Base de Datos no SQL

Descripción

MongoDB (que proviene de «humongous») es la base de datos NoSQL líder y permite a las empresas ser más ágiles y escalables. Organizaciones de todos los tamaños están usando MongoDB para crear nuevos tipos de aplicaciones, mejorar la experiencia del cliente, acelerar el tiempo de comercialización y reducir costes.

Es una base de datos ágil que permite a los esquemas cambiar rápidamente cuando las aplicaciones evolucionan.

MongoDB ha sido creado para brindar escalabilidad con un elevado rendimiento, tanto para lectura como para escritura, potenciando la computación en memoria (in-memory). La replicación nativa de MongoDB y la tolerancia a fallos automática ofrece fiabilidad a nivel empresarial y flexibilidad operativa.

Las suscripciones de MongoDB ofrecen un servicio de asistencia técnica profesional, licencias comerciales y acceso a características de software de MongoDB Enterprise. Las suscripciones no solo ayudan a los clientes a lograr una infraestructura de TI estable, escalable y segura, sino también a alcanzar sus objetivos empresariales más amplios, tales como reducir los costes, acelerar el tiempo de comercialización y disminuir los riesgos.

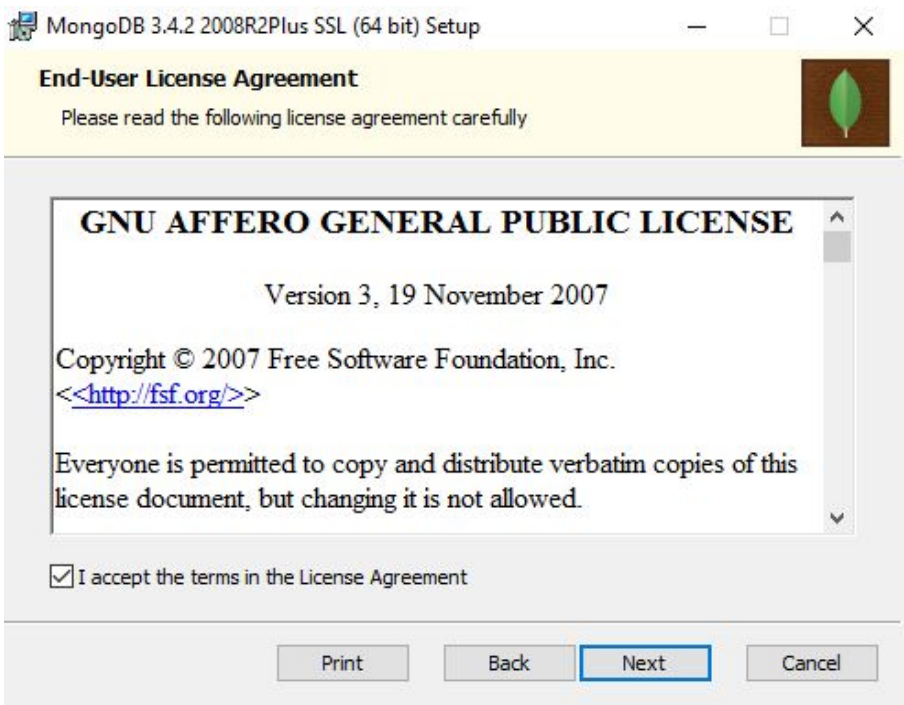
MongoDB Enterprise ofrece seguridad avanzada, monitorización on-premises, soporte SNMP, certificaciones de SO y mucho más. El servicio de gestión de MongoDB (MMS) ofrece funcionalidad de monitorización y respaldo en la nube o bien on-premises como parte de MongoDB Enterprise.

