UD 05. Bases de datos NoSQL

1. Introducción

Las bases de datos NoSQL:

- No siguen el modelo clásico de las BD relacionales.
- No utiliza SQL para consultas.
- No utiliza estructuras fijas de almacenamiento.

Intentan resolver probemas que las BD relacionales no pueden:

- Almacenamiento masivo.
- Consultas masivas.
- Problemas de persistencia
- Respuesta rápida.

2. Características BD NoSQL

Principales características

- Capacidad almacenar grandes volúmenes de datos, estructurados, semi-estructurados o sin estructura
- Ausencia de esquema
- Escalabilidad horizontal, para mejorar el rendimiento simplemente se añaden más nodos.
- Velocidad
- Cada registro (fila) puede contener una definición de tipo de dato diferente.

3. Tipos BD NoSQL

Podemos diferenciar cuatro grandes tipos:

- **BD clave-valor**: Cada elemento está identificado por una llave única, lo que permite la recuperación de la información de forma muy rápida. Ejemplo Redis, DynamoDB. Páginas como Amazon o Best Buy utilizan esta implementación.

- **Orientadas a documentos**: Gestionan datos semi estructurados (documentos) Estos datos son almacenados en algún formato estándar como puede ser XML, JSON o BSON. Gran versatilidad. Ejemplos: MongoDB, CouchDB. Netflix utiliza esta tecnología.

3. Tipos BD NoSQL

- **Orientadas a columnas**: similara aclave/valor, pero la clave es la fila de una columna, junto con un timestam. Este tipo de bases de datos están pensadas para realizar consultas y agregaciones sobre grandes cantidades de datos. Cassandra, BigTable, Hadoop/HBase. Twiter o Adobe utilizan este modelo.

- **En grafo**: Utiliza un sistema de grafos para recorrer los datos y sus relaciones, que se puede extender a múltiples máquinas.. Ejemplo Neo4J, GraphBase, Virtuoso.

4. Mongo DB.

MongoDB es un sistema de BD NoSQL:

- multiplataforma.
- Orientado a documentos.
- Muy rápido (escrito en C++).
- De licencia libre.
- Disponible en la mayoría de S.O.

4. Mongo DB. Modelo de datos

MongoDB trabaja con el siguiente modelo:

- Base de datos: Es un contenedor de colecciones.
- Colección: Es un grupo de documentos MongoDB. Equivalente a tabla en BDR. Normalmente los documentos de una colección tienen mismo campo, pero no es obligatorio seguir un mismo esquema.
- documento: Un registro en MongoDB es un documento, estructura de datos compuesta de pares campo-valor. Documentos MongoDB son similares a objetos JSON. Equivale a una fila en DBR.
- Campo: equivale a la columnas en BDR. Cada campo tienen un valor.

Características del modelo:

- No soporta JOINs ni transacciones, para ello utiliza referencias a otros documentos.
- Tiene un lenguaje de consulta propio.
- Las operaciones atómicas se realizan en un solo documento y no soporta transacciones de multiples documentos.

4. Mongo DB. Modelo de datos

MongoDB trabaja con el siguiente modelo:

- Base de datos: Es un contenedor de colecciones.
- Colección: Es un grupo de documentos MongoDB. Equivalente a tabla en BDR. Normalmente los documentos de una colección tienen mismo campo, pero no es obligatorio seguir un mismo esquema.
- documento: Un registro en MongoDB es un documento, estructura de datos compuesta de pares campo-valor. Documentos MongoDB son similares a objetos JSON. Equivale a una fila en DBR.
- Campo: equivale a la columnas en BDR. Cada campo tienen un valor.

Características del modelo:

- No soporta JOINs ni transacciones, para ello utiliza referencias a otros documentos.
- Tiene un lenguaje de consulta propio.
- Las operaciones atómicas se realizan en un solo documento y no soporta transacciones de multiples documentos.

5. JSON

MongoDB utiliza **JSON** (Java Script Object Notation) basado en un subconjunto del Lenguaje de Programación JavaScript, Standard ECMA-262.

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, etc.

A nivel interno los documentos JSON no son almacenados como texto, sino en en BSON (Binary JSON) que es una representación en binario del propio JSON.

