Storage: Implementación

Índice

Mongo DB

Mongo DB

MongoDB es un sistema de base de datos NoSQL orientado a documentos open source.

Es un sistema de gestión de base de datos documental diseñado para la persistencia de datos de alto rendimiento y alta disponibilidad.

Almacena datos en una representación binaria llamada **BSON (Binary JSON).** La codificación BSON extiende la popular representación JSON (JavaScript Object Notation) para incluir tipos adicionales como int, long, date, coma flotante y decimal128.
Los documentos BSON contienen uno o más campos, y cada campo contiene un valor de un tipo de datos específico, que incluye matrices, datos binarios y subdocumentos.

Los documentos MongoDB BSON están estrechamente alineados con la estructura de los **objetos** en el lenguaje de programación.

Mongo DB vs. Base de Datos Relacional

En vez de usar Tablas y Filas, como en las BD relacionales, se usan **Colecciones y Documentos**.

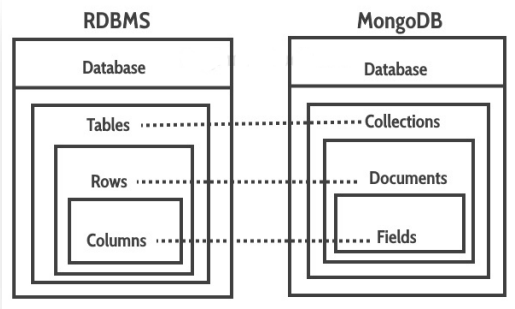
Una Base de Datos es un espacio donde se almacena una colección.

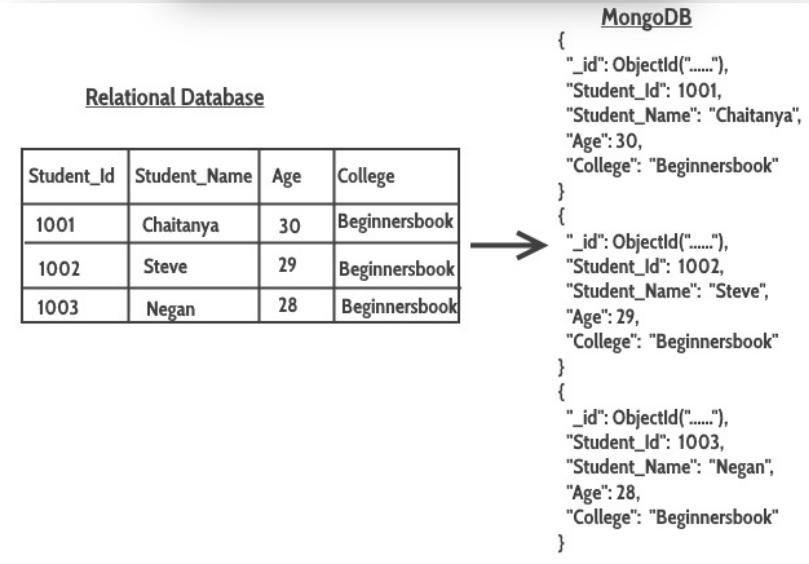
Una **colección (Collection)** es como si fuera una tabla y contiene documentos que son el equivalente a los registros.

Un **documento (Document)** es mapeado por un objeto en la aplicación.

Un documento tiene **campos (Fields)**, que son un par de Key: Value.

Contrario a la una BD relacional, un documento de una colección puede tener 4 campos mientras que otro puede tener 6.





Spring Data + Mongo DB

Anotaciones Spring Mongo Data

@Document: Usada en clases, le indica al MongoDriver que puede tomar objetos de esa clase como Mongo Documents en una Colección particular. El nombre de la colección se deduce del nombre de la clase, pero si tienen diferentes nombres es necesario agregar el nombre de la collection que se va a asociar. Es similar a la anotación @Entity utilizada en JPA.

@Id: Todo Document en una Collection Mongo debe tener un campo de id. Es único y es indexado. Es similar a una clave primaria en la BD relacionales.

@Field: Instruye al framework de qué manera persistir los atributos de nuestras clases en Mongo. Podemos utilizar el atributo “name” para especificar el nombre que tiene ese campo en el Document de MongoDB.

@Transient: Excluir un atributo para que no sea persistido.

```
@Document(collection="airplanes")
public class Aircraft {
    private String id;
    private String model;
    private int nbSeats;
    …
}
```

```
@Document
public class Aircraft {
    @Id private String id;
    private String model;
    @Field(name="seats") private int nbSeats;
    …
}
```

```
@Document
public class Aircraft {
    @Id private String id;
    private String model;
    @Transient private int nbSeats;
    …
}
```

…más anotaciones en Spring Mongo Data

@Indexed: Si queremos ejecutar queries con filtros más rápidamente. Si por ejemplo vamos a realizar muchas queries sobre el field model es bueno indexarlo. Puede especificarse la dirección del index, y si es único.