Instalación

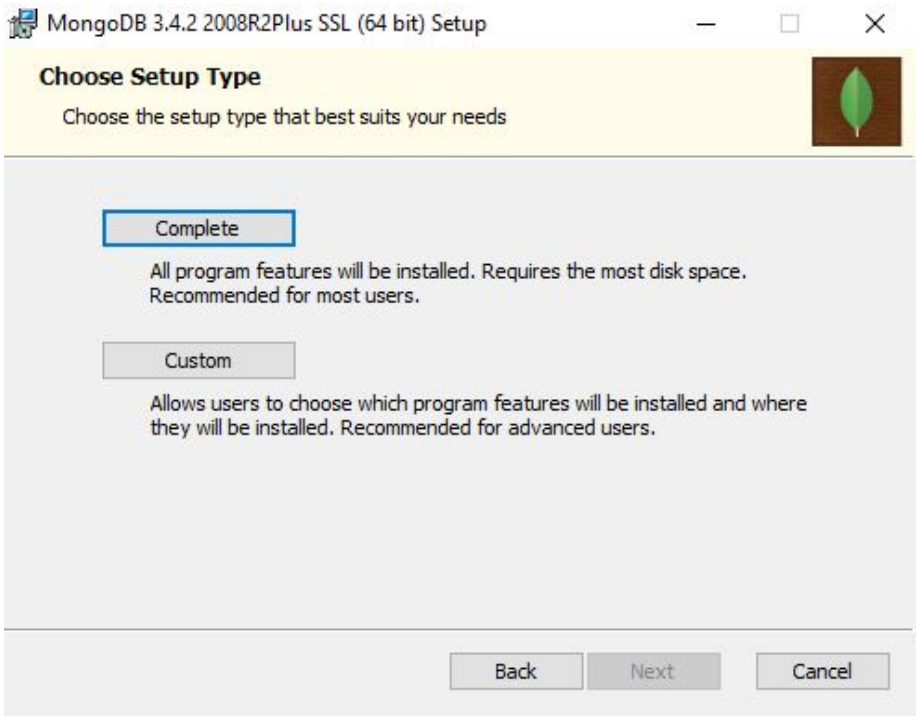
1. Nos descargamos la última versión de MongoDB desde su página oficial: [Descargar MongoDB]

2. Una vez descargado el archivo .msi, lo iniciamos y empezará la instalación:

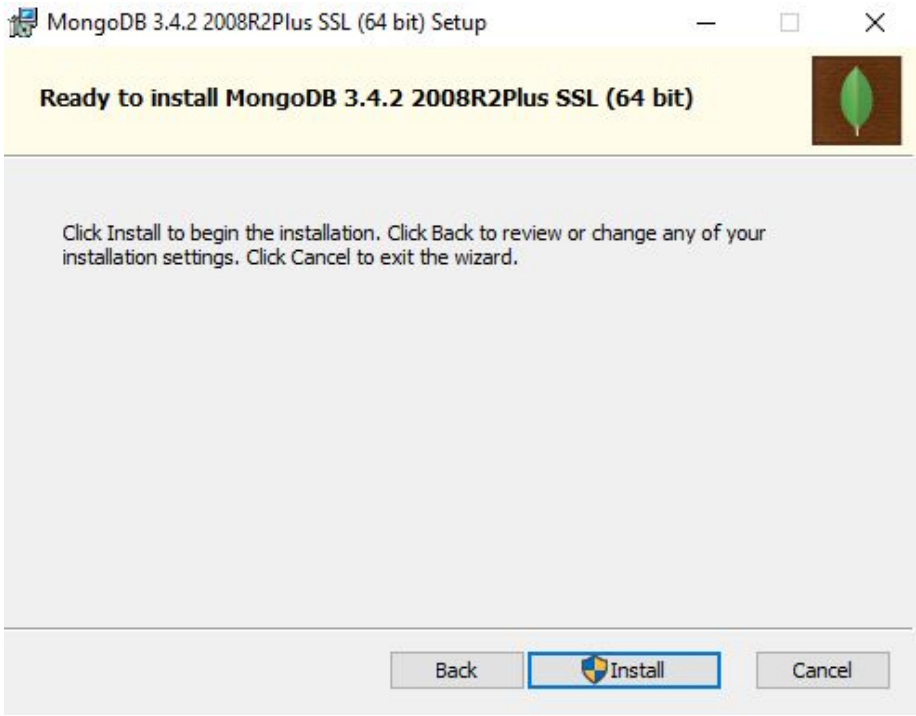
3. Aceptamos los términos que no vamos a leer:

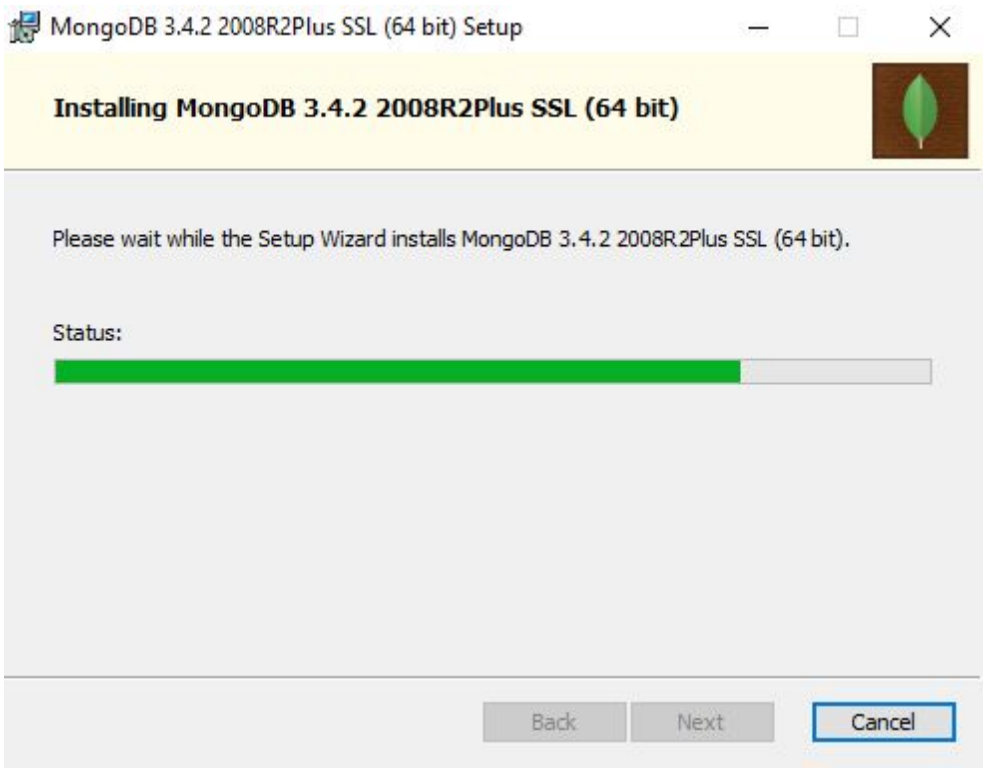


4. Elegimos la opción de instalación completa:

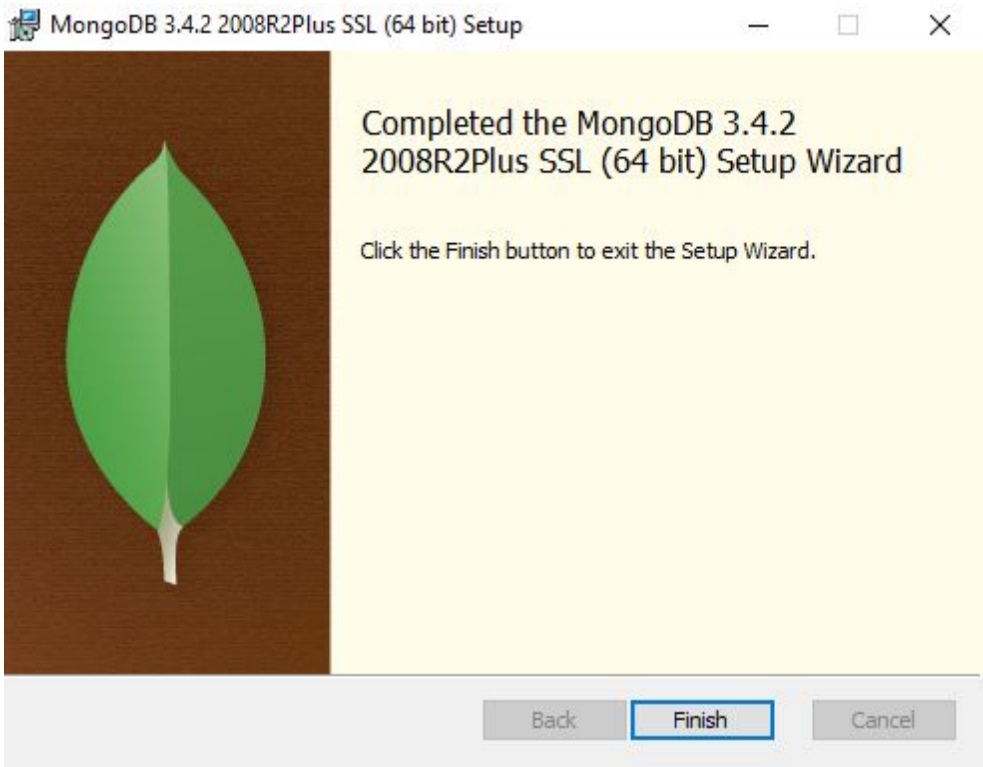


5. Una vez terminada la configuración, le damos a instalar:





6. Para finalizar la instalación pulsamos en *Finish*



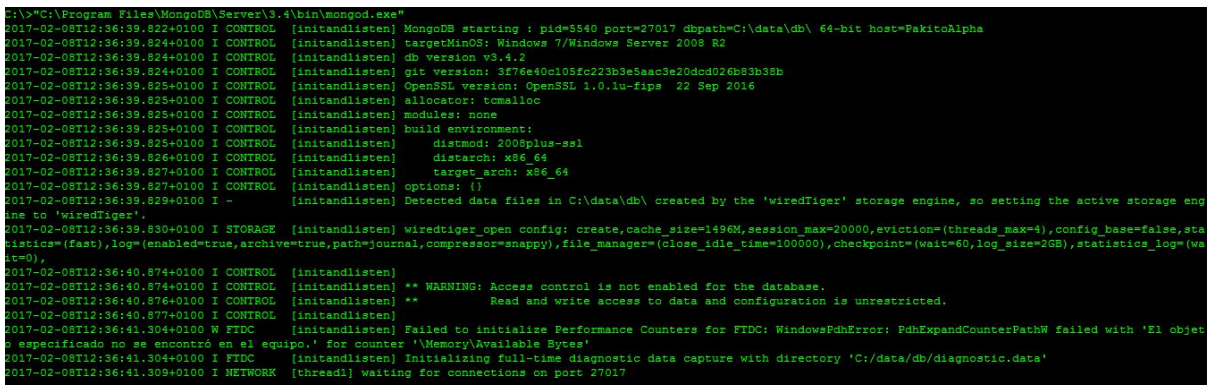
Iniciar MongoDB

1. Primero de todo tenemos que crear una carpeta en la raíz del disco duro en el que hemos instalado mongoDB (C:/ en nuestro caso). Podemos crearla mediante la interfaz de Windows o mediante comandos.Tenemos que crear una carpeta llamada ***data*** y dentro de esta otra carpeta llamada ***db.***

```
md \data\db
```

2. Ahora tenemos que iniciar MongoDB, ejecutando mongod.exe. Eso inicia el proceso inicial de la base de datos.

