

## Java para el acceso a datos en MongoDB

### Contenido

<b>Java para el acceso a datos en MongoDB</b> .....	1
<b>Objeto Java y Objeto MongoDB</b> .....	1
<b>Conectar con la BD.</b> .....	2
<b>Operaciones CRUD sobre la BD</b> .....	3
<b>Insertar un documento nuevo en una colección</b> .....	3
<b>Actualizar un documento:</b> .....	4
<b>Leer un documento de la BD</b> .....	5
<b>Método find():</b> .....	5
<b>Método first()</b> .....	6
<b>Método toJson():</b> .....	6
<b>Localizar todos los documentos que cumplan un criterio de búsqueda</b> .....	6
<b>Clase FindIterable</b> .....	6
<b>Listar todos los documentos de una colección</b> .....	6
<b>Borrar documentos:</b> .....	6

### ¿Qué es un BSON en MongoDB?

BSON= Binary JSON. Que representa aun superset de JSON, es decir:

- Almacena datos en binario.
- Incluye un conjunto de tipos de datos que no están incluidos en JSON:
  - o ObjectId, Date, BinData

Los documentos BSON tienen algunas restricciones:

- El tamaño no puede ser superior a 16MB
- El atributo **\_id** está reservado para representar el identificador de documento (clave primaria).
- Los nombres de los campos nunca pueden empezar por \$ y no pueden contener un "." como separador.

```
var vecino = {
    nombre: "Antonio",
    fechanace: new Date("Jan 27, 2000"),
    hijos: ["Rosa", "Manuel", "Carmen"].
    fechatomadatos = new Timestamp()
}
```

A tener en cuenta en Mongo: Es sensible a los tipos de datos y a las mayúsculas (hijos, Hijos)

## Objeto Java y Objeto MongoDB.

### Objeto JAVA

```
public class Cliente {
    private String nombre;
    private Date f_nac;
    private String calle;
    private Integer bono_regalo;
}
```

### Objeto DBObject

```
"nombre": "Carlos Fernández",
"f_nac": "2000",
"calle": "Av. Primero Mayo 1",
"bono_regalo": {
    "$numberInt": "100"
}
```

Cuando trabajamos Java con MongoDB en lugar de objetos vamos a manejar DBObject. Esta librería maneja objetos de la clase BasicDBObject que implementa la interfaz DBObject. Se trata de una clase que recibe dos parámetros: el documento y el valor de un campo.

Enlace a DBObject:

<https://mongodb.github.io/mongo-java-driver/3.4/javadoc/com/mongodb/BasicDBObject.html>

### Secuencia de operaciones.

1. Usar un cliente o MongoClient para conectarnos a la base de datos. Como siempre, necesitamos url, usuario, contraseña y base de datos.
2. Crear una colección que nos permitirá trabajar con un conjunto de documentos. Por ejemplo, una colección de Empleados.
3. Crear un objeto del tipo documento Empleado. Vamos a representarlo mediante un POJO y luego pasaremos sus parámetros a un BasicDBObject para referirnos a un documento concreto. Con un DBObject ya podemos interactuar con la base de datos

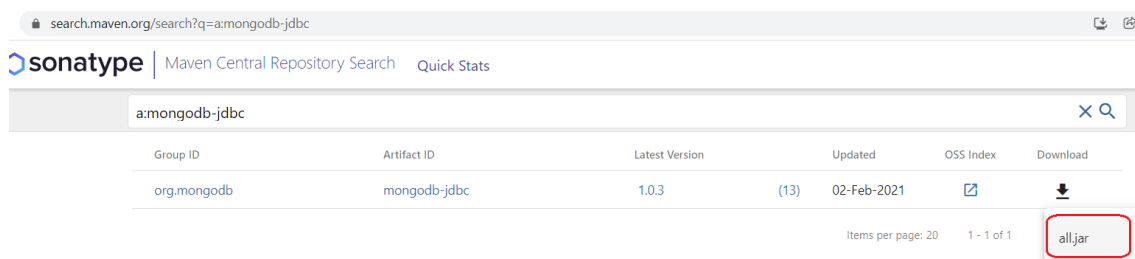
Las operaciones de CRUD para el acceso a MongoDB desde una aplicación Java, vamos a verlas con un ejemplo práctico.

- BD: ejemplodb
- Colección: clientes.

