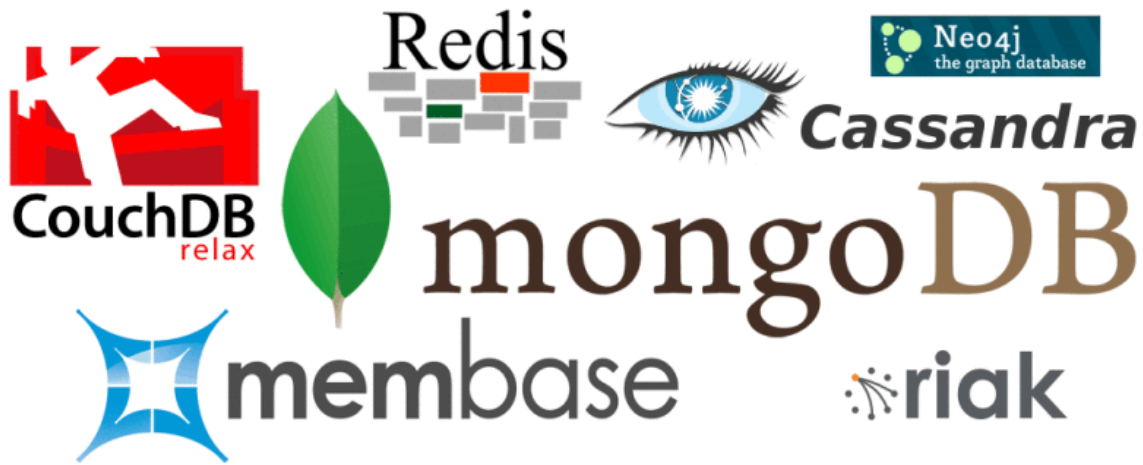


## AD07.- Bases de datos NoSQL



## 1.- Introducción

Durante la última década hemos visto el nacimiento de un nuevo tipo de bases de datos, conocidas bajo la denominación NoSQL.

Las bases de datos relacionales han dominado el mundo de la gestión de datos desde la década de los 70, pero el nacimiento de Internet y su auge como plataforma de aplicaciones ha puesto a prueba el dominio de las soluciones relacionales.

El **volumen de datos** al que debe hacer frente una aplicación web ha crecido exponencialmente durante los últimos años, así como el número de usuarios que utiliza las aplicaciones y servicios disponibles en Internet, y en consecuencia el **volumen de transacciones** y la demanda a la que se ven sometidas, ya que los usuarios esperan un tiempo de respuesta inmediato en sus interacciones online con el website.

## 1.1.- Tipos de bases de datos NoSQL

Las tecnologías NoSQL han evolucionado para dar respuesta a distintos problemas, y aunque tienen muchos aspectos en común, también son muy diferentes entre sí. Dada la diversidad de tecnologías NoSQL, habitualmente se clasifican en cuatro grupos, por su forma de modelar los datos:

- **Bases de datos Clave-Valor:** Tienen el modelo de datos más sencillo de todos, una clave indexada asociada a un valor, que desde el punto de vista de la base de datos es información opaca que simplemente almacena y recupera asociada a la clave. El consumidor de esta información es responsable de conocer la estructura de la información almacenada. Están diseñadas para escalar masivamente manteniendo un tiempo de respuesta muy rápido y disponibilidad total. Se suelen usar para almacenar información de sesión, preferencias o perfiles de usuario, carritos de la compra y en general como cachés de cualquier conjunto de información que se pueda recuperar por una clave. Algunos ejemplos son Redis, Riak o Aerospike.
- **Orientadas a Documento:** Utilizan el modelo de documento, mayoritariamente en formato JSON, para almacenar y consultar información. Permiten gestionar información con complejas estructuras jerárquicas, y ofrecen índices secundarios y completos lenguajes de consulta y agregación de datos. Esto unido a la flexibilidad del esquema de datos las convierten en las más versátiles y de propósito general. Dentro de este grupo tenemos tecnologías como MongoDB, CouchDB o CouchBase entre otras.
- **Orientadas a Grafos:** El modelo de datos se centra en entidades y las relaciones entre éstas. Tanto las entidades (nodos del grado) como las relaciones (aristas) pueden además tener atributos. Una entidad puede tener numerosas relaciones con cualquier otra entidad. Recorrer las uniones entre entidades a través de estas relaciones es el fuerte de las bases de datos orientadas a grafos, y permiten hacerlo con gran velocidad, independientemente del volumen de datos, lo que posibilita explorar conexiones entre entidades que de otra forma sería muy difícil con las bases de datos relacionales. El caso de uso más conocido de este tipo de bases de datos son las redes sociales. Ejemplos de esta tecnología son Neo4j, OrientDB o Titan.
- **Orientadas a Columnas:** Este tipo de bases de datos son similares a una tabla en las bases de datos relacionales, de hecho derivan en su mayoría del modelo BigTable publicado por Google, pero un registro puede contener cualquier número de columnas (o familias de columnas). Son ideales para realizar consultas y agregaciones sobre grandes cantidades de datos cuando éstas se pueden determinar previamente y no cambian con frecuencia. En este grupo encontramos ejemplos como Cassandra o HBase.

## 2.- Bases de datos NoSQL: MongoDB

MongoDB es un sistema de base de datos multiplataforma orientado a documentos, se pueden almacenar cualquier tipo de contenido sin obedecer a un modelo o esquema. Está escrito en C++ con lo que es bastante rápido a la hora de ejecutar sus tareas. Además, está licenciado como GNU AGPL 3.0, de modo que se trata de un software de licencia libre. Funciona en sistemas operativos Windows, Linux, OS X y Solaris.

Una de sus características principales es la velocidad y la sencilla forma que tiene para hacer consultas a los contenidos.. MongoDB se utiliza para cualquier aplicación que necesite almacenar datos semiestructurados, caso de aplicaciones CMS, aplicaciones móviles, de juegos, o plataformas e-commerce. MongoDB no soporta JOINS ni transacciones, aunque posee índices secundarios, un propio lenguaje de consulta muy expresivo, operaciones atómicas en un solo documento (pero no soporta transacciones de múltiples documentos), y lecturas consistentes.

La mayor diferencia entre las bases de datos relacionales y MongoDB es la forma en que se crea el modelo de datos, el modelo relacional es un modelo rígido y estructurado mientras que el modelo MongoDB es un modelo dinámico. En la siguiente tabla se muestra la terminología utilizada en el modelo relacional y el modelo de documento de MongoDB:

Con el modelo MongoDB se pasa de un modelo de datos rígido basado en estructuras de datos bidimensionales, formado por tablas, filas y columnas a un modelo de datos de documentos rico y dinámico con subdocumentos y matrices embebidas. En MongoDB se pueden crear colecciones sin definir su estructura, también se puede alterar la estructura de los documentos simplemente añadiendo nuevos campos o borrando los ya existentes. Esta característica convierte a Mongo en una BD muy flexible con respecto a las alternativas relacionales.

MongoDB almacena documentos JSON (JavaScript Object Notation) en una representación binaria llamada BSON (Binary JSON). BSON es una serialización codificada en binario de documentos JSON, Soporta todas las características de JSON e incluye los tipos de datos int, long, float o arrays. El documento (el registro en el modelo relacional) representa la unidad básica de datos en MongoDB.

## 2.1.- Estructuras JSON

JSON (JavaScript Object Notation - Notación de Objetos de JavaScript) es un formato ligero de intercambio de datos. Leerlo y escribirlo es simple para humanos, mientras que para las máquinas es simple interpretarlo y generarlo. Está basado en un subconjunto del Lenguaje de Programación JavaScript, *Standard ECMA-262 3rd Edition - Diciembre 1999*. JSON es un formato de texto que es completamente independiente del lenguaje, pero utiliza convenciones que son ampliamente conocidos por los programadores de la familia de lenguajes C, incluyendo C, CH, C#, Java, JavaScript, Perl, Python, y muchos otros. Estas propiedades hacen que JSON sea un lenguaje ideal para el intercambio de datos. (Fuente: <http://www.json.org/json-es.html>).