5. JSON

En JSON, se presentan de estas formas:

- objeto: es un 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.

```
{ "dinosaur": {"name": "Acrocanthosaurus", "id_period": 2, "height" :7, "weight": 1100, "length": 12}}
```

- Array: es una colección de valores. Comienza con [corchete izquierdo y termina con]corchete derecho. Los valores se separan por ,coma. NO tienen por qué tener los mismos pares clave/valor.

```
{ "dinosaur": [
 {"name": "Acrocanthosaurus", "id_period": 2, "height" :7, "weight": 1100},
 {"name": "Albertosaurus", "height" :1, "weight": 400, "length": 4}
   ]}
```

- Valores: podrán ser cadenas de caracters (entre “”), un número, un booleano (true o false), null, un array, u otro objeto.

```
{ "dinosaur": {
    "name": "Tyrannosaurus Rex",
    "id_period": 2,
    "height" :7,
    "weight": 1100,
    "length": 12,
    "digging": true,
    "excavations": ["Tendaguru Formation", "Maevarano Formation"],
    "genes": {"location": "Tooth and Claw Hardness", "percentage": 50}
    }
}
```

5. JSON: relaciones entre documentos en MongoDB

En mongoDB los documentos se relacionan con otros documentos de dos formas:

- Referencias manuales: mediante la clave “_id”. Es el equivalente a una clave ajena en BD SQL:


```
[
{"_id":1,"name":"Acrocanthosaurus","period":2,"height":7,"weight":1100,"length":12,"excavations":[2,4]},
{"_id":2,"name":"Albertosaurus","period":3,"height":1,"weight":400,"length":4,"excavations":[6,5]},
```



```
[
{"_id":1,"name":"Tendaguru Formation"},
{"_id":2,"name":"Maevarano Formation"},
{"_id":3,"name":"Bahariya Formation"},
{"_id":4,"name":"Chenini Formation"},
{"_id":5,"name":"Tegama Beds"},
{"_id":6,"name":"Iren Dabasu Formation"},
```


- DBRefs: son referencias que aparte de _id, incluyen el nombre de la colección y opcionalmente la BD.
- En ambos casos es necesaria una segunda búsqueda para obtener la información de los documentos referenciados.

Práctica

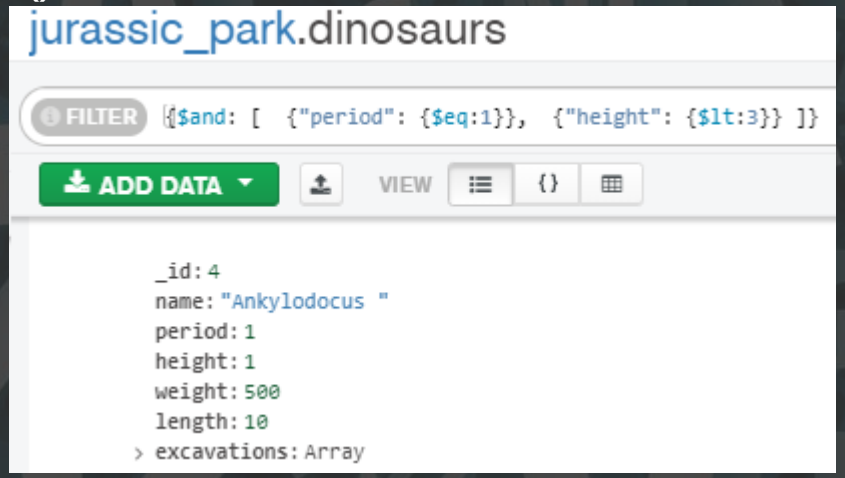
Práctica 1: Instalación de MongoDB y Compass

6. Operaciones y consultas con MongoDB

Desde la consola de mongoDB se pueden utilizar comandos para manejar la BD. Si bien no veremos comandos aquí, ya que lo habitual es utilizar algún cliente gráfico como Compass o Robo 3T, sí veremos los selectores que podemos utilizar con find() para realizar consultas. Estos selectores se pueden utilizar también con los clientes gráficos.

Para las consultas se utiliza la orden db.nombre_coleccíon.find(). Y dentro de find utilizar los siguientes selectores:

- Comparación: $eq, $gt, $gte, $in, $lt, $lte, $ne, $nin.
- Lógicos: $and, $not, $nor, $or
- Elemento: $exists, $type
  (comprueba la existencia o tipo de un campo).
- Evaluación: $regex, $text, $expr
  (para expresiones agregadas (ej: calve > cave)).



A parte de los operadores de consulta existen de actualización, borrado, agregación, y pipeline que no veremos.

Los selectores de consultas pueden consultarse en:
https://docs.mongodb.com/manual/reference/operator/query/

6. Mongo DB con Java: conexión

En java se utiliza la clase **MongoClient** para conectar a una BD, y a partir de ella podemos sacar la BD en un objeto **MongoDatabase** con el método .**getDatabase**(), y de esta cualquier colección a un objeto **MongoCollection<Document>** con **getCollection**().