## Conectar con la BD.

Lo primero es obtener el driver JDBC para MongoDB, seleccionando la opción para todas las dependencias:

<https://search.maven.org/search?q=a:mongodb-jdbc>



search.maven.org/search?q=a:mongodb-jdbc

sonatype | Maven Central Repository Search | Quick Stats

a:mongodb-jdbc

Group ID	Artifact ID	Latest Version	Updated	OSS Index	Download
org.mongodb	mongodb-jdbc	1.0.3	(13) 02-Feb-2021	<a href="#">[i]</a>	<a href="#">Download</a>

Items per page: 20 1 - 1 of 1

all.jar

**Ejemplo:** Base de datos **ejemplodb** con la colección **clientes**.

Para conectarnos a la BD desde Java tenemos que crear un objeto de tipo MongoClient, que se encarga de gestionar internamente las conexiones.

Previamente, debemos saber el servidor donde está alojada o la ubicación URI de nuestra base de datos, creando un objeto de tipo MongoClientURI.

A partir de la MongoClient conexión, podemos obtener o conectarnos a una colección concreta de la BD.

```
MongoClientURI uri = new MongoClientURI("mongodb+srv://admin:Admin-123@cluster0.vsmws.mongodb.net/ejemplobd?retryWrites=true&w=majority");  
MongoClient mongoClient = new MongoClient(uri);
```

El acceso a la BD “ejemplobd” de la instancia.

```
MongoDatabase database = mongoClient.getDatabase("ejemplobd");
```

El acceso a la colección “clientes”

```
MongoCollection collection = database.getCollection("clientes");
```

Desconectar de la base de datos:

```
mongoClient.close();
```

MongoClient realiza la conexión con la base de datos. El constructor por defecto se conecta con localhost al puerto 27017. Además, lanza una UnknownHostException cuando no encuentra un servidor funcionando

DBCollection nos permite interactuar con la colección y también realizar operaciones CRUD.

Saber más: Enlace a la interfaz MongoCollection y sus métodos:

[Interface MongoCollection<TDocument>](#)

## Operaciones CRUD sobre la BD.

### Insertar un documento nuevo en una colección.

Operaciones para insertar o crear un documento nuevo en la colección:

- db.collection.insert()
- db.collection.insertOne()
- db.collection.insertMany()

**Ejemplo: Insertar un cliente nuevo, empleado insertOne():**

```

/**
 * Método que inserta un documento en una collection
 *
 * @param cliente
 * @param collection
 */
public static void insertDocument(Cliente cliente, MongoCollection collection) {
    //creamos el objeto Document
    Document document = new Document();
    //escribimos los datos del nuevo cliente
    document.append("nombre", cliente.getNombre());
    document.append("f_nac", cliente.getF_nac());
    document.append("calle", cliente.getCalle());
    document.append("bono_regalo", cliente.getBono_regalo());
    //actualizamos la colección, añadiendo el cliente nuevo
    collection.insertOne(document);
}

```

### Ejemplo: Insertar una lista de documentos empleado insertMany():

```

public static void insertManyDocuments(ArrayList<Cliente> lista,
    MongoCollection collection) {
    List<Document> listdocuments = new ArrayList<>();
    for (Cliente c : lista) {
        Document document = new Document();
        //escribimos los datos del nuevo cliente
        document.append("nombre", c.getNombre());
        document.append("f_nac", c.getF_nac());
        document.append("calle", c.getCalle());
        document.append("bono_regalo", c.getBono_regalo());
        listdocuments.add(document);
    }
    collection.insertMany(listdocuments);
}

```

### Actualizar un documento:

- db.collection.updateOne(): actualiza un documento de una colección. Previamente debemos localizar o filtrar por algún valor del documento y, en el caso de que haya más de un documento filtrado, actualizará solo el primero.
- db.collection.updateMany(): actualiza un conjunto de documentos en una colección.
- db.collection.replaceOne(): reemplaza un document completo.

### Método updateOne():

Ejemplo: Actualizar el bono regalo de un cliente localizado por nombre.

