TEMA 5. BASES DE DATOS NOSQL.

Objetivos

- Bases de datos NoSQL
- Características de las Bases de Datos NoSQL
- Ventajas de las Bases de Datos NoSQL
- Diferencias con las Bases de Datos SQL
- Tipos de Bases de Datos NoSQL

Contenidos

1.- Introducción.

La mayoría de aplicaciones utilizan algún tipo de bases de datos para funcionar.

Hasta ahora estábamos acostumbrados a utilizar bases de datos SQL como son MySQL, Oracle o MS SQL, pero desde hace ya algún tiempo han aparecido otras que reciben el nombre de NoSQL (Not only SQL – No sólo SQL) y que han llegado con la intención de hacer frente a las bases relacionales utilizadas por la mayoría de los usuarios.

El término NoSQL aparece con la llegada de la web 2.0 ya que hasta ese momento sólo subían contenido a la red aquellas empresas que tenían un portal, pero con la llegada de aplicaciones como Facebook, Twitter o Youtube, cualquier usuario puede subir contenido, provocando así un crecimiento exponencial de los datos.

Aparecen problemas para la gestión de toda esa información almacenada en bases de datos relacionales. Intentan solucionarlos:

- Optaron por utilizar un mayor número de máquinas pero se dieron cuenta de que esto no solucionaba el problema, además de ser una solución muy cara.
- Otra solución era la creación de sistemas pensados para un uso específico que con el paso del tiempo han dado lugar a soluciones robustas, apareciendo así el movimiento NoSQL.

Por lo tanto hablar de bases de datos NoSQL es hablar de estructuras que nos permiten almacenar información en aquellas situaciones en las que las bases de datos relacionales generan ciertos problemas debido principalmente a problemas de escalabilidad y rendimiento de las bases de datos relacionales donde se dan cita miles de usuarios concurrentes y con millones de consultas diarias.

Las bases de datos NoSQL:

- Son sistemas de almacenamiento de información que no cumplen con el esquema entidad–relación.
- No utilizan una estructura de datos en forma de tabla donde se van almacenando los datos sino que para el almacenamiento hacen uso de otros formatos como clave–valor, mapeo de columnas o grafos.

2.- Ventajas de los Sistemas NoSQL.

El uso de sistemas NoSQL tienen las siguientes ventajas:

- Se ejecutan con pocos recursos: Estos sistemas, a diferencia de los sistemas basados en SQL, no requieren de apenas computación, por lo que se pueden montar en máquinas de un coste más reducido.
- Escalabilidad horizontal: Para mejorar el rendimiento de estos sistemas simplemente se consigue añadiendo más nodos, con la única operación de indicar al sistema cuáles son los nodos que están disponibles.
- Pueden manejar gran cantidad de datos: Esto es debido a que utiliza una estructura distribuida, en muchos casos mediante tablas Hash.
- No genera cuellos de botella: El principal problema de los sistemas SQL es que necesitan transcribir cada sentencia para poder ser ejecutada, y cada sentencia compleja requiere además de un nivel de ejecución aún más complejo, lo que constituye un punto de entrada en común, que ante muchas peticiones puede ralentizar el sistema.

3.- Principales diferencias con BD SQL.

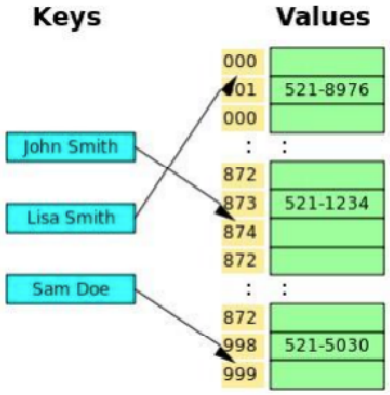
Los sistemas NoSQL se diferencian de las bases de datos SQL en que:

- No utilizan SQL como lenguaje de consultas: La mayoría de BD NoSQL evitan utilizar este tipo de lenguaje.
- No suelen permitir operaciones JOIN. Al disponer de un volumen de datos tan extremadamente grande suele resultar deseable evitar los JOIN. Esto se debe a que, cuando la operación no es la búsqueda de una clave, la sobrecarga puede llegar a ser muy costosa. Las soluciones más directas consisten en realizar el JOIN mediante software, en la capa de aplicación.

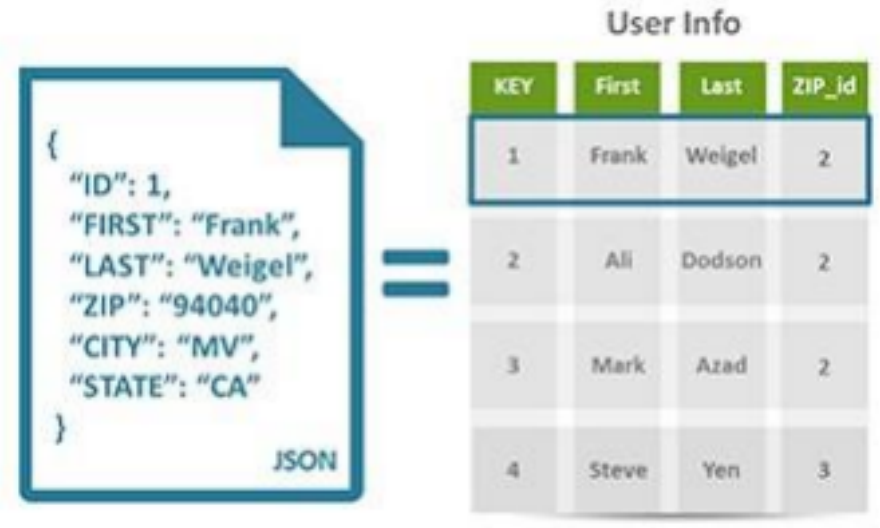
4.- Tipos de bases de Datos NoSQL.