```
“C:\ProgramFiles\MongoDB\Server\3.4\bin\mongod.exe”
```



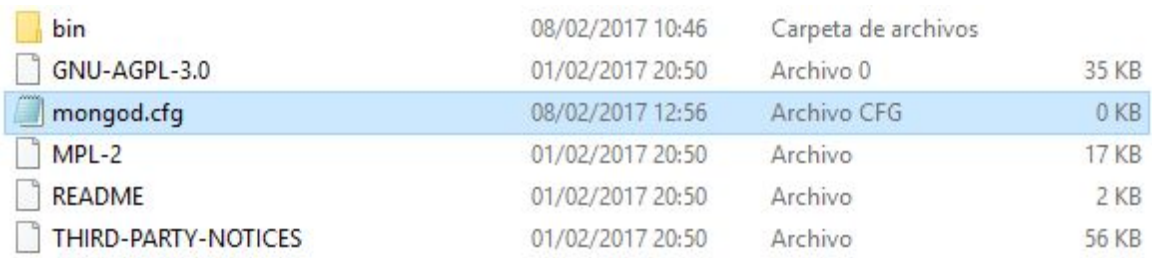
3. Una vez iniciado mongod.exe (sin cerrar la ventana) abrimos otra nueva. IMPORTANTE: hay que abrir el cmd con **permisos de administrador** y en ella iniciamos mongo.exe. “C:\Program Files\MongoDB\Server\3.4\bin\mongo.exe”

Configurar los servicios de Windows

1. Tenemos que crear una nueva carpeta dentro de la carpeta ***data*** creada anteriormente. Esta nueva carpeta debe llamarse ***log***.

mkdir c:\data\log

2. Ahora tenemos que implementar el archivo cfg con la configuración sobre el *Path*. Para ello nos vamos a *C:\Program Files\MongoDB\Server\3.4\ y c*amos un archivo llamado ***mongod.cfg***.