Document es una implementación de las interfaces:

- Map<String, Object>: se podrá acceder al contenido por clave/valor
- Serializable: se podrá iterar
- Bson : se almacenará en formato BSON.

```
MongoClient mongoClient = MongoClients.create();
MongoDatabase database = mongoClient.getDatabase("test");
MongoCollection<Document> collection = database.getCollection("restaurants");
```

6. Mongo DB con Java: consultar colecciones

- EL método **find**() de una colección devuelve un FindIterable (hereda de Iterable). Para obtener las claves y valores de una colección utilizamos el métodos:
  - **.find**().**into**() que se le pasa el objeto a devolver
  - **.find**().**iterator**() que devuelve un iterador.
  - **.find().map():** mapea el contenido a un iterable haciendo un casting de los objetos.
  - **.find().first():** devuelve el primer objeto encontrado.

- Para obtener una clave en concreto de un documento, accedemos a ella por nombre, utilizando los métodos:
  - **getXXX**(“clave”) si conocemos el tipo de dato.
  - **get**(“clave”) con el que no se especifica el tipo de dato y devuelve un objeto.
  - En todas ellas se puede poner un valor por defecto si no se encuentra la clave.
  - Si se desea obtener por ejemplo un array se hace un casting a una lista:

```
List<Document> list = (List<Document>)str.get(“clave");
```

- El contenido de un documento entero puede visualizarse en formateado como texto con **toString**() o en formato JSON con **toJson**().

La documentación puede encontrase:
https://mongodb.github.io/mongo-java-driver/4.2/driver/tutorials/perform-read-operations/

6. Mongo DB con Java: consultar colecciones

```java
//Consultar todos los datos de una colección con una lista
List<Document> consulta = col_Dinosaurrios.find().into(new ArrayList<Document>());
for (Document d: consulta)
    System.out.println(d.toString());
```

```
Document{{_id=1, name=Acrocanthosaurus, period=2, height=7, weight=1100, length=12, excavations=[29, 24]}}
Document{{_id=2, name=Albertosaurus, period=3, height=1, weight=400, length=4, excavations=[36, 36]}}
Document{{_id=3, name=Allosaurus, period=2, height=9, weight=2600, length=10, excavations=[17, 15]}}
Document{{_id=4, name=Ankylodocus , period=1, height=1, weight=500, length=10, excavations=[34, 21]}}
```

```java
//Consultar ciertas claves
for (Document d: consulta)
    System.out.println("Dinosaurio: "+d.getString("name")
                       +", peso: "+d.getInteger("weight",0) //valdrá 0 si no encuentra la clave weight
                       +", excavaciones:"+ ((List<Integer>)d.get("excavations")).toString());
```

```
Dinosaurio: Acrocanthosaurus, peso: 1100, excavaciones:[2, 4]
Dinosaurio: Albertosaurus, peso: 400, excavaciones:[6, 5]
Dinosaurio: Allosaurus, peso: 2600, excavaciones:[17, 15]
Dinosaurio: Ankylodocus , peso: 500, excavaciones:[34, 21]
```

```java
//Consultar todos los datos de una colección con un iterable
MongoCursor<Document> cursor = col_Dinosaurrios.find().iterator();
while (cursor.hasNext()) {
    Document d = cursor.next();
    System.out.println(d.toJson());
}
```

```
{"_id": 1, "name": "Acrocanthosaurus", "period": 2, "height": 7, "weight": 1100, "length": 12, "excavations": [29, 24]}
{"_id": 2, "name": "Albertosaurus", "period": 3, "height": 1, "weight": 400, "length": 4, "excavations": [36, 36]}
{"_id": 3, "name": "Allosaurus", "period": 2, "height": 9, "weight": 2600, "length": 10, "excavations": [17, 15]}
{"_id": 4, "name": "Ankylodocus ", "period": 1, "height": 1, "weight": 500, "length": 10, "excavations": [34, 21]}
```

Ejemplo de un programa se consulta sencillo:
https://www.mongodb.com/docs/drivers/java/sync/current/quick-start/