Dependiendo de la forma en la que almacenen la información, existen varios tipos distintos de BD NoSQL:

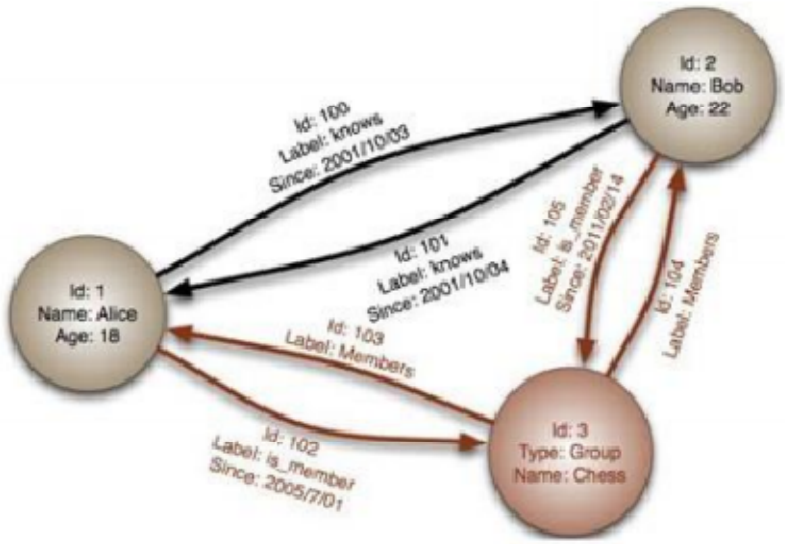
- **BD clave – valor**: Son las más usadas, además de ser la más sencilla en cuanto a funcionalidad. En este tipo de sistema, cada elemento está identificado por una llave única, lo que permite la recuperación de la información de forma muy rápida, información que habitualmente está almacenada como un objeto binario (BLOB). Se caracterizan por ser muy eficientes tanto para las lecturas como para las escrituras. Algunos ejemplos de este tipo son Cassandra, BigTable o HBase.



- **BD documentales**: Almacenan la información como un documento (JSON o XML) y se utiliza una clave única para cada registro. Permiten, además de realizar búsquedas por clave–valor, realizar consultas más avanzadas sobre el contenido del documento. Son las bases de datos NoSQL más versátiles. Se pueden utilizar en gran cantidad de proyectos, incluyendo muchos que tradicionalmente funcionarían sobre bases de datos relacionales. Algunos ejemplos de este tipo son MongoDB o CouchDB..



- **BD en grafo**: La información se representa como nodos de un grafo y sus relaciones con las aristas del mismo, pudiendo hacer uso de la teoría de grafos para recorrerla. Este tipo de BD ofrece una navegación más eficiente entre relaciones que en un modelo relacional. Algunos ejemplos de este tipo son Neo4j, InfoGrid o Virtuoso.



- **BD orientadas a objetos**: La información se representa mediante objetos, de la misma forma que son representados en los lenguajes de programación orientada a objetos (POO) como ocurre en JAVA, C# o Visual Basic .NET. Algunos ejemplos de este tipo de bases de datos son Zope, Gemstone o Db4o.

5.- Ejemplos de bases de Datos NoSQL.

Casandra.

- Se trata de una base de datos creada por Apache del tipo clave–valor.
- Dispone de un lenguaje propio para realizar consultas CQL (CassandraQuery Language).
- Cassandra es una aplicación Java por lo que puede correr en cualquier plataforma que cuente con la JVM.

Redis.

- Apoyado por VMWare.
- De tipo clave–valor.
- Se puede imaginar como un array gigante en memoria para almacenar datos, datos que pueden ser cadenas, hashes, conjuntos de datos o listas.
- Operaciones son atómicas y persistentes.
- No permite realizar consultas, sólo se puede insertar y obtener datos.
- Creado en ANSI C, por lo que es compatible y funciona en sistemas Unix, Linux y sus derivados, Solaris, OS/X sin embargo no existe soporte oficial para plataformas Windows.

CouchDB

- Creado por Apache.
- Escrito en lenguaje Erlang que funciona en la mayoría de sistemas POSIX, incluyendo GNU/LINUX y OSX, pero no tiene soporte para sistemas Windows.
- Restful HTTP API como interfaz y JavaScript como principal lenguaje de interacción.
- Almacenamiento en archivos JSON. Permite la creación de vistas, que son el mecanismo que permite la combinación de documentos para retornar valores de varios documentos, es decir, CouchDB permite la realización de las operaciones JOIN típicas de SQL.

MongoDB

- Orientada a documentos, de esquema libre, es decir, que cada entrada puede tener un esquema de datos diferente que nada tenga que ver con el resto de registros almacenados.
- Bastante rápido a la hora de ejecutar sus operaciones ya que está escrito en lenguaje C++.
- Utiliza un sistema propio de almacenamiento en documentos conocido con el nombre BSON, que es una evolución del conocido JSON pero con la peculiaridad de que puede almacenar datos binarios.
- Se ha convertido en una de las bases de datos NoSQL favoritas por los desarrolladores.

6.- Uso de bases de Datos NoSQL.

Algunas de las razones que nos pueden llevar a decantarnos por el uso de las bases de datos NoSQL en lugar de las clásicas SQL son:

- Cuando el volumen de los datos crece muy rápidamente en momentos puntuales, pudiendo llegar a superar el Terabyte de información.
- Cuando la escalabilidad de la solución relacional no es viable tanto a nivel de costes como a nivel técnico.
- Cuando tenemos elevados picos de uso del sistema por parte de los usuarios en múltiples ocasiones.
- Cuando el esquema de la base de datos no es homogéneo, es decir, cuando en cada inserción de datos la información que se almacena puede tener campos distintos.

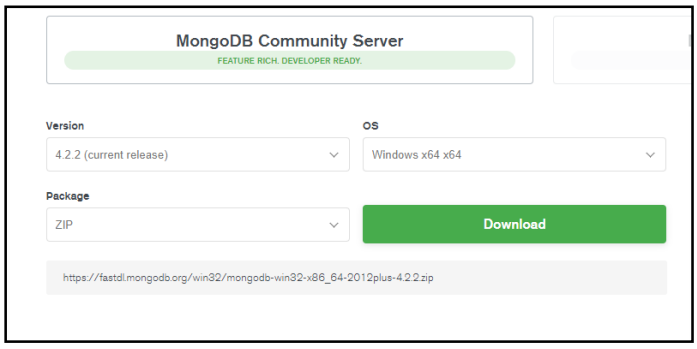
Son muchas las grandes empresas que hacen uso de este tipo de bases de datos no relacionales, como:

- Cassandra: Facebook, Twitter...
- HBase: Yahoo, Adobe...
- Redis: Flickr, Instagram, Github...
- Neo4j: Infojobs...
- MongoDB: FourSquare, SourceForge, CERN...