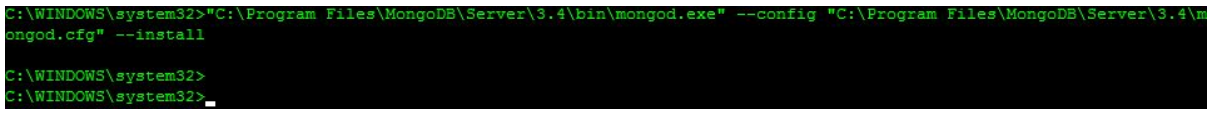


3. Abrimos el archivo con cualquier editor de texto y dentro de él escribimos los siguiente:

```
systemLog:
        destination: file
        path: c:\data\log\mongod.log
storage:
        dbPath: c:\data\db
```

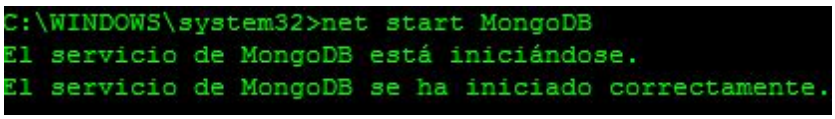
4. Iniciamos el cmd con **Permisos de Administrador** y copiamos el siguiente comando para instalar el servicio. Todo en la misma línea.

   "C:\Program Files\MongoDB\Server\3.4\bin\mongod.exe" --config "C:\Program Files\MongoDB\Server\3.4\mongod.cfg" --install



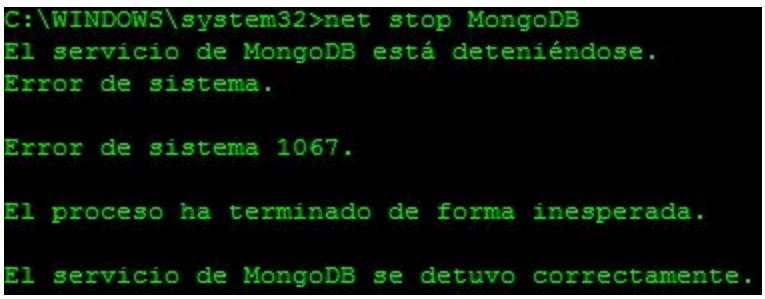
5. Si todo ha salido bien, se tendría que haber instalado los servicios, ahora solo tendríamos que iniciarlos.

   net start MongoDB



   Para detener el servicio se hace con el siguiente comando:

   net stop MongoDB



Para eliminar los servicios de MongoDB:

```
"C:\Program Files\MongoDB\Server\3.4\bin\mongod.exe" --remove
```

```
C:\WINDOWS\system32>"C:\Program Files\MongoDB\Server\3.4\bin\mongod.exe" --remove
2017-02-08T13:03:45.146+0100 I CONTROL  [main] Trying to remove Windows service 'MongoDB'
2017-02-08T13:03:45.148+0100 I CONTROL  [main] Service 'MongoDB' removed
```

Estructura de datos

Modelo Relacional	MongoDB
Base de datos	Base de datos
Tabla	Colección
Fila	Documento
Columna	Campo
Índice	Índice
Join	Documento embebido o referencia

MongoDB utiliza JSON (**J**ava**S**cript **O**bject **N**otation) para su estructura de datos. Puede contener tanto objetos como arrays, además de poder anidar unos objetos o arrays dentro de otros.