@DbRef: Actúa parecido a un join de una BD relacional. Se une un Document a otro de una Collection diferente.

```
@Document
public class Aircraft {
    @Id private String id;
    @Indexed(direction=IndexDirection.ASCENDING, unique=false)
    private String model;
    private int nbSeats;
    …
}
```

```
@Document
public class Manufacturer{
    @Id private String id;
    private String name;
    …
}
```

```
@Document
public class Aircraft {
    @Id private String id;
    private String model;
    @DbRef
    private Manufacturer man;
}
```

Creando una BD

Antes de comenzar

Instalar MongoDB
https://docs.mongodb.com/manual/installation/
la forma de instalación y ubicación dependerá del sistema operativo.

Instalar Robo 3T
https://robomongo.org/download una interfaz gráfica para Mongo DB.

Creando la BD Mongo

1) Una vez instalada MongoDB, nos posicionamos en el directorio:

```
C:\Program Files\MongoDB\Server\4.2\bin
```

2) Ejecutamos el comando **mongo** para entrar en la shell de MongoDB donde podremos crear la BD.

3) Creamos la BD usando el comando **use dbname.**

```
> use mongoexample
switched to db mongoexample
>
```

4) Creamos una Collection llamada Books con el comando

```
> db.createCollection("books");
```

5) Insertamos un **documento** en la colección

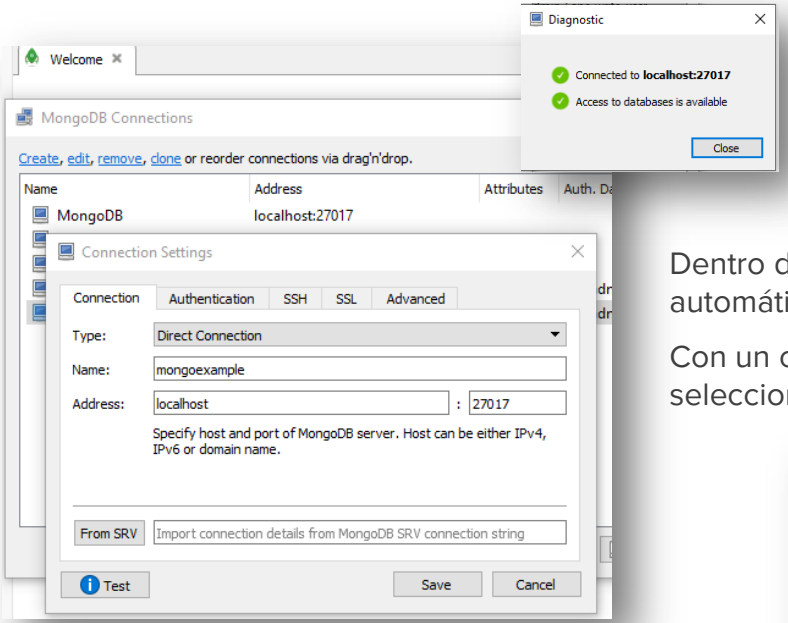
```
> db.books.insert({ author:"Jorge Luis Borges", book:"Poesía Completa"})
WriteResult({ "nInserted": 1 })
>
```

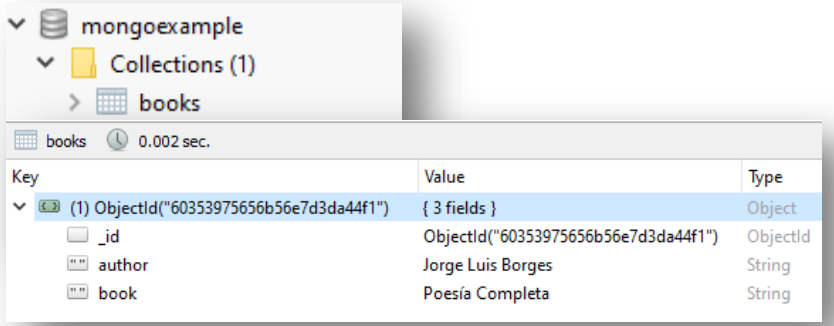
6) Realizamos una **query** que nos muestre **todos** los **documentos** de la colección books.

```
> db.books.find({})
{
    "_id" : ObjectId("60353975656b56e7d3da44f1"),
    "author" : "Jorge Luis Borges",
    "book" : "Poesía Completa"
}
>
```

Conectandonos a Mongo desde ROBO 3T

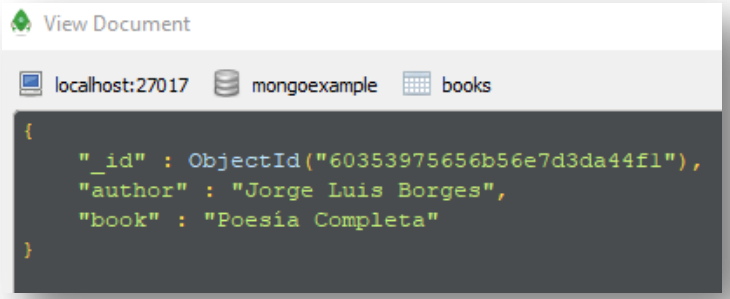
10) Ya podemos conectarnos a nuestra base de datos desde ROBO 3T para seguir creando documentos.





