**Programación Java de BD MongoDB**

**Objetivos:**

* Realizar conexiones con BD MongoDB
* Consultar datos de un documento
* Acceder a colecciones
* Añadir documentos a una colección
* Añadir atributos a un documento
* Modificar valores de atributos de un documento
* Eliminar documentos de una colección

**EJERCICIO 1**

Realiza un programa Java que conecta con la base de datos **geografía de MongoDB,** presenta el siguiente menú y, en función de la opción elegida, realiza sobre la colección **ccaa** la operación solicitada en las condiciones que se indican a continuación:

**1.- Obtener comunidades autónomas y capitales**

**2.- Obtener comunidades autónomas con habitantes comprendidos entre valores**

**3.- Obtener comunidades autónomas uniprovinciales.**

**4.- Obtener capitales de comunidad autónoma con más habitantes que**

**5.- Obtener comunidades autónomas sin fecha de estatuto**

**6.- Obtener provincias de comunidad autónoma**

**7.- Crear fichero JSON**

**Explicación previa de consultas de la API de MongoDB Driver Java:**

Las consultas se realizan con el método **find** de *Collection* o con el método **aggregate** si se trata de consultas de agregación. El método **find** tiene dos sintaxis:

* **find():** Devuelve todos los documentos de la colección con todos sus atributos
* **find (Bson filtro)** : Devuelve los documentos de la colección que cumplen la condición de filtro, con todos sus atributos. El filtro se usa con objetos *Bson* de filtro de igual forma que los vistos en la actividad anterior para el método **updateOne**.

Tanto el método **find** como el método **aggregate** devuelven un objeto ***FindIterable<Document>.***

A un objeto ***FindIterable***y, por tanto, a lo devuelto por **find** y por **aggregate** podemos aplicarles varios métodos como:

* **iterator() :** Devuelve un objeto ***MongoCursor<Document>*** que es una colección java de objetos *Document*. Un objeto ***MongoCursor<Document>*** lo podemos recorrer con los métodos **hasNext()** y **next**() para acceder de uno en uno a cada ***Document****.*

Ejemplo:

**MongoCursor<Document>** cursor=coleccion.find().**iterator();**

while( cursor.**hasNext**()){

Document doc=cursor.**next();**

System.out.println("Com. Autonoma" +doc.getString("nombre"));

}

* **first()** : Devuelve un objeto *Document* que será el primero de los datos de la consulta realizada. Si la consulta no hubiera devuelto objetos *Document*, el método devuelve null.

Ejemplo:

**Document doc**=coleccion.find(eq("codigo","02")).**first**();

if(doc==null)

System.out.println("No existe la Comunidad Autónoma");

else {

System.out.println("Comunidad Autónoma : "

+(String)doc.get("nombre"));

}

* **limit(num):** Selecciona los **num** primeros objetos *Document* de la consulta, devolviendo otro ***FindIterable<Document>.***

Ejemplo:

**MongoCursor<Document>** cursor = coleccion.find().**limit(3).**iterator();

* **sort(ascending(atributo)) | sort(descending(atributo)):** Devuelve los resultados de la consulta ordenados ascendente o descendentemente por el atributo indicado. Los nuevos resultados se devuelven en un ***FindIterable<Document>.***

Ejemplo:

**MongoCursor<Document>** cursor = coleccion.find().**sort(ascending("nombre")).**iterator();

* **projection(include(atrib1, atrib2, …) ):** Devuelve en los resultados de la consulta únicamente los atributos incluidos en **include**. Se realiza una proyección.

Ejemplo:

**MongoCursor<Document>** cursor = coleccion.find().**projection(include("nombre",** **"habitantes")).**iterator();

* **projection(exclude (atrib1, atrib2, …) ):** Devuelve en los resultados de la consulta todos los atributos de los objetos *Document* excepto los atributos incluidos en exclude.

Ejemplo:

**MongoCursor<Document>** cursor = coleccion.find().**projection(exclude("\_id")).**iterator();

* **into(TClase ):** Devuelve los resultados de la consulta en una colección Java de la clase pasada como parámetro en un constructor.

Ejemplo:

**ArrayList<Document>** lista = coleccion.find()

.**into(new ArrayList<Document> ());**

**OPCIÓN 1:**

Se muestran los nombres de todas las comunidades autónomas y los de sus capitales con formato JSON.

Debes realizarlo con **projection.**

**OPCIÓN 2:**

Se pide por teclado un valor mínimo de habitantes y un valor máximo y se escriben en pantalla en mayúsculas los nombres y habitantes de las comunidades autónomas que tienen un número de habitantes comprendido entre los anteriores. No hace falta comprobar que el mínimo es menor que el máximo.

Se debe realizar con un filtro **and** que opera con los filtros **lt** y **gt**.

El listado debe salir ordenado por habitantes descendentemente.

**OPCIÓN 3:**

Se escriben en pantalla los nombres de todas las comunidades autónomas que tienen una sola provincia en el atributo array provincias.

**OPCIÓN 4:**

Se pide por teclado un número de habitantes y se escriben en pantalla los nombres de capitales de comunidades autónomas que tienen más habitantes que los que se hayan introducido por teclado. El listado debe estar ordenado alfabéticamente por nombres de las capitales.

Para extraer los datos de cada subdocumento capital tendrás que obtenerlo previamente con el método **get** de cada ***Document*** de una comunidad autónoma.

**OPCIÓN 5:**

Se escriben en pantalla los nombres de todas las comunidades autónomas que no tienen atributo **fecha\_estatuto** o que lo tienen a valor nulo.

**OPCIÓN 6:**

Se pide por teclado el código de una comunidad autónoma y, si existe, se escribe en pantalla su nombre y los nombres de sus provincias (uno en cada línea)

**OPCIÓN 7:**

Se graba un fichero de texto, cuya ruta y nombre se dan por teclado, el JSON correspondiente a la consulta de todas las comunidades autónomas con los atributos **nombre, capital, provincias, habitantes, extension**.

Investiga como hacer que la salida JSON del método **toJson** represente cada atributo en una línea y con indentación o sangrado.