Un **objeto** comienza con **{** (llave de apertura) y termina con **}** (llave de cierre), dentro de este objeto están contenidos los datos en forma de clave valor separados por **,** (comas), y asignados con **:** (dos puntos):

```
{ “persona”: { “nombre” : “Brian”, “Oficio” : “Prog” } }
{ “zona”: { “codzona” : 10, “nombre” : “Madrid” } }
```

Un **array**, es decir, una colección de valores, comienza con **[** (corchete de apertura) y termina con **]** (corchete de cierre), y se separan por comas. no tienen por qué tener los mismos pares nombre/valor.

```
{ “persona”: [
{ “nombre” : “Paco”, “Oficio” : “Prog” , “Ciudad” : “Parla” } ,
{ “nombre” : “David” , “Oficio” : “Prog” }
 ] }
```

Los valores pueden contener Strings, numeros, booleanos, o null y pueden anidarse en un array.

```
{ “ventana : {
        “titulo” : “Gestión de Artículos” ,
        “alto” : 300 ,
        “ancho” : 500 ,
        “menu” : null ,
        “modal” : true ,
        “botones” : [ “ok” , “cancel” ] }
}
```

Operaciones Básicas

Todos los comandos para operar con esta base de datos se escriben en minúscula, los más comunes son:

show databases → Lista las bases de datos.
db → Muestra la base de datos actual.
show collections → Muestra las colecciones de la base de datos.
use *nombrebasededatos* → Usar una base de datos, si no existe, la creará en el momento que añadamos un objeto.

Los comentarios se añaden con // igual que en Java.

1. Creación de Registros

Para añadir datos a la base de datos utilizamos **.save** o **.insert** (ambos hacen lo mismo) según este formato:

```
db.nombre_coleccion.save(dato JSON) ;
db.nombre_coleccion.insert(dato JSON) ;
```

Donde db es un identificador de la base de datos actual y nombre colección es la colección donde se van a añadir los registros, si no existe se crea en ese momento.

Ejemplo: Creo la base de datos *mibasedatos*, y dentro de ella la colección *amigos* con dos amigos:

```
use mibasedatos;
Amigo1 = { nombre : "Ana" , curso : "1DAM" , nota : 7 } ;
Amigo2 = { nombre : "Marleni" , curso : "1DAM" , nota : 8 } ;
db.amigos.save(Amigo1) ;
db.amigos.insert(Amigo2) ;
```

Añado un amigo más, pero ahora sin crear los objetos.

```
db.amigos.save(
{ nombre : "Juanito" , curso : "2DAM" , nota : 6 }
) ;
```

Cada documento (Registro) tiene un identificador único (ID). Se asigna uno automáticamente al crear el documento usando un número hexadecimal que consta de 12 bytes. También se pueden crear de forma manual, el cómo es un misterio.

2. Consulta de Registros

Para consultar datos de una colección utilizamos la orden **.find()**.

```
db.nombre_coleccion.find( ) ;
```

Por ejemplo, si queremos ver la colección amigos, escribiremos:

```
db.amigos.find( ) ;
```

Se muestran los identificativos (_id) de cada objeto JSON junto con el resto de campos.

```
> db.amigos.find();
{ "_id" : ObjectId("58a3796bd61093800cdcfe9a"), "nombre" :
"Ana", "curso" : "1DAM", "nota" : 7 }
{ "_id" : ObjectId("58a3796dd61093800cdcfe9b"), "nombre" :
"Marleni", "curso" : "1DAM", "nota" : 8 }
{ "_id" : ObjectId("58a37988d61093800cdcfe9c"), "nombre" :
"Juanito", "curso" : "2DAM", "nota" : 6 }
> _
```

Si lo que queremos es que la salida salga en orden ascendente por uno de los campo, utilizamos el operador **.sort()**.
Por ejemplo, para obtener los datos de la colección ordenador por el nombre:

```
db.amigos..find().sort( { nombre : 1 } ) ;
```

El número que acompaña a la orden indica el tipo de ordenación, 1 ascendente y -1 descendente.

Si se desea hacer búsquedas de documentos que cumplan una o varias condiciones, utilizamos el siguiente formato:

```
db.nombre_coleccion.find( filtro , campos ) ;
```

En ***filtro*** indicamos la condición de búsqueda, añadiendo los pares clave:valor a buscar. Si omitimos este parámetro o pasamos un documento vacío ({ }) devuelve todos los documentos.