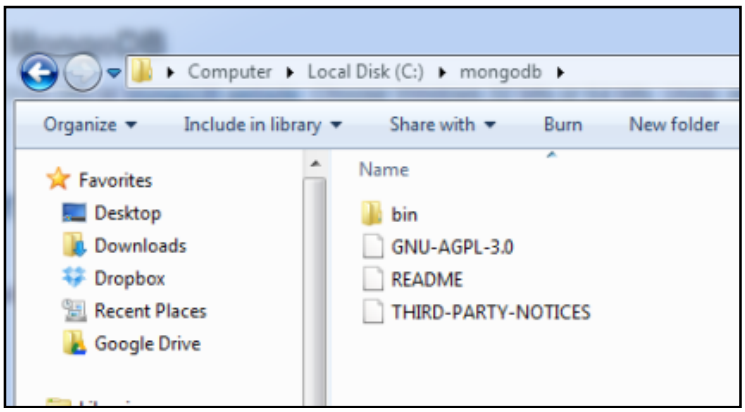
7.- MongoDB.

Instalación de MongoDB.

Enlace descargar MongoDB: https://www.mongodb.com/download-center/community.

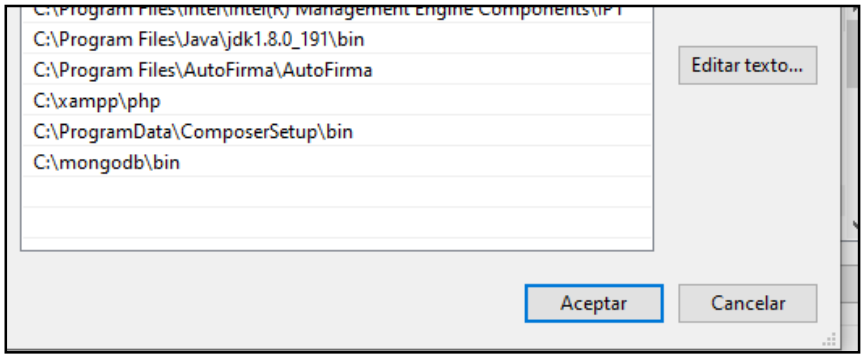


Una vez descargado, descomprime el archivo en, por ejemplo, c:\mongodb\ (crea antes la carpeta).



La carpeta bin contiene una serie de ejecutables, por lo que es conveniente añadir dicha carpeta en las variables de entorno de Windows. Para ello:

- Click con el boton derecho sobre Mi PC / Propiedades.
- Click en “Configuración avanzada del Sistema”
- En la pestaña “Avanzado”, click en “Variables de Entorno”.
- En “Variables del sistema”, en la variable “Path”, añade al final del valor “;c:\mongodb\bin\”.
- Pulsa OK a esta ventana y a las otras.

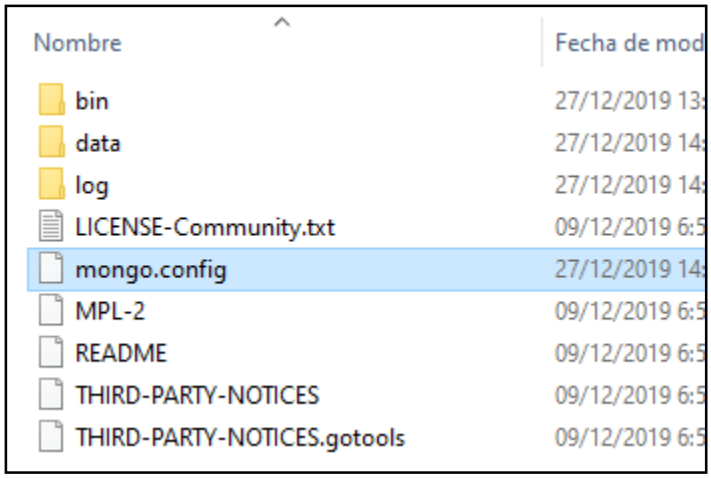


Crea un archivo con nombre "mongo.config" en c:\mongodb\ y en éste indica estos parámetros:

```
##store data here
dbpath=c:\mongodb\data

##all output go here
logpath=c:\mongodb\log\mongo.log
```

Crea las carpetas "data" y "log" en "c:\mongodb\". En ellas se añadirán los datos de las bases de datos y los archivos logs respectivamente.



Para ejecutar el servidor de MongoDB podemos añadir al servidor de MongoDB como Servicio de Windows para que se inicie automáticamente cuando se arranque el sistema operativo.
Para ello, ejecuta "cmd.exe" (el command de Windows) pero en este caso como administrador (para ello, click con el botón derecho sobre "cmd.exe" y pulsa sobre "Ejecutar como administrador") e introduce los siguientes comandos:

```
mongod --config c:\mongodb\mongo.config –install
net start MongoDB
```

El primer comando instala a MongoDB como un servicio de Windows, y el segundo comando inicia el servidor de MongoDB.

Para detener el servicio:

```
net stop MongoDB
```

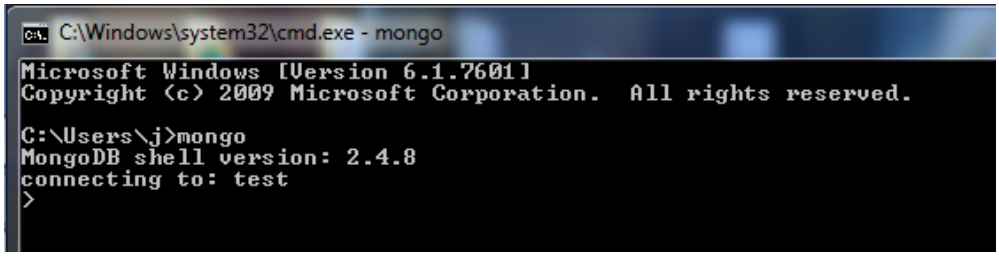
Para eliminar el servicio:

```
mongod –remove
```

Para probar que MongoDB se está ejecutando correctamente desde la consola de comandos escribimos:

```
mongo
```

Si todo salió con éxito, el resultado será algo similar al de la imagen:



8.- Uso de la Base de datos MongoDB.