JSON está constituido por dos estructuras:

- **Una colección de pares de nombre/valor.** En varios lenguajes esto es conocido como un objeto, registro, estructura, diccionario, tabla hash, lista de claves o un array asociativo.
- **Una lista ordenada de valores.** En la mayoría de los lenguajes, esto se implementa como arrays, vectores, listas o secuencias. Estas son estructuras universales: virtualmente todos los lenguajes de programación las soportan de una forma u otra. Es razonable que un formato de intercambio de datos que es independiente del lenguaje de programación se base en estas estructuras.

Estas son estructuras universales: virtualmente todos los lenguajes de programación las soportan de una forma u otra. Es razonable que un formato de intercambio de datos que es independiente del lenguaje de programación se base en estas estructuras.

En JSON, se presentan de estas formas:

- Como un **objeto**, conjunto desordenado de **pares nombre/valor**. Un objeto comienza con { (llave de apertura) y termine con } (llave de cierre). Cada nombre es seguido por : (dos puntos) y los pares *nombre/valor* están separados por , (coma). En el ejemplo creo un objeto persona con nombre y oficio, y un objeto zona con su código y su nombre:

```
{ "persona": { "nombre": "Alicia", "oficio": "Profesora", "ciudad": "Talavera" },  
  "zona": { "codzona": 10, "nombre": "Madrid" } }
```

- Un **array**, es decir, una colección de valores. Un array comienza con [ (corchete izquierdo) y termina con ] (corchete derecho). Los valores se separan por , (coma). En el ejemplo creo el objeto persona, un array de dos elementos, no tienen por qué tener los mismos pares *nombre/valor*:

```
{ "persona": [  
    { "nombre": "Alicia", "oficio": "Profesora", "ciudad": "Talavera" },  
    { "nombre": "María Jesús", "oficio": "Profesora" }  
]
```

- Un **valor** puede ser una cadena de caracteres con comillas dobles, o un número, o true o false o null, o un objeto o un array. Estas estructuras pueden anidarse. En el ejemplo se muestra un objeto de nombre ventana con distintos tipos de nombre/valor "

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

- Una **cadena de caracteres** es una colección de cero o más caracteres Unicode, encerrados entre comillas dobles, usando barras divisorias invertidas como escape. Un carácter está representado por una cadena de caracteres de un único carácter. Una cadena de caracteres es parecida a una cadena de caracteres C o Java.
- Un **número** es similar a un número C o Java, excepto que no se usan los formatos octales y hexadecimales.

Los espacios en blanco pueden insertarse entre cualquier par de símbolos *nombre/valor*.

## 2.2.- Instalación MongoDB

En este apartado instalaremos la base de datos NoSQL *MongoDB*. Descargamos el archivo desde la página <https://www.mongodb.com/download-center>. (*link: https://www.mongodb.com/download-center*.) Podemos elegir entre una versión Community o Enterprise y dentro de cada una elegiremos la versión y el sistema operativo.c

La base de datos se instala por defecto en (*Archivos de programa*) *C:\Program Files\MongoDB*. Para arrancar la base de datos buscaremos la carpeta bin de MongoDB (*C:\Program Files\MongoDB\Server\...\bin*), dentro de esa carpeta se encuentra el archivo *mongod.exe*, que es el que arranca la BD.

Una vez arrancada la BD, se queda escuchando por el puerto 27017 las conexiones de los clientes.

Si ahora queremos conectamos como cliente a la BD, ejecutaremos el programa *mongo.exe* de la carpeta *bin*, también desde la línea de comando. Desde este cliente podremos trabajar con la base de datos.

## 2.3.- Operaciones básicas en MongoDB

Todos los comandos para operar con esta base de datos se escriben en minúscula, los mas 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, creará en el momento que anadamos un objeto JSON, con las funciones o *insert()*, *insertOne()* o *insertMany()*. Si queremos saber el número de documentos dentro de las colecciones, utilizaremos la función **count**, escribiremos: **db.nombre\_coleccion.count()**. También se utilizan las funciones *size()* y *length()*.
- Para añadir comentarios utilizamos los caracteres **//** de comentario de Java.