En ***campos*** se especifican los campos a devolver de los documentos que coinciden con el filtro de la consulta. Para devolver todos los campos de los documentos omitimos este parámetro. Si se desea devolver uno o más campos escribiremos **{ nombre_campo1 : 1 , nombre_campo2 : 1 … }** o **{ nombre_campo1 : 0 , nombre_campo2 : 0 … }** dependiendo de si queremos que nos muestre (1), o nos oculte los campos (0) (también se puede usar true y false).

Por ejemplo: para buscar el amigo con nombre Marleni:

```
db.amigos.find( { nombre : “Marleni” } ) ;
```

Si solo quiero saber su nota:

```
db.amigos.find( { nombre : “Marleni” } , { nota : 1 } ) ;
```

Si quiero el nombre y la nota de los alumnos de 1DAM:

```
db.amigos.find( { curso : "1DAM" } , { nombre : 1 , nota : 1 } );
```

Si queremos saber el número de registros que devuelve la consulta usamos la sentencia **count()**.

```
db.amigos.find( { curso : "1DAM" } ).count( ) ;
```

3. Selectores de Búsqueda de Comparación

$eq, igual a un valor. Esta orden obtiene los de nota = 6:

```
db.amigos.find( { nota : { $eq : 6 } } ) ;
```

$gt, mayor que, **$gte** mayor o igual que. Nota >= 6:

```
db.amigos.find( { nota : { $gte : 6 } } ) ;
```

$lt, menos que, **$lte** menor o igual que. Nota entre 7 y 9:

```
db.amigos.find( { nota : { $gte : 7 , $lte : 9 } } ) ;
```

$ne, distinto a un valor. Nota distinta de 7:

```
db.amigos.find( { nota : { $ne : 7 } } ) ;
```

$in, está entre una lista de valores, **$nin** no está entre una lista de valores. Notas que sean 5, 7, u 8:

```
db.amigos.find( { nota : { $in : [ 5 , 7 , 8 ] } } ) ;
```

4. Selectores de Búsqueda Lógicos

$or selecciona todos los registros que cumplan con cualquiera de las condiciones indicadas igual que la sentencia “OR” en SQL. Los amigos con nombre Ana o Marleni:

```
db.amigos.find( { $or : [ { nombre: “Ana” } ,
{ nombre: “Marleni” } ]
} ) ;
```

$and selecciona todos los registros que cumplan todas las condiciones indicadas igual que la sentencia “AND” and SQL
(El operador es implícito, o sea que si no lo ponemos seguirá cumpliendo esta función).
Los amigos del curso 2DAM y nota 6:

```
db.amigos.find( { $and : [ { curso : "2DAM" } , { nota : 6 } ] } ) ;
db.amigos.find( {curso : "2DAM" , nota : 6 } );
```

$not representa la negación. Amigos con nota no mayor de 7:

```
db.amigos.find( { nota : { $not { &gt : 7 } } } ) ;
```

$exists operador booleano, permite filtrar la búsqueda tomando en cuenta la existencia del campo de la expresión.
Los amigos que tengan nota:

```
db.amigos.find( { nota : { $exists : true } } ) ;
```

5. Actualización de Registros

Para actualizar datos utilizamos el comando .update() según este formato de uso:

```
db.nombre_coleccion.update(
        filtro_busqueda,
        nuevos_datos,
        { upsert: booleano, multi: booleano } ) ;
```

En **filtro_busqueda**, se indica la condición para localizar los registros o documentos a modificar.

En **nuevos_datos**, se especifican los datos nuevos que tendrá el registro buscado, es decir: el resultado final del documento es lo que se escriba en **nuevos_datos**.

Ahora cambio la nota de Marleni por 10, y solo mantengo el campo nombre y nota, el resto al no aparecer en **nuevos_datos**, se eliminarán:

```
db.amigos.update( { nombre : “Marleni” } ,
{ nombre : “Marleni” , nota : 10 } ) ;
```

También tenemos dos campos más, **upsert** y **multi**, que de manera predeterminada están en false.

upsert indica, que si el filtro de búsqueda no encuentra ningún resultado, entonces, debe ser insertado como nuevo registro.

multi indica, que en caso de que el filtro de búsqueda devuelve más de un resultado, el cambio se realizará en todos los resultados, si está en false solo modificará el primero que encuentre, es decir, el que tenga menor _id.