MongoDB es un sistema de base de datos multiplataforma orientado a documentos. Algunas de las características son la velocidad y la sencillez de sus consultas.

Comparativa en la terminología utilizada en el Modelo Relacional y MongoDB:

Modelo Relacional	MongoDB
Base de datos	Base de datos
Tabla	Colección
Registro	Documento
Campo	Campo

Operaciones Básicas con Mongodb.

Las Operaciones básicas con la base de datos son:

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

- Listar las bases de datos: show databases
- Mostrar la base de datos actual: db
- Mostrar las colecciones de la base de datos actual: show collections
- Usar una base de datos (similar a MySQL): use nombrebasedatos, si no existe no importa, la creará en el momento que añadamos un objeto JSON, con las funciones .save o .insert.
- Si queremos saber el número de documentos dentro de las colecciones, utilizaremos la función count, escribiremos: db.nombrecolección.count ( ) .También se utilizan las funciones size() y length().
- Para añadir comentarios utilizamos los caracteres // de comentario de Java.

Crear Registros

Para añadir datos a la base de datos utilizaremos los comandos .save o .insert según este formato:

```
db.nombre_colección.save(dato JSON)
```

```
db.nombre_colección.insert(dato JSON)
```

Donde db es la base de datos actual, la que estemos usando (la abriremos con use) y nombre de 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',telefono:545656885,curso:'1DAM', nota:9};
Amigo2={nombre:'Marleni',telefono:3446500,curso:'1DAW',nota:8};
db.amigos.save(Amigo1);
db.amigos.save(Amigo2);
```

Añado un amigo más, pero ahora utilizo la orden insert:
db.amigos.insert({nombre: 'Juanito' , telefono:55667788, curso: '2DAM', nota: 6 });

Identificador de Objetos. El Objectid (campo _id).

Los identificadores de cada documento (registro) son únicos. Se asignan automáticamente al crear el documento.

Migrar datos desde MySQL a MongoDB.

En la pestaña SQL del phpmyAdmin escribir:

```
select *  into outfile 'peliculas.csv' FIELDS TERMINATED BY ',' LINES
TERMINATED BY '\n' from videoclub.peliculas;
```

En la carpeta c:\xampp\mysql\data\videoclub se crea el fichero peliculas.csv

Desde la consola CMD de windows, estando en la carpeta donde tenemos el fichero peliculas.csv escribir:

```
mongoimport -d videoclub -c peliculas --type csv --file peliculas.csv
--fields "codpelicula,titulo,tema,duracion,precio"
```

Configurar las líneas de salida de las consultas (Paginación).

Para configurar la salida de consultas a 100 líneas para evitar paginación escribir en la consola de mongoDB la sentancia:
DBQuery.shellBatchSize = 100;

Consultar registros

Para consultar datos de una colección utilizaremos la orden .find(), escribiremos:

```
db.nombrecolección.find()
```

Se muestran los identificativos (_id) de cada objeto JSON, únicos por colección, con el resto de campos.

Si se desea que la salida sea ascendente por uno de los campos, utilizamos el operador .sort(), por ejemplo, para obtener los datos de la colección ordenados por nombre escribirnos:

```
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 desean hacer búsquedas de documentos que cumplan una o varias condiciones, utilizamos el siguiente formato:

```
db.nombreColeccion.find(filtros, campos )
```

En filtro indicaremos la condición de búsqueda, podemos añadir los pares nombre: valor a buscar. Si omitimos este parámetro 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 desean devolver uno o más campos escribiremos {nombre_campo1: 1, nombre_campo2: 1,...}. Si no se desean que se seleccionen los campos escribimos {nombre_campo1: 0, nombre.campo2: 0,...}

Por ejemplo, para buscar el amigo con nombre Marleni lo escribirnos así:

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

Si solo deseo saber su teléfono escribo:

```
db.amigos.find({nombre:"Marleni"},{teléfono:1});
```

Si deseo buscar el nombre y la nota de los alumnos de 1DAM escribiremos:

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

Si queremos saber el número de registros que devuelve una consulta pondremos **db.nombreColeccion.find({filtros}).count()**, por ejemplo, para saber cuántos son del curso 1DAM escribiremos:

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

Selectores de Búsquedas de Comparación

Se pueden hacer consultas más complejas añadiendo selectores de búsquedas.

- **$eq**, igual a un valor. Esta orden obtiene los documentos con nota = 6:

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

- **$gt**, mayor que y $gte mayor o igual que. Esta orden obtiene los documentos con nota >= 6:

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

- **$lt** , menor que y **$lte**, menor o igual que. El ejemplo muestra los amigos de 1DAM con notas entre 7 y 9 incluidas, preguntamos por un intervalo >=7 y <=9:

```
db.amigos.find({curso : "1DAM", nota : { $gte : 7, $lte : 9} })
```

- **$ne**, distinto a un valor. El siguiente ejemplo obtiene los documentos con nota distinta de 7:

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

- **$in**, entre una lista de valores y **$nin**, no está entre la lista de valores. En el ejemplo se obtienen los documentos cuya nota sea uno de estos valores: 5,7 y 8:

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

- Contiene, comienza o termina por un texto. / son los indicadores de comienzo y fin de la expresión regular. Utilizaremos las siguientes expresiones regulares:
  /a/ para indicar que contiene la a.
  /^a/ para indicar que comienza por la a.
  /a$/ para indicar que termina en a.

  En el ejemplo se obtienen los documentos cuyo nombre contiene al menos una r :

```
db.amigos.find({ name: /r/ })
```

Selectores de Búsquedas Lógicos

- **$or**. La siguiente orden obtiene los documentos de los cursos 1 DAM , o los que tienen nota > de 7:

```
db.amigos.find({ $or: [{nota: {$gt: 7 }}, {curso : "1DAM"} ]})
```

  Esta consulta obtiene los amigos con nombre Ana o Marleni:

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

- **$and**. Este operador se maneja implícitamente, no es necesario especificarlo. Las siguientes órdenes hacen lo mismo, obtienen los amigos del Curso 2DAM y con nota 6:

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

```
db.amigos.find({curso: "2DAM", nota : 6})
```

  Esta otra consulta devuelve el documento con nombre Marieni y con teléfono 3446500:

```
db.amigos.find( {nombre : "Marleni", telefono : 3446500} )
```

```
db.amigos.find({$and:[{nombre:"Marleni"},{telefono:3446500}]})
```

- **$not**. Representa la negación, el ejemplo muestra los amigos con nota no mayor de 7.