Dentro de **Collections** se creó **books**, y al hacer doble click sobre books automáticamente nos muestra los **Documents** que hay.

Con un click derecho podemos **insertar** un Document en formato json. O seleccionando el Documento podemos **editarlo**.



Creación del Proyecto

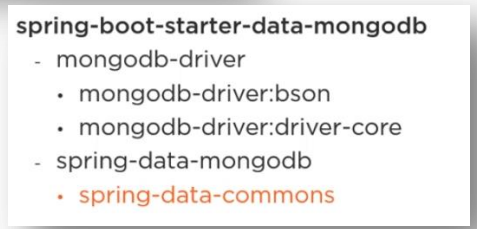
Dependencias

Con SpringBoot Initializer desde IntelliJ IDEA, podemos crear un proyecto Spring con las siguientes dependencias en el **pom.xml**:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-mongodb</artifactId>
</dependency>
```

spring-boot-starter-data-mongodb
Tiene todo lo necesario para crear
Un proyecto Springboot con Mongo



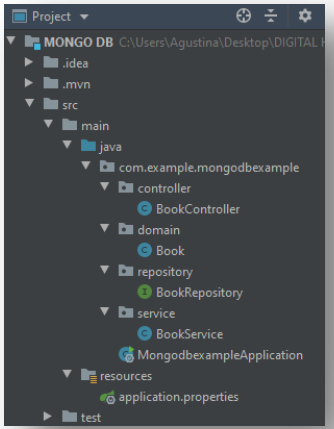
Propiedades de Conexión

En resources creamos un file con el nombre **application.properties** donde agregaremos la configuración.

```
#mongodb
spring.data.mongodb.host=localhost
spring.data.mongodb.port=27017
spring.data.mongodb.database=mongoexample
```

Clases y estructura del proyecto

- La clase **Book** que va a mapear a la collection books.
- El **BookRepository** que extiende de MongoRepository
- El **BookService** con el método findAllBooks.
- El **Controller**.




```java
@Document(collection = "books")
public class Book {
    @Id
    private String id;
    private String author;
    @Field(name="book")
    private String bookTitle;
}
```

```java
@RestController
@RequestMapping(value = "/mongoexample")
public class BookController {

    private final BookService bookService;

    public BookController(BookService bookService) {
        this.bookService = bookService;
    }

    @GetMapping(value = "/books")
    public List<Book> getAllBooks() {
        return bookService.findAllBooks();
    }
}
```

```java
@Service
public class BookService {

    private final BookRepository bookRepository;

    public BookService(BookRepository bookRepository) {
        this.bookRepository = bookRepository;
    }

    public List<Book> findAllBooks() {
        return bookRepository.findAll();
    }
}
```

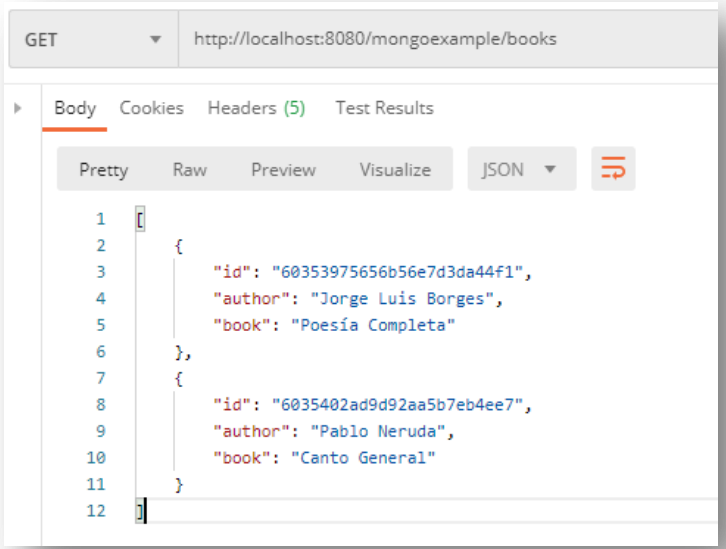
```java
@Repository
public interface BookRepository extends MongoRepository<Book, String> {
}
```

Funcionamiento

Correr la aplicación con **mvn spring-boot:run**
Desde el navegador o desde Postman realizar un GET a la url:

http://localhost:8080/mongoexample/books

Realizará una request que producirá una query a nuestra BD de Mongo local y nos devolversa una response con todos los documents de la collection Books.



¡Gracias! ☺