6. Operadores de Modificación

$set permite actualizar con nuevas propiedades a un documento.
Añado la edad 24 a Ana y 34 a Marleni:

```
db.amigos.update({ nombre : "Ana" } , { $set: { edad : 24 } } );
db.amigos.update({ nombre : "Marleni" },{ $set: { edad :34}});
```

$unset permite eliminar propiedades de un documento.
Borro la edad 32 de Marleni:

```
db.amigos.update({nombre:"Marleni"},{ $unset: { edad :34}});
```

$inc incrementa en una cantidad numérica algún campo.
Incremento la edad de Ana en 2 años:

```
db..amigos.update( { nombre : "Ana" } , { $inc: { edad : 2 } } )
;
```

$rename cambia el nombre de algún campo.
Cambiamos los campos nombre y edad de Ana a inglés:

```
db.amigos.update( { nombre : "Ana" } , { $rename:
{ edad : "age" , nombre : "name" } } ) ;
```

7. Operaciones de Borrado

Para borrar datos JSON usamos las órdenes **.remove** y **.drop.** Se puede eliminar los que cumplan una condición, o todos los documentos de la colección, o la colección completa.

Para borrar un documento que cumpla una condición utilizaremos la orden:

```
db.nombre_colección.remove( { nombre : valor } ) ;
```

Elimina a Marleni:

```
db.amigos.remove( { nombre : “Marleni” } ) ;
```

Elimina el elemento con nombre Juanito y nota 6:

```
db.amigos.remove( { nombre : “Juanito” , nota : 6 } ) ;
```

Elimina todos los elementos de una colección:

```
db.amigos.remove( { } ) ;
```

Elimina la colección:

```
db.amigos.drop( ) ;
```

Operaciones con Arrays

1. Inserciones

   Las inserciones de arrays es como hemos visto anteriormente, indicándolo entre corchetes “[...]”:

   ```
   db.libros.insert( { codigo : 1 , nombre : "Acceso" , pvp : 35 ,
   editorial : "Garceta" , temas : [ "Base de datos" ,  "hibernate"
   , "neodatis" ] } ) ;
   db.libros.insert( { codigo : 2 , nombre : "Entornos" , pvp : 27 ,
   editorial : "Garceta" , temas : [ "UML" ,  "Subversion" ,
   “ERMaster" ] } ) ;
   db.libros.insert( { codigo : 3 , nombre : "Programacion" ,
   pvp : 25 , editorial : "Garceta" ,  temas : [ "SOCKET" ,
   "Multihilo" ] } ) ;
   ```

2. Consultas

   Para consultar los elementos del array escribimos el array y el elemento a consultar:

   ```
   db.libros.find( { temas : "UML" } ) ;
   db.libros.find( { $or: [ { temas : “UML” } ,
   { temas: “Neodatis” } ] } ) ;
   db.libros.find( { editorial : “Garceta” , pvp : { $gt : 25 } ,
   $or: [ { temas : “UML” } , { temas : “Neodatis” } ] } ) ;
   ```

3. Operaciones de Modificación

$push Añade un elemento a un array, añade el tema “MongoDB al libro con código 1:

```
db.libros.update( { codigo : 1 } , { $push : { temas :
“MongoDB” } } ) ;
```

$addToSet Agrega elementos a un array solo si estos no existen, añade tema el “Datos” a todos los libros con temas:

```
db.libros.update( { temas : { $exist : true } } ,
{ $addToSet : { temas : “Datos” } } , { multi : true } ) ;
```

$each Se usa en conjunto con $push o $addToSet para indicar que se añaden varios elementos al array:

```
db.libros.update( { codigo : 1 } , { $push :
{ temas : { $each : [ “JSON” , “XML” ] } } } ) ;
db.libros.update( { codigo : 2 } , { $addToSet :
{ temas : { $each : [ “Eclipse” , “Developper” ] } } } ) ;
```

$pop Elimina el primer o último valor de un array, -1 borra el primero, 1, el último, borra el primer tema del libro con código 3:

```
db.libros.update( { codigo : 3 } , { $pop : { temas : -1 } } ) ;
```

$pull Elimina los valores de un array que cumpla con una condición, borra de todos los libros los elementos “Datos” y “JSON”:

```
db.libros.update( { } , { $pull : { temas :
{ $in : [ “Datos” , “JSON” ] } } } , { multi : true } ) ;
```