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

- **$exists**, este operador booleano permite filtrar la búsqueda tomando en cuenta la existencia del campo de la expresión. Este ejemplo obtiene los registros que tengan nota.

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

Actualizar registros en MongoDB

Para actualizar datos utilizaremos el comando .update, según este formato de uso:

```
db.nombreColeccion.update(filtro_búsqueda, cambios_a_realizar,
{upsert: booleano, multi: booleano });
```

En filtro búsqueda, se indica la condición para localizar los registros o documentos a modificar. En cambios_a_realizar, se especifican los cambios que se desean hacer. Hay que tener cuidado al utiliza esta orden, pues si no se escriben todos los campos que tenía el documento, estos no los incluye en la modificación, entonces, los elimina. Por ejemplo:

```
db.amigos.update({nombre:"Ana"},{nombre: "Ana María" });
```

Esta orden cambia el documento con nombre Ana por nombre Ana María. El resto de campos como no aparecen en **cambios_a_realizar** los elimina.

Nos encontramos con dos tipos de cambios: cambiar el documento completo por otro que indiquemos, o modificar solo los campos especificados, para ello utilizamos los parámetro upsert y multi, ambos son opcionales y su valor por defecto es false:

- **upsert**: Si asignarnos true a este parámetro, se indica que si el filtro de búsqueda no encuentra ningún resultado, entonces, el cambio debe ser insertado como un nuevo registro.
- **multi**: En caso de que el filtro de búsqueda devuelva más de un resultado, si especificamos este parámetro a true, el cambio se realizará a todos los resultados, de lo contrario solo se cambiará al primero que encuentre, es decir, al que tenga menor identificativo de objeto "_id".

Ejemplo: Cambiar el teléfono de Pepita por 12345, y utilizamos upsert. Como Pepita no existe lo va a añadir por indicar **upsert: true**.

```
db.amigos.update({nombre:'Pepita'},{nombre:'Pepita',teléfono:12345},
{upsert: true});
```

Operadores de modificación

El comando .update cuenta con una serie de operadores para realizas actualizaciones más complejas.

- **$set**, permite actualizar con nuevas propiedades a un documento (o conjunto de documentos). Por ejemplo, añadir la edad 24 a Ana María y 34 a Marleni:

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

  Si el documento ya tiene ese campo, no lo añade, cambiaría el valor si es distinto.

- **$unset**, permite eliminar propiedades de un documento. Por ejemplo, borramos la edad 34 de Marleni:

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

- **$inc**, incrementa en una cantidad numérica especificada en el valor del campo a incrementar. Por ejemplo sumo 1 a la edad de Ana María:

```
db.amigos.update({nombre:"Ana María"},{ $inc: {edad:1} })
```

Por ejemplo, subimos la nota 1 punto a todos los alumnos de 1DAM:

```
db.amigos.update({curso:"1DAM"},{$inc:{nota:1}},{multi:true});
```

- **$mul**, multiplica una cantidad numérica especificada por el valor del campo a actualizar. Por ejemplo multiplicamos el sueldo un 10% de Ana María:

```
db.amigos.update({nombre:"Ana María"},{ $mul: {sueldo:1.10} })
```

Borrar registros

Para borrar registros utilizamos remove y .drop. Se pueden eliminar los documentos que cumplan una condición, todos los documentos de la colección o la colección completa.

Para borrar un documento que cumpla una condición utilizaremos la orden remove({ nombre: valor }). Por ejemplo, se borra a Marleni:

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

Si se desea borrar al elemento con nombre Ana y teléfono 545656885 escribimos:

```
db.amigos.remove({nombre : "Ana", teléfono : 545656885 });
```

Para borrar todos los elementos de la colección ponemos:

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

Para borrar la colección escribiremos:

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

Para borrar la base de datos escribiremos:

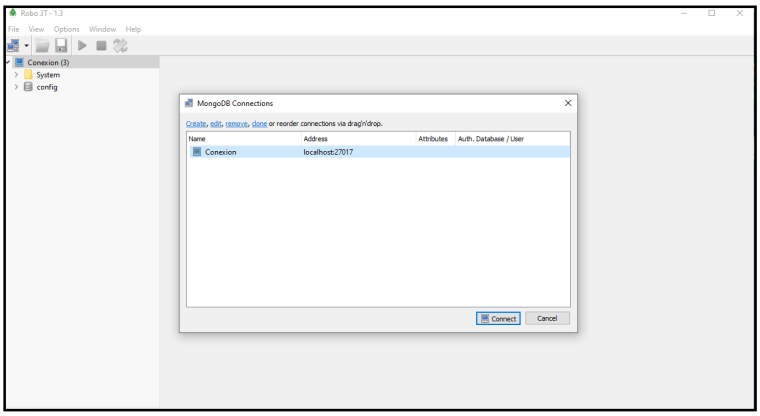
```
db.dropDatabase();
```

9.- Herramienta Robomongo.