6. Mongo DB con Java: filtrar documentos

- Para filtrar documentos es posible añadir operadores dentro del método .find(). El resultado se puede pasar a una lista, o a un iterador. Los diferentes operadores son:
  - Comparación: eq, gt, gte, in, lt, lte, ne, nin-
  - Lógicos: and, not, nor, or.
  - Elemento: exists, type.
  - Evaluación: expr (para expresiones agregadas (ej: calve > cave)), regex, text.
  - Array: all (coinciden todos los elementos), elemMatch (algunos coinciden con todos los pasados), size.

```
List<Document> consulta = col_Dinosaurrios.find(and(eq("period",1),lt("height",3)))
        .into(new ArrayList<Document>());;
for (Document d: consulta)
    System.out.println(d.toJson());
```

```
{"_id": 4, "name": "Ankylodocus ", "period": 1, "height": 1, "weight": 500, "length": 10, "excavations": [34, 21]}
{"_id": 9, "name": "Brachiosaurus", "period": 1, "height": 1, "weight": 3000, "length": 12, "excavations": [8, 39]}
{"_id": 51, "name": "Proceratosaurus", "period": 1, "height": 2, "weight": 800, "length": 10, "excavations": [12, 27]}
{"_id": 66, "name": "Tsintaosaurus", "period": 1, "height": 2, "weight": 3000, "length": 5, "excavations": [43, 33]}
```

- La utilización de estos operadores requiere importar la clase filters de forma estática:

```
import static com.mongodb.client.model.Filters.*;
```

- En la documentación de mongoBD puede encontrar todos los operadores y su descripción: https://docs.mongodb.com/manual/reference/operator/query/

6. Mongo DB con Java: operaciones sobre el cursor

- El método .find() devuelve un FindIterable (hereda de Iterable). Sobre este se pueden utilizar los siguientes métodos para realizar diferentes operaciones:
  - .count(), .limit(), size(), .skip(), .toArray(): son autoexplicativos.
  - .sort() que permite los métodos .ascending/.descending(List<String> fieldNames) para ordenar los documentos y orderby() para combinar los anteriores. Requiere importar la clase Sorts de forma estática:

```
import static com.mongodb.client.model.Sorts.*;
```

```
List<Document> consulta = col_Dinosaurrios.find(and(eq("period",1),lt("height",3)))
        .sort(descending("name")).limit(2)
        .into(new ArrayList<Document>());;
for (Document d: consulta)
    System.out.println(d.toJson());
```

```
{"_id": 66, "name": "Tsintaosaurus", "period": 1, "height": 2, "weight": 3000, "length": 5, "excavations": [43, 33]}
{"_id": 51, "name": "Proceratosaurus", "period": 1, "height": 2, "weight": 800, "length": 10, "excavations": [12, 27]}
```

La documentación puede encontrase:
https://mongodb.github.io/mongo-java-driver/4.1/apidocs/mongodb-driver-core/com/mongodb/client/model/Sorts.html

6. Mongo DB con Java: inserción, actualización y borrado

Para la inserción de documentos se utilizan los métodos:

- .**insertOne**(): inserta un documento
- .**insertMany**(): inserta una lista de documentos.

Para la actualización de documentos se utilizan los métodos:

- .**updateOne**(): actualiza un documento
- .**updateMany**(): actualiza una lista de documentos.

  Ambas devuelven un **UpdateResult** con métodos para saber los documentos encontrados y actualizados.

  Requiere importar la clase Sorts de forma estática:

```
import static com.mongodb.client.model.Updates.*;
```

Para el borrado de documentos se utilizan los métodos:

- .**deleteOne**(): borra un documento
- .**deleteMany**(): borra una lista de documentos.

  Ambas devuelven un **DeleteResult** con métodos para saber los documentos borrados

```
//Inserción
Document d1 = new Document();
d1.put("name","Zepalosaurus");
d1.put("period",2);
d1.append("height",7);
d1.append("weight",1100);
d1.append("length",12);
List<Integer> ex = new ArrayList<Integer>(Arrays.asList(2,4));
d1.append("excavations",ex );

col_Dinosaurrios.insertOne(d1);

//Actualización
UpdateResult up = col_Dinosaurrios.updateOne(eq("name","Zepalosaurus"),
                                             set("period",3));
System.out.println("Encontrados: "+up.getMatchedCount()+
        " elementos y "+up.getModifiedCount()+" actualizados.");

//Borrado
DeleteResult del = col_Dinosaurrios.deleteOne(eq("name","Zepalosaurus"));
System.out.println("Borrados: "+del.getDeletedCount());
```

La documentación puede encontrase: https://mongodb.github.io/mongo-java-driver/4.2/driver/tutorials/perform-write-operations/

Práctica

Práctica 2: Creación de un proyecto y primeras consultas

6. Mongo DB con Java: proyecciones y agregaciones

- El método .find() devuelve documentos con todos sus campos. Para sacar solamente algunos campos del documento se utiliza una proyección.

- Requiere importar la clase Projection de forma estática:

```
import static com.mongodb.client.model.Projections.*;
```

https://www.mongodb.com/docs/drivers/java/sync/current/fundamentals/builders/projections/