En MongoDB sería así:

```

clientes.update(           // colección
{ nombre: { $eq: "Rosa Ruiz"}}, // criterio
{ $set: {bono_regalo: 500}} // modificación
)

```

En Java:

```
//Ejemplo: localizo documento con el nombre = "Rosa Ruiz".
//Actualizo bono_regalo a 500
Document bson = new Document("nombre", "Rosa Ruiz");
Document bson1 = new Document("bono_regalo", 500);
//en el documento escribo el campo a modificar
Document bson2 = new Document("$set", bson1);
collection.updateOne(bson, bson2);
```

### Método replaceOne():

Si queremos reemplazar un documento por otro, podemos usar **replaceOne()**:

Ejemplo: queremos reemplazar todos los datos del documento del cliente de nombre "Carlos Fernández".

```
//busco el documento que quier reemplazar
Document bson = new Document("nombre", "Carlos Fernández");
//creo el nuevo documento
Document bson1 = new Document();
bson1.append("nombre", "El nuevo Carlos");
bson1.append("f_nac", tools.parseUtilDate("2001-05-12"));
bson1.append("calle", "La nueva calle");
bson1.append("bono_regalo", 10);
//reemplazo
collection.replaceOne(bson, bson1);
```

En el siguiente apartado veremos como filtrar los documentos usando Filters.

```
collection.replaceOne(Filters.eq("_id", document.get("_id")), document);
```

### Leer un documento de la BD.

La información está en los documentos de una colección. En nuestro caso, en la colección **clientes**.

### Método find():

- Permite buscar documentos de una colección.
- Si no le pasamos ningún parámetro, busca todos los elementos de la colección.
- Se le pueden pasar parámetros (clave, valor) para la búsqueda.
- Si tenemos colecciones muy grandes, no devuelve todos los documentos, sino los primeros. Por eso conviene establecer un límite **limit(int ndocs)**

Parámetros de búsqueda para find():

- **find(eq("name", "Ana García"))** - Buscará todos los documentos que tengan como nombre Ana García.
- **find(gt("bono\_regalo", 5))** - Buscará todos los documentos que tengan más de 5 como bono regalo.
- **find(gte("bono\_regalo", 5))** - Buscará todos los documentos que tengan 5 o más de bono\_regalo.
- **find(and(gte("bono\_regalo", 5), lte("bono\_regalo", 8)))** - Buscará todos los documentos que tengan entre 5 y 8 puntos de bono\_regalo (ambos valores incluidos)
- **find(or(eq("nombre", "Ana García"), eq("calle", "Av Primero de mayo")))** - Buscará todos los documentos que cumplan ambas condiciones.

### Método first()

- Recupera el primer documento de la colección.
- Si tenemos una consulta con parámetros de búsqueda, recupera el primero que encuentra.

### Método toJson():

- Permite obtener el resultado de una consulta con formato de JSON.

En nuestro ejemplo:

```
Document doc = (Document) collection.find().first();
System.out.println("Primer documento: ");
System.out.println(doc.toJson());
```

### Localizar todos los documentos que cumplan un criterio de búsqueda

#### Clase FindIterable:

Iterar los documentos que cumplan un criterio.

Un objeto FindIterable se utiliza para hacer iteraciones sobre los documentos recuperados de una búsqueda o consulta.

En nuestro ejemplo: Recuperar un número `ndocs` documentos de una colección.

```
FindIterable finditerable = database.getCollection("clientes").find().limit(ndocs);
Iterator<Document> it = finditerable.iterator();
while (it.hasNext()) {
    Document doc = it.next();
    System.out.println("doc= " + doc.toJson());
}
```

### Listar todos los documentos de una colección.

```
List<Document> lista = (List<Document>) collection.find().into(new ArrayList<Document>());
for (int i = 0; i < lista.size(); i++) {
    System.out.println(lista.get(i));
}
```

La estrategia es cargar en un ArrayList en memoria la lista de documentos de la colección y recuperarlos todos con un bucle

### Borrar documentos:

Las opciones que hay para el borrado de los documentos de una colección son:

- `db.collection.remove()`
- `db.collection.deleteOne()`
- `db.collection.deleteMany()`

Para borrar un documento concreto, lo primero es localizarlo con una búsqueda por alguna de sus claves.

Ejemplo: borrar el documento de la colección clientes cuyo name sea Food AGL.

```
Document doc = (Document) collection.find(eq("nombre", "Ana García")).first();  
System.out.println(doc.toJson());  
collection.deleteOne(eq("nombre", "Ana García"));
```