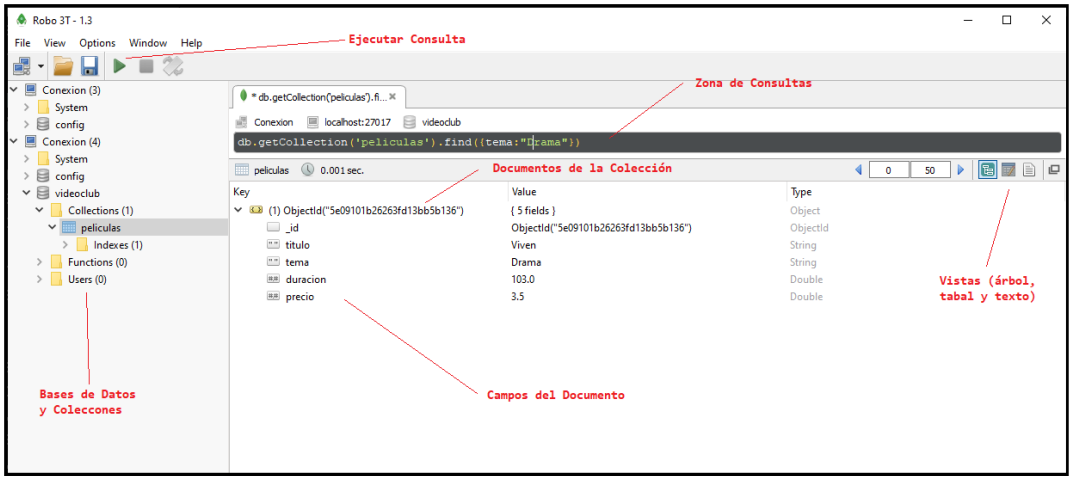
Robomongo es una herramienta multiplataforma con la que se pueden administrar de forma más visual las bases de datos MongoDB. Esta herramienta integra la Shell de mongodb en un entorno gráfico con todas sus funcionalidades, además se podrá trabajar con múltiples conexiones a las bases de datos, podremos navegar por las colecciones y a la hora de hacer las consultas contaremos con el resaltado de sintaxis y autocompletado del código, muy útil para detectar los errores.

Robomongo se descarga de la URL https://robomongo.org/. Existe la versión instalable la .exe, o la versión portable .zip.

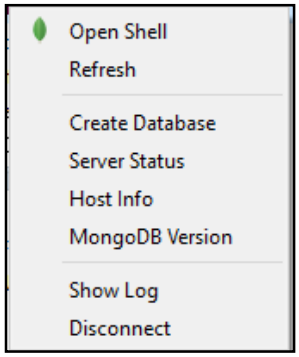
Para trabajar con la versión portable descomprimimos el zip en una carpeta y ejecutamos el archivo **Robo3t.exe**. Pide conexión con la base de datos, debemos tener la base de datos arrancada para que la detecte. El puerto de escucha de MongoDB es 27017.



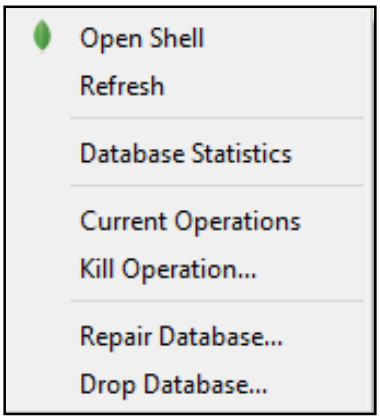
Desde la ventana de Robomongo podemos navegar por las bases de datos, las colecciones, y los objetos de las colecciones. Ad hacer clic en una colección se abrirá una pestaña donde se podrán ver los documentos de la colección, también podemos ver los campos del documento si hacemos doble clic en el objeto o si lo desplegamos. En la parte superior es donde escribiremos las consultas.



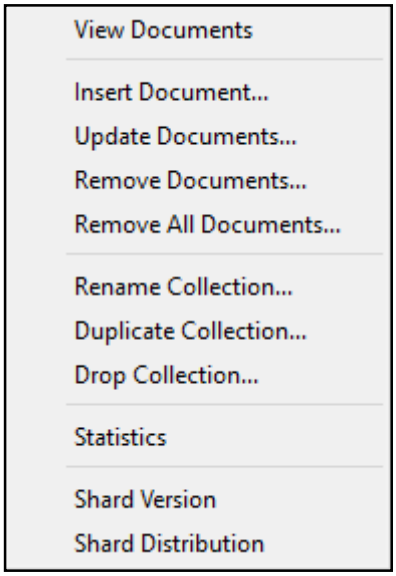
Desde el menú contextual de la conexiónala base de datos podremos crear nuevas bases de datos, abrir la Shell de MongoDB , que se abrirá en una pestaña para hacer las consultas, obtener información de la base de datos o desconectamos.



Desde el menú contextual asociado a una base de datos, además de abrir la Shell, se podrán obtener estadísticas, reparar o borrar la base de datos.



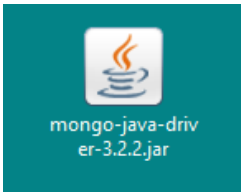
Desde el menú contextual de la Colección podremos ver los documentos, insertar, modificar o borrar documentos. Podremos cambiar el nombre de la Colección, hacer una copia o borrar la Colección.



10.- MongoDB desde Java.

Driver mongo-java.

El driver JAR para realizar conexiones y operaciones sobre la base de datos mongoDB se llama



El driver JAR de mongo-java debe incluirse en el proyecto creado con eclipse o netbeans.

Conectar a la base de datos mongoDB.

Podemos crear una conexión a la base de datos mongoDB mediante una instancia del tipo MongoClient, indicando el host y el puerto, por defecto localhost y 27017. Podemos cerrar la base de datos con close().

```
package conexion;
// importar clases
import com.mongodb.*;
import com.mongodb.client.*;
import java.util.logging.*;

public class Conexion {
    public static void main(String[] args) {
        // Desactiva los mensajes en la consola de MongoDB
        Logger.getLogger("org.mongodb.driver").setLevel(Level.SEVERE);
        System.out.println("Conexión MongoDB");
        // Creamos instancia de conexion a Mongodb
        MongoClient conexion = new MongoClient("localhost", 27017);
        System.out.println("Lista de bases de datos: ");
        // Recuperamos el listado de las bases de datos en una
        // esctructura MongoCursos que se recorre con un iterator
        MongoCursor<String> bases = conexion.listDatabaseNames().iterator();
        // Mientras haya datos en la lista de bases de datos
        // muestra la siguiente base de datos
        while (bases.hasNext()) {
            System.out.println("- "+bases.next());
        }
        // Cerramos la conexión
        conexion.close();
    }
}
```

Resultado:

```
Conexión MongoDB
Lista de bases de datos:
- admin
- config
- local
```

Añadir datos a la base de datos MongoBD.

Para insertar un documento(registro) en una colección(tabla) debemos crear una instancia de tipo MongoDatabase() para seleccionar la base de datos con la que vamos a trabajar.

