campo2: 5}
$unset:doc documento con los campos y valores a eliminar $set:{campo1:""} → no hace falta poner un valor, ya que se elimina el campo1
$inc:doc incrementa en x unidades el valor del campo indicado campo {campo1:5}
$rename:doc cambia un campo de nombre {campo1:"nuevonombre"}
```

OPCIONES de UPDATE (SIN SÍMBOLO DÓLAR):

```
upsert:true/false si ningún documento cumple el <filtro> se inserta el json del $set al final
```

Ejemplos:

a) Actualización de **todos** los documentos que cumplan un filtro determinado:

```
var j_filtro      = {nombre:"Pedro"}
var j_set         = {nota:5}
var j_accion      = {$set:j_set}
db.estudiantes.updateMany(j_filtro, j_accion)
```

b) Actualización del **primer documento** que cumpla un filtro determinado:

```
// Con updateOne se recomienda usar el _id en el filtro para asegurarnos que se actualiza el documento que queremos que se actualice
db.estudiantes.updateOne(j_filtro, j_accion)
```

c) Actualización de **todos** los documentos de una colección

```
// Para no usar filtro, tenemos que dejar expresamente en blanco el filtro, como hacíamos con el find cuando queríamos aplicar una proyección sin filtrar
db.estudiantes.updateMany({}, j_accion)
```

d) Actualización de **todos** documentos de la colección que cumplen el filtro y, si ninguno lo cumple, inserto un nuevo

```
//Si el <filtro> NO se cumple inserto un <json> nuevo con el o los campos del $set
var j_opciones    = {upsert:true}
db.estudiantes.updateMany(j_filtro, j_accion, j_opciones)
```

e) Borrado de **los valores de un campo**

```
var j_set         = {nota:""}
var j_accion      = {$set:j_set}
db.estudiantes.updateMany(j_filtro, j_accion)
```

f) Borrado de **un campo y sus valores**

```
var j_set         = {nota:""}
```

```
var j_accion      = {$unset:j_set}
db.estudiantes.updateMany(j_filtro, j_accion)
```

g) Borrado de dos **campos y sus valores**

```
var j_set         = {nota:"", nombre:""}
var j_accion      = {$unset:j_set}
db.<coleccion>.updateMany(j_filtro, j_accion)
```

Variantes válidas:

```
db.<col>.updateOne(j_filtro, j_accion)
db.<col>.updateOne({}, j_accion)
db.<col>.updateOne({}, j_accion, j_opciones)
```

```
db.<col>.updateMany(j_filtro, j_accion)
db.<col>.updateMany({}, j_accion)
db.<col>.updateMany({}, j_accion, j_opciones)
```

Variantes NO válidas:

~~db.<col>.updateOne()~~
~~db.<col>.updateOne(j_accion)~~
~~db.<col>.updateOne(j_opciones)~~
~~db.<col>.updateOne(j_accion, j_opciones)~~
~~db.<col>.updateOne(j_filtro, j_opciones)~~

~~db.<col>.updateMany()~~
~~db.<col>.updateMany(j_accion)~~
~~db.<col>.updateMany(j_opciones)~~
~~db.<col>.updateMany(j_accion, j_opciones)~~
~~db.<col>.updateMany(j_filtro, j_opciones)~~

Fíjate en que la acción es siempre obligatoria y el filtro, si no existe, hay que indicarlo como {}.

Ejemplos de actualizaciones

```
// limpiar pantalla (buena praxis)
cls
use pruebas
// a) Sube en un punto la nota de todos los documentos de los estudiantes mayores de 30 años.
// Muestra el antes y el después (solo apellidos y nota)
// Déjalo como estaba antes para poder ejecutar este script varias veces
var j_proyeccion = {apellidos: 1, nota:1, _id: 0}
var j_valor      = {$gt:30}
var j_filtro     = {edad: j_valor}
db.estudiantes.find(j_filtro,j_proyeccion)
var j_valor      = {nota: 1}
var j_accion     = {$inc: j_valor}
db.estudiantes.updateMany(j_filtro, j_accion)
db.estudiantes.find(j_filtro,j_proyeccion)
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 }
]
```

```
// b) Cambia el email del estudiante con email graciamerinda@gmail.com  por gracia@gmail.com
// Muestra el antes y el después (solo los emails)
// Déjalo como estaba antes para poder ejecutar este script varias veces
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' },
  {}
]
pruebas>       var j_filtro     = {email: "gracia@gmail.com"}

pruebas>       var j_valor      = {email: "graciamerinda@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: '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> 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 }
```

1.5 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.

La orden **delete** está obsoleta y desaconsejada (*deprecated*).

Borrado (deleteOne / deleteMany)

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

Ejemplos:

a) 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)
```

b) 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)
```

c) Borrado de **todos** los documentos de una colección (truncate)