### 3.- Operaciones CRUD en MongoDB

En este apartado veremos cómo se realizan las operaciones CRUD (Create, read, update y delete) en mongodb.

## Para saber más

En el siguiente enlace puedes obtener información y ayuda amplia para realizar las operaciones CRUD en MongoDB

MongoDB CRUD operations (*link: <https://docs.mongodb.com/manual/crud/>* )



### 3.1.- Crear registros

MongoDB proporciona los siguientes métodos para insertar documentos en una colección:

```
db.collection.insertOne()
db.collection.insertMany()
```

En MongoDB, las operaciones de inserción apuntan a una sola colección. Todas las operaciones de escritura en MongoDB son atómicas en el nivel de un solo documento.

```
db.users.insertOne(  ← collection
{
  name: "sue",        ← field: value
  age: 26,            ← field: value
  status: "pending"   ← field: value
}                    } document
)
```

#### IDENTIFICADOR DE OBJETOS

Los identificadores de cada documento (registro) son únicos. Se asignan automáticamente al crear el documento, se generan de forma rápida y ordenada. También se pueden crear de forma manual. Es un número hexadecimal que consta de 12 bytes, los primeros 4 son una marca de tiempo, los tres siguientes la identificación de la máquina, 2 bytes de identificador de proceso y un contador de 3 bytes empezando en un número aleatorio. **El *Objectid* o *\_id***, es como si fuese la clave del documento, no se repetirá en una colección. Si un documento no tiene *\_id*, *MongoDB* se lo asignará automáticamente, es lo que ocurre cuando insertamos y no indicamos el identificador.

## 3.2.- Consultar registros

MongoDB proporciona los siguientes métodos para leer documentos de una colección:

```
db.collection.find()
```

Es posible especificar filtros de consulta o criterios que identifiquen los documentos a devolver.

<code>db.users.find(</code>	 <b>collection</b>
<code>  { age: { \$gt: 18 } },</code>	 <b>query criteria</b>
<code>  { name: 1, address: 1 }</code>	 <b>projection</b>
<code>).limit(5)</code>	 <b>cursor modifier</b>

Con el operador `.limit(5)`, limitamos la salida a los primeros 5 elementos.

Si queremos saber el número de registros que devuelve una consulta pondremos

- `db.users.find({filtros}).count();`

Si, por ejemplo, se desea que la salida sea ascendente por uno de los campos, utilizamos el operador `.sort`. Para obtener los datos de la colección ordenados por nombre escribimos: (El número que acompaña a la orden indica el tipo de ordenación, 1 ascendente y -1 descendente).

- `db.users.find().sort({name:1});`

En el **query criteria o filtro** indicamos la condición de búsqueda, podemos añadir los pares *nombre valor* a buscar. Si omitimos este parámetro devuelve todos los documentos.

En **projection o 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, ...}*) También podemos poner *true* o *false* en lugar de 1 o 0.

En las búsquedas podemos utilizar una gran cantidad de selectores de comparación, lógicos etc.

### 3.3.- Actualizar registros

Las operaciones de actualización modifican documentos existentes en una colección . MongoDB proporciona los siguientes métodos para actualizar documentos de una colección:

```
db.collection.updateOne()

db.collection.updateMany()

db.collection.replaceOne()
```

En MongoDB, las operaciones de actualización apuntan a una sola colección. Todas las operaciones de escritura en MongoDB son atómicas a nivel de un solo documento.

Se pueden especificar criterios o filtros que identifiquen los documentos a actualizar. Estos filtros usan la misma sintaxis que las operaciones de lectura.

```
db.users.updateMany(           ← collection
  { age: { $lt: 18 } },        ← update filter
  { $set: { status: "reject" } } ← update action
)
```

En **update filter o filtro de búsqueda**, se indica la condición para localizar los registros o documentos a modificar.