También debemos crear una instancia del tipo MongoCollection() para indicar la colección (tabla) sobre la que realizaremos operaciones de inserción, actualización, consulta y borrado.

Además crearemos una instancia de tipo Document() para crear con append los diferentes campos del documento.

Por último insertaremos el documento en la colección con insertOne().

```
package conexion;
import com.mongodb.*;
import com.mongodb.client.*;
import java.util.logging.*;
import org.bson.*;

public class Insertar {
    public static void main(String[] args) {
        // Desactiva los mensajes en la consola de MongoDB
        Logger.getLogger("org.mongodb.driver").setLevel(Level.SEVERE);
        System.out.println("Conexión MongoDB");
        // Creamos instancia de Conexion a Mongodb
        MongoClient conexion = new MongoClient("localhost", 27017);

        //Si no existe la base de datos la crea
        MongoDatabase base = conexion.getDatabase("videoclub");

        //Crea una tabla si no existe y agrega datos
        MongoCollection tabla = base.getCollection("peliculas");

        //Crea un documento (registro) y añadir con append los campos
        Document miDocumento = new Document();
        miDocumento.append("titulo", "Pirata");
        miDocumento.append("tema", "Aventuras");
        miDocumento.append("duracion", 123);
        miDocumento.append("precio", 4.5);

        // Mediante insertOne insertamos el documento (registro)
        tabla.insertOne(miDocumento);

        // Cerramos la conexion a la base de datos
        conexion.close();

        System.out.println("Documento Insertado.");
    }
}
```

Listar los documentos de una colección MongoDB.

Para listar los documentos de una colección debemos recorrer el conjunto de resultados convirtiendo el objeto FindIterable que devuelve el método find() en un cursor, usando el método iterator(). El método iterator() devuelve un objeto MongoCursor y podemos acceder al primer documento con el método next(). Una vez sobre el documento, podremos aplicar métodos getString(), getInteger(), etc..., similares a los usados con JDBC para recuperar el valor de los campos.

```
package conexion;
import com.mongodb.*;
import com.mongodb.client.*;
```

```
import java.util.logging.*;
import org.bson.*;

public class Listar {
    public static void main(String[] args) {
        // Desactiva los mensajes en la consola de MongoDB
        Logger.getLogger("org.mongodb.driver").setLevel(Level.SEVERE);
        System.out.println("Conexión MongoDB");
        // Creamos instancia de Conexion a Mongodb
        MongoClient conexion = new MongoClient("localhost", 27017);

        //Si no existe la base de datos la crea
        MongoDatabase base = conexion.getDatabase("videoclub");

        //Crea una tabla si no existe y agrega datos
        MongoCollection tabla = base.getCollection("peliculas");

        // resultadoDocumentos almacena el resultado de la consulta
        MongoCursor<Document> resultadoDocumentos = tabla.find().iterator();

        System.out.println("listado:");
        Document miDocumento;
        // Recorremos el resultado de la consulta
        while (resultadoDocumentos.hasNext()) {
            miDocumento = resultadoDocumentos.next();
            System.out.println("Titulo:" + miDocumento.getString("titulo"));
            System.out.println("Tema:" + miDocumento.getString("tema"));
            System.out.println("Duración:" + miDocumento.getInteger("duracion"));
            System.out.println("Precio:" + miDocumento.getDouble("precio"));
            System.out.println("-----------------------------------------");
        }
        // Cerramos la conexion a la base de datos
        conexion.close();
    }
}
```

Consultar documentos de una colección MongoDB.

Para consultar un documento de una colección debemos acceder al primer documento de la colección convirtiendo el objeto FindIterable que devuelve el método find() en un cursor, usando el método iterator(). El método iterator() devuelve un objeto MongoCursor que podremos recorrer con el método next(). Una vez sobre el documento, podremos aplicar métodos getString(), getInteger(), etc..., similares a los usados con JDBC para recuperar el valor de los campos.

```
package conexion;
import com.mongodb.*;
import com.mongodb.client.*;
import static com.mongodb.client.model.Filters.*;
import java.util.logging.*;
import org.bson.*;

public class Consultar {
```

```
    public static void main(String[] args) {
        // Desactiva los mensajes en la consola de MongoDB
        Logger.getLogger("org.mongodb.driver").setLevel(Level.SEVERE);
        System.out.println("Conexión MongoDB");
        // Creamos instancia de Conexion a Mongodb
        MongoClient conexion = new MongoClient("localhost", 27017);

        //Si no existe la base de datos la crea
        MongoDatabase base = conexion.getDatabase("videoclub");

        //Crea una tabla si no existe y agrega datos
        MongoCollection tabla = base.getCollection("peliculas");

        // resultadoDocumentos almacena el resultado de la consulta
        MongoCursor<Document> resultadoDocumentos =
              tabla.find(eq("titulo", "Casper")).iterator();

        // Nos posicionamos en el primer documento del resultado
        Document miDocumento = resultadoDocumentos.next();

        // Mostranos los datos del documento
        System.out.println("Titulo:" + miDocumento.getString("titulo"));
        System.out.println("Tema:" + miDocumento.getString("tema"));
        System.out.println("Duración:" + miDocumento.getInteger("duracion"));
        System.out.println("Precio:" + miDocumento.getDouble("precio"));

        // Cerramos la conexion a la base de datos
        conexion.close();
    }
}
```

