# Mongo DB

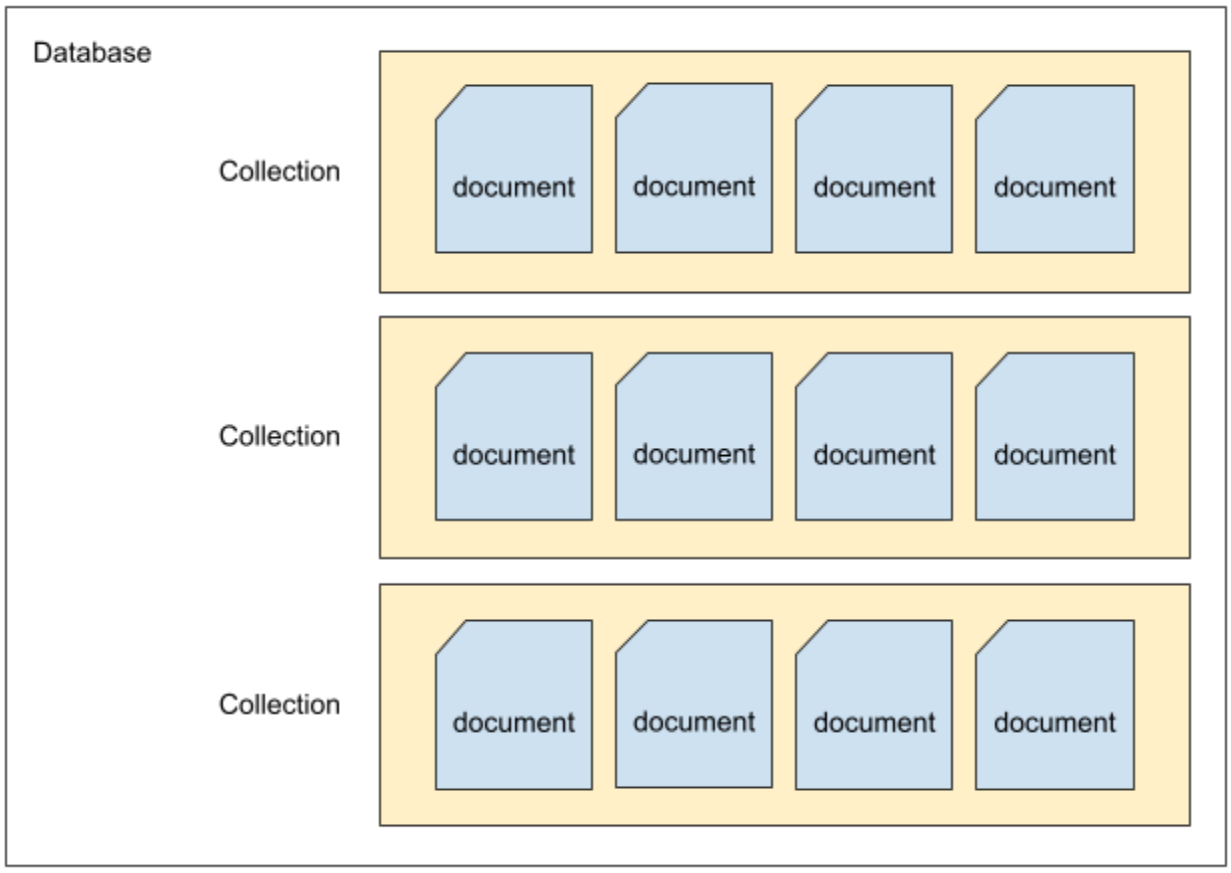


## معرفی

نوعی دیتابیس NoSQl است یعنی ساختار ذخیره سازی داده در آن به شکل سطر و ستون نیست بلکه به شکل کلید-مقدار است و شبیه به json است اما به آن Bson یعنی Binary Json Object