```
db.estudiantes.deleteMany({})
```

Variantes válidas:

```
db.<col>.deleteOne(j_filtro)
db.<col>.deleteOne({})
db.<col>.deleteMany (j_filtro)
db.<col>.deleteMany({})
```

Variantes NO válidas:

~~db.<col>.deleteOne()~~ ~~db.<col>.deleteMany()~~

Fíjate en que es siempre obligatorio el filtro, si no existe, hay que indicarlo como {}.

Ejemplos de borrados

```
// a) 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)
```

```
// b) 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)
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)
[
  { apellidos: 'Muro Menéndez', nota: 2 },
  { apellidos: 'Gracia Merinda', nota: 4 }
]
pruebas>       db.estudiantes.deleteOne(j_filtro)
{ acknowledged: true, deletedCount: 1 }
pruebas>       db.estudiantes.find(j_filtro, j_proyeccion)
[ { apellidos: 'Gracia Merinda', nota: 4 } ]
```

```
// c) 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)
db.estudiantes.deleteOne({})
db.estudiantes.find({}, j_proyeccion)
```

```
pruebas>       var j_proyeccion = {apellidos: 1}

pruebas>       db.estudiantes.find({}, j_proyeccion)
[
  { _id: 100, apellidos: 'Gracia Merinda' },
  { _id: 200, apellidos: 'Concar López' },
  { _id: 300, apellidos: 'Gredos Martínez' },
  { _id: 600, apellidos: 'Gracia Merinda' },
  { _id: 700, apellidos: 'Concar López' },
  { _id: 1000, apellidos: 'Muro Menéndez' }
]
pruebas>       db.estudiantes.deleteOne({})
{ acknowledged: true, deletedCount: 1 }
pruebas>       db.estudiantes.find({}, j_proyeccion)
[
  { _id: 200, apellidos: 'Concar López' },
  { _id: 300, apellidos: 'Gredos Martínez' },
  { _id: 600, apellidos: 'Gracia Merinda' },
  { _id: 700, apellidos: 'Concar López' },
  { _id: 1000, apellidos: 'Muro Menéndez' }
]
```

2. CONSULTAS AVANZADAS

2.1 DOCUMENTO 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 tipo:

asignatura: {Nombre:"Bases de Datos", Nota:6}

Este campo podría ser también un Array de documentos con los datos de todas las asignaturas

```
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.

Para hacer referencia a los campos dentro de asignatura o asignaturas en un filtro/proyección se necesita usar el **nombre del documento padre 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.

Consultas en Documentos Embebidos

Actualizamos la base de datos de estudiantes, suponiendo que partimos de los datos originales del punto 1.1.
Ponemos algunas notas 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)
```

Lista de notas de BD

```
 var j_proyeccion = {nombre: 1,"asignatura.nota": 1, _id: 0}     // Este tipo de campos entre comillas
 var j_filtro      = {"asignatura.nombre": "BD"}                 // siempre
```

```
db.estudiantes.find(j_filtro,j_proyeccion)
```

```
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
    }
  ]
}
```

NOTA: *Se puede observar que el comando ejecuta la búsqueda tanto en los objetos embebidos simples como en los de dentro del Array. Esto tiene una ventaja a la hora de buscar, pues podemos filtrar de golpe con condiciones de campos esten en un array o no mientras el nombre sea similar, pero un inconveniente para mostrar los resultados, ya que lo que se selecciona es el documento entero que tiene un "asignatura.nombre":"BD" y muestra "asignatura.nota" para todo el documento, con lo que si está en un array con otras asignaturas no da la tota de BD sino todas las notas del Array. Veremos más adelante como tratar estos casos.*

Otra opción, aunque menos habitual, es **filtrar elementos usando directamente el indice 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.

Ten en cuenta que esta 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.

Tened en cuenta que las 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.NombreCol.aggregate([{docStage1},{docStage2},...])

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

2.2.1 STAGES CON EQUIVALENCIA EN EL FIND

- *$match:{doc}* Similar al filtro de la función find o al WHERE de SQL.
- *$project:{doc}* 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 en el $project un campo no se podrá usar posteriormente para otros propósitos.
- *$sort:{doc}* 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:"Nombre"* Similar a la operación count del find, pero esta vez no se pasa el número de documentos como un número sino como un documento con un campo "Nombre" con ese número como valor del campo.

Consultas básicas usando agregaciones

a) Filtrado: 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)