Para consultar los documentos de una colección debemos recorrer el conjunto de resultados convirtiendo el objeto FindIterable que devuelve el método find() en un cursor, usando el método iterator(). El método iterator() devuelve un objeto MongoCursor que podremos recorrer con el método next(). Una vez sobre el documento, podremos aplicar métodos getString(), getInteger(), etc…, similares a los usados con JDBC para recuperar el valor de los campos.

```
package conexion;

import com.mongodb.*;
import com.mongodb.client.*;
import static com.mongodb.client.model.Filters.*;
import java.util.logging.*;
import org.bson.*;

public class Consultar {

    public static void main(String[] args) {
        // Desactiva los mensajes en la consola de MongoDB
        Logger.getLogger("org.mongodb.driver").setLevel(Level.SEVERE);
        System.out.println("Conexión MongoDB");
```

```
        // Creamos instancia de Conexion a Mongodb
        MongoClient conexion = new MongoClient("localhost", 27017);

        //Si no existe la base de datos la crea
        MongoDatabase base = conexion.getDatabase("videoclub");

        //Crea una tabla si no existe y agrega datos
        MongoCollection tabla = base.getCollection("peliculas");

        // Indicamos que la ordenación será ascendente (1) o descendente (-1)
        // creando una instancia del tipo Document
        Document orden = new Document("duracion",1);

        // resultadoDocumentos almacena el resultado de la consulta
        MongoCursor<Document> resultadoDocumentos =
            tabla.find(and(gt("precio", 4), eq("tema", "Aventuras")))
            .sort(orden).iterator();

        System.out.println("listado:");
        Document miDocumento;
        // Recorremos el resultado de la consulta
        while (resultadoDocumentos.hasNext()) {
            miDocumento = resultadoDocumentos.next();
            System.out.println("Titulo:" + miDocumento.getString("titulo"));
            System.out.println("Tema:" + miDocumento.getString("tema"));
            System.out.println("Duración:" + miDocumento.getInteger("duracion"));
            System.out.println("Precio:" + miDocumento.getDouble("precio"));
            System.out.println("-----------------------------------------");
        }
        // Cerramos la conexion a la base de datos
        conexion.close();
    }
}
```

Actualizar documentos de una colección MongoDB.

Para actualizar documentos de una colección debemos crear instancias de tipo Document con el criterio y el nuevo valor. Utilizamos $set para solo cambiar el campo deseado y dejar el resto sin cambios. Lanzaremos con UpdateOne() la actualización del documento y si queremos cambiar varios documentos utilizamos updateMany().

```
package conexion;
import com.mongodb.*;
import com.mongodb.client.*;
import static com.mongodb.client.model.Filters.*;
import java.util.logging.*;
import org.bson.*;

public class Actualizar {

    public static void main(String[] args) {
        // Desactiva los mensajes en la consola de MongoDB
        Logger.getLogger("org.mongodb.driver").setLevel(Level.SEVERE);
```

```
        System.out.println("Conexión MongoDB");
        // Creamos instancia de Conexion a Mongodb
        MongoClient conexion = new MongoClient("localhost", 27017);

        //Si no existe la base de datos la crea
        MongoDatabase base = conexion.getDatabase("videoclub");

        //Crea una tabla si no existe y agrega datos
        MongoCollection tabla = base.getCollection("peliculas");

        // Creamos el criterio de actualización
        Document criterio = new Document("titulo", "Casper");
        // Creamos el nuevo valor
        Document nuevoValor = new Document("precio", 2.5);
        // Creamos el cambio con $set para cambiar solo el valor
        // del campo precio y dejar el resto sin cambios
        Document cambio = new Document("$set", nuevoValor);
        // Lanzamos la actualización
        tabla.updateOne(criterio, cambio);

        System.out.println("Registro Actualizado");

        // Cerramos la conexion a la base de datos
        conexion.close();
    }
}
```

Eliminar documentos de una colección MongoDB.

Para eliminar documentos de una colección debemos utilizar deleteOne() o deleteMany() y como parámetro usar los filtros de MongoDB sobre una pareja de campo valor creando una instancia del tipo Bson.

```
package conexion;
import com.mongodb.*;
import com.mongodb.client.*;
import static com.mongodb.client.model.Filters.*;
import java.util.logging.*;
import org.bson.conversions.*;

public class Eliminar {

    public static void main(String[] args) {
        // Desactiva los mensajes en la consola de MongoDB
        Logger.getLogger("org.mongodb.driver").setLevel(Level.SEVERE);
        System.out.println("Conexión MongoDB");
        // Creamos instancia de Conexion a Mongodb
        MongoClient conexion = new MongoClient("localhost", 27017);

        //Si no existe la base de datos la crea
        MongoDatabase base = conexion.getDatabase("videoclub");
```

```
        //Crea una tabla si no existe y agrega datos
        MongoCollection tabla = base.getCollection("peliculas");

        // Creamos el criterio de borrado con una instancia del tipo Bson
        Bson criterio = and(eq("tema","Drama"),gt("precio",1));
        // Lanzamos la eliminación
        tabla.deleteMany(criterio);

        System.out.println("Registro Eliminado");

        // Cerramos la conexion a la base de datos
        conexion.close();
    }
}
```

Número de documentos de una colección MongoDB.

Para devolver el número de documentos de una colección debemos utilizar el método count().

```
package conexion;
import com.mongodb.*;
import com.mongodb.client.*;
import java.util.logging.*;

public class Contar {
    public static void main(String[] args) {
        // Desactiva los mensajes en la consola de MongoDB
        Logger.getLogger("org.mongodb.driver").setLevel(Level.SEVERE);
        System.out.println("Conexión MongoDB");
        // Creamos instancia de Conexion a Mongodb
        MongoClient conexion = new MongoClient("localhost", 27017);

        //Si no existe la base de datos la crea
        MongoDatabase base = conexion.getDatabase("videoclub");

        //Crea una tabla si no existe y agrega datos
        MongoCollection tabla = base.getCollection("peliculas");

        // Mostramos los documentos de una colección con count()
        System.out.println("Documentos: " + tabla.count());

        // Cerramos la conexion a la base de datos
        conexion.close();
    }
}
```

11.- Exportar/Importar base de datos desde mongoDB.

Para Exportar datos de mongoDB utilizamos en la consola cmd la sentencia:

```
mongodump -h direccion:puerto -d base_de_datos -c coleccion -u usuario -p contraseña -o ruta_de_exportacion
```

Por ejemplo, para exportar desde mongodb la base de datos videoclub a la carpeta Descargas del usuario:

```
mongodump -h localhost:27017 -d videoclub -o "C:\Users\Mario G\Downloads"
```

Para Importar datos a mongoDB utilizamos en la consola cmd la sentencia:

```
mongorestore -h direccion:puerto -d base_de_datos -u usuario -p contraseña ruta_del_fichero_.bson_exportado
```

Por ejemplo, para importar la base de datos videoclub desde la carpeta creada en Descargas del usuario:

```
mongorestore -h localhost:27017 -d videoclub "C:\Users\Mario G\Downloads\videoclub"
```