En **update action o cambios a realizar**, se especifican los cambios que se desean hacer. Hay que tener cuidado al utilizar esta orden, pues en cambios a realizar se indica cómo quedará el documento que se busca si este existe, es decir: el resultado final del documento es lo que se escriba en cambios a realizar.

Si updateOne(), updateMany() o replaceOne() incluye **upsert : true** y ningún documento coincide con el filtro especificado, la operación crea un nuevo documento y lo inserta. Si hay documentos coincidentes, la operación modifica o reemplaza el documento o documentos coincidentes.

## 3.4.- Borrar registros

MongoDB proporciona los siguientes métodos para eliminar documentos de una colección:


```
db.collection.deleteOne()

db.collection.deleteMany()
```

Todas las operaciones de escritura en MongoDB son atómicas en el nivel de un solo documento.

Se pueden especificar criterios o filtros que identifiquen los documentos que se eliminarán. Estos filtros usan la misma sintaxis que las operaciones de lectura.

```
db.users.deleteMany(
  { status: "reject" }
)
```



collection

delete filter

## 4.- Trabajar desde Java

Para trabajar en Java con MongoDB necesitamos descargar el driver desde la URL de MongoDB <https://mongodb.github.io/mongo-java-driver/>

Antes de trabajar con MongoDB definamos BSON: **BSON** es un formato de serialización binaria de JSON, se utiliza para almacenar documentos y hacer llamadas a procedimientos en MongoDB. La especificación BSON se encuentra en *bsonspec.org*. BSON soporta los siguientes tipos de datos como valores en los documentos, cada tipo de dato tiene un número y un alias que se pueden utilizar con el operador `$type` para consultar los documentos por tipo BSON. Puedes consultar los tipos BSON en el siguiente [enlace \(link: https://docs.mongodb.com/manual/reference/bson-types/ \)](https://docs.mongodb.com/manual/reference/bson-types/)

### Para saber más

En el siguiente enlace puedes descargar el driver y consultar el manual de referencia para trabajar con MongoDB en Java.

Mongo Java Driver (*link: <https://mongodb.github.io/mongo-java-driver/> )*

## 4.1.- Conexión a la BD

Para conectarnos a la base de datos creamos una instancia **MongoClient**, por defecto crea una conexión con la base de datos local, y escucha por el puerto 27017. Todos los métodos relacionados con operaciones CRUD (Create, Read, Update and Delete) en Java se acceden a través de la interfaz **MongoCollection**. Las instancias de **MongoCollection** se pueden obtener a partir de una instancia MongoClient por medio de una **MongoDatabase**. Así pues para conectarme a la base de datos *mibasedatos* ya la colección *amigos* escribiré lo siguiente:

```
MongoClient mongoClient = MongoClient.create(
    MongoClientSettings.builder()
        .applyToClusterSettings(builder ->
            builder.hosts(Arrays.asList(new ServerAddress("localhost", 27017))))
        .credential(credential)
        .build());
MongoDatabase database = mongoClient.getDatabase("mibasedatos");
MongoCollection<Document> collection = database.getCollection("amigos");
```

MongoCollection es una interfaz genérica: el parámetro de tipo TDocument es la clase que los clientes utilizan para insertar o modificar los documentos de una colección, y es el tipo predeterminado para devolver búsquedas (find) y agregados (aggregate). El método de un solo argumento getCollection devuelve una instancia de MongoCollection < Document> y así es como podemos trabajar con instancias de la clase de documento.

### CONSULTAR DOCUMENTOS

El método **find()** devuelve un cursor, devuelve una instancia **FindIterable**. Podemos utilizar el método *iterator()* para recorrer el cursor. En el ejemplo recuperamos todos los documentos de la colección y se visualizan en formato *JSON*:

```
MongoCursor<Document> result = collection.find(filter)
    .projection(project)
    .sort(sort).iterator();
try {
    while (result.hasNext()) {
        System.out.println(result.next().toJson());
    }
} finally {
    result.close();
}
```