```
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
}
```

b) Proyección: Mostrar solo 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])
```

// Equivalente a db.estudiantes.find(j_filtro,j_proy)

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

NOTA: *Si aplicamos primero la proyección que el filtro db.estudiantes.aggregate([stage_proy,stage_filtro]) no se muestra nada, ya que cuando llega al filtro el único campo que se tiene es el nombre.*

c) Contar: Mostrar la cantidad de estudiantes que tienen 19 años)

```
var j_filtro          = {edad:19}
var stage_filtro      = {$match:j_filtro}
var stage_cuenta      = {$count:"Estudiantes_19"}
db.estudiantes.aggregate([stage_filtro,stage_cuenta])
```

// Similar a db.estudiantes.find(j_filtro).count(), pero ésta última devuelve directamente número, no documento

```
db.estudiantes.aggregate([stage_filtro,stage_cuenta])
{
  Estudiantes_19: 2
}
```

d) 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])
// Equivalente a db.estudiantes.find({},j_proy).sort(j_orden).limit(3)
```

```
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 SEPARAR LOS ELEMENTOS DEL ARRAY

La agregación permite aplicar una fase para separar los elementos de un array, de modo que en la salida se obtiene un a 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.

Deconstrucción de Arrays

a) Mostrar los estudiantes con el Array de asignaturas deconstruido.

```
var stage_separar     = {$unwind:"$asignatura"}
db.estudiantes.aggregate([stage_separar])
```

```
db.estudiantes.aggregate([stage_separar])
{
  _id: 100,
  nombre: 'Sergio',
  apellidos: 'Gracia Merinda',
  email: 'sergio@gmail.com',
  edad: 19,
  asignatura: {
    nombre: 'BD',
    nota: 6
  }
}
{
  _id: 200,
  nombre: 'Pablo',
  apellidos: 'Concar López',
  email: 'graciamerinda@gmail.com',
  asignatura: {
    nombre: 'Prog',
    nota: 4
  }
}
{
  _id: 300,
  nombre: 'Eva',
  apellidos: 'Gredos Martínez',
  email: 'gredoseva@gmail.com',
  edad: 13,
  asignatura: {
    nombre: 'BD',
    nota: 8
  }
}
```

```
{
  _id: 400,
  nombre: 'Miranda',
  apellidos: 'Aranda Bada',
  email: 'indo@arandabada.com',
  edad: 49,
  asignatura: {
    nombre: 'BD',
    nota: 6
  }
}
{
  _id: 400,
  nombre: 'Miranda',
  apellidos: 'Aranda Bada',
  email: 'indo@arandabada.com',
  edad: 49,
  asignatura: {
    nombre: 'Prog',
    nota: 8
  }
}
```

```
{
  _id: 500,
  nombre: 'Gabriel',
  apellidos: 'Muro Menéndez',
  edad: 12,
  asignatura: {
    nombre: 'BD',
    nota: 2
  }
}
{
  _id: 500,
  nombre: 'Gabriel',
  apellidos: 'Muro Menéndez',
  edad: 12,
  asignatura: {
    nombre: 'Prog',
    nota: 5
  }
}
```

NOTA: *Se pueden observar 3 cosas; los documentos en que "asignatura" no es un Array no se modifican, los documentos donde es un Array se clonan, cada uno con una entrada del Array de asignaturas 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.*

b) Mostraremos ahora correctamente el listado de notas de BD, solo con las notas de BD.

```
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])
```

```
> 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. “JOINS”.

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.

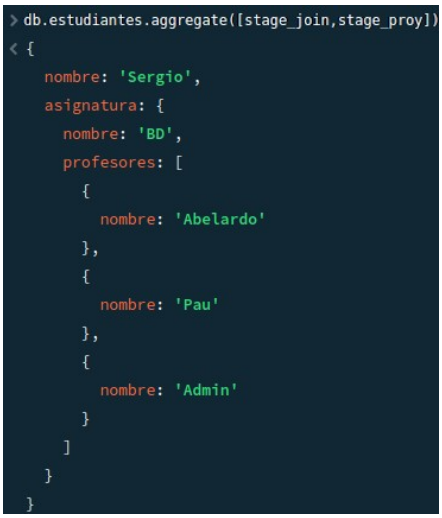
- *$lookup:{doc}* A esta stage hay que pasarle un documento con 4 parámetros para realizar la unión:
  - *from:* nombre de la colección a incluir como objeto embebido
  - *localField:* nombre del campo en la colección del aggregate que se corresponde al foreignField.
  - *foreignField:* nombre del campo en la colección del from que se corresponde al localField.
  - *as:* nombre que le queremos dar al campo en el que quedan incluidos los documentos de la colección del from

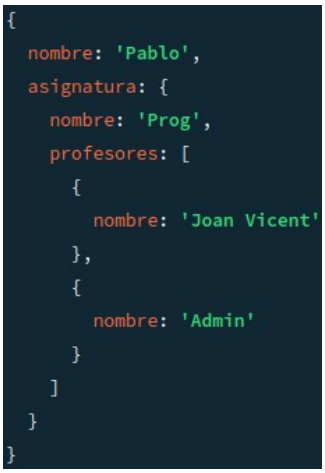
Unión de Colecciones

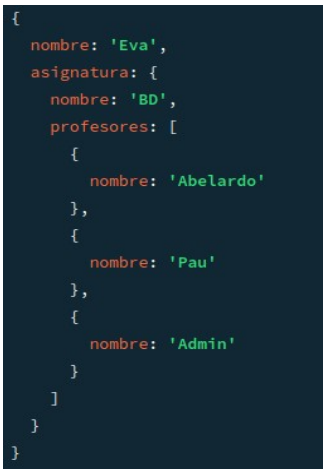
Creamos una nueva colección profesores que tenga un campo con la asignatura que imparten, esto puede funcionar como una “clave ajena” para unir con las asignaturas en la colección estudiantes.

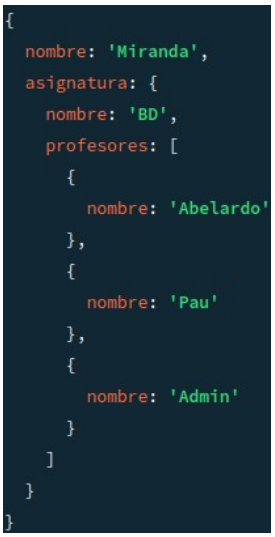
```
db.createCollection("profesores")
var j_p1 = {_id: 100, nombre:"Abelardo", email:"abe1@gmail.com", imparte:"BD"}
var j_p2 = {_id: 200, nombre:"Pau", email:"pau1@gmail.com", imparte:"BD"}
var j_p3 = {_id: 300, nombre:"Joan Vicent",email:"joanv1@gmail.com", imparte:"Prog"}
var j_p4 = {_id: 400, nombre:"Admin",email:null, imparte:["Prog","BD"]}
db.profesores.insertMany([j_p1, j_p2, j_p3, j_p4])

a) Listado de Alumnos con las asignaturas que tienen y los nombres de los profesores de las mismas
 var stage_separar    = {$unwind:"$asignatura"}
 var j_proy         = {nombre: 1,"asignatura.nombre": 1,"asignatura.profesores.nombre": 2, _id: 0}
 var stage_proy     = {$project:j_proy}
```

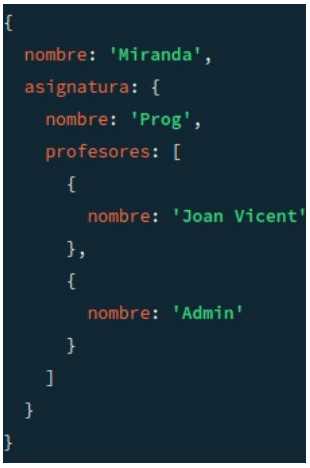
```
var j_join           = {from:"profesores", localField:"asignatura.nombre",
                        foreignField:"imparte", as:"asignatura.profesores"}
var stage_join       = {$lookup:j_join}
db.estudiantes.aggregate([stage_separar,stage_join,stage_proy])
```

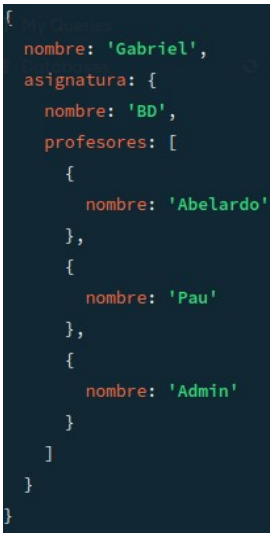


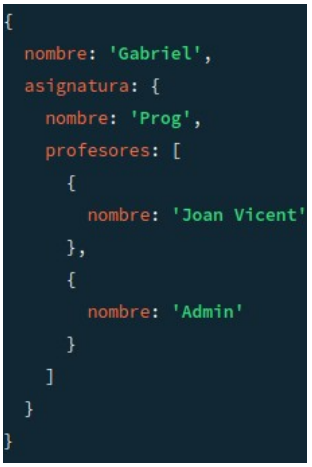












NOTA: *Como se puede observar lookup es capaz incluso de hacer un join correctamente cuando los campos afectados están dentro de un documento e incluso formando parte de un Array, añadiendo incluso los profesores que tienen varios módulos en el módulo correcto. Esto tiene una limitación, cuando el localField forma parte de un Array, como es es caso, se debe deconstruir o funciona como un conjunto.*

Si se quita el stage_separar de la orden se puede observar que para 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 los nombres de las asignaturas, y solo muestra el campo profesores con los profesores de todas las asignaturas que el alumno tiene.

En resumen, para unir campos que están dentro de un array en el origen del aggregate se debe aplicar antes un unwind.

2.2.4 AGRUPAMIENTO Y FUNCIONES AGREGADAS

- *$group:{doc}* Esta fase se usa para realizar agrupamientos de forma parecida al GROUP BY de SQL y usar funciones agregadas. A esta fase hay que pasarle un documento con la clave de agrupación (GROUP BY) y los campos con las funciones agregadas, llamadas acumuladores en MongoDB.
  - *_id:* Clave de agrupación. La fase agrupa los documentos en grupos según esta clave, como en un GROUP BY, de modo que la salida es un documento por cada valor único de esta clave, que puede ser un campo, varios o una expresión. Debe ser precedido por **$**. El id es necesario, así que si no se desea agrupar por ninguna categoría se puede dejar en blanco o null. Ejemplos *{_id:$edad,...}* , *{_id:"",...}* , *{_id:null,...}*
  - *Nombre:{acumulador:Campo/Expresión}:* nombre que damos al campo, acumulador a aplicar y campo/expresión donde aplicar, con **$**. Por ejemplo *{..., Media:{$avg:"$Nota"}}*

    La lista de acumuladores es algo más completa que la de funciones agregadas, pero aquí nos limitaremos a las típicas:

    - *$avg* media
    - *$sum* suma
    - *$count* cuenta elementos. Sin parámetros. $count:{}
    - *$max* máximo
    - *$min* mínimo

- Existe otra opción que permite calcular sencillamente funciones agregadas en elementos que están dentro de un Array dentro de 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 del *$project*, sin necesidad de $group. Esto se puede hacer incluso en la proyección del find, aunque no se recomienda. Además esta opción simplemente calcula la agregada sin agrupar realmente nada, con lo que los elementos del Array siguen pudiendo mostrarse (todos, si se aplica un $max a la nota y se muestra también el nombre, por ejemplo no se muestra sólo el módulo con más nota, sino todos).

Agrupamiento y Funciones de Agregación

```
a) Listado de Alumnos con su nota media (Arupación 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])
```

```
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
}
```

NOTA: *Añadiendo asignatura al $project se puede comprobar que los datos siguen accesibles, slo se ha calculado la media de los datos de las notas de dentro pero el Array sigue sin agrupar*

b) Número de Alumnos por edad

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

```
db.estudiantes.aggregate([stage_group])
< {
    _id: null,
    Cantidad: 2
  }
  {
    _id: 13,
    Cantidad: 1
  }
```

```
{
  _id: 12,
  Cantidad: 1
}
{
  _id: 49,
  Cantidad: 2
}
{
  _id: 19,
  Cantidad: 2
}
```

```
{
  _id: 39,
  Cantidad: 1
}
{
  _id: 29,
  Cantidad: 1
}
```

c) Número de Alumnos por edad de nuevo

Habrás podido notar que al usar el group se muestra el campo de agrupación como _id, resultando un poco confuso, esto se puede arreglar con un $project que muestre "$_id" con el nombre que queramos y oculte el _id original del group.

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

d) Número de Alumnos total (sin agrupar)

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

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

e) Nota media de cada asignatura

En este caso se debe deconstruir 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 stage_separar    = {$unwind:"$asignatura"}
var j_cuenta         = {"_id":"$asignatura.nombre",Maxima:{$max:"$asignatura.nota"},
                        Minima:{$min:"$asignatura.nota"},Media:{$avg:"$asignatura.nota"}}
var stage_group   = {$group:j_cuenta}
db.estudiantes.aggregate([stage_separar,stage_group])
```

```
db.estudiantes.aggregate([stage_separar,stage_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 (IDES)

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

https://www.youtube.com/watch?v=keTtL5fFO0Y&list=PLXXiznRYETLcJE_4U9qN2pysZOSYyL4Mh&index=2

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

https://www.youtube.com/watch?v=uXPv6xeuTKM&list=PLXXiznRYETLcJE_4U9qN2pysZOSYyL4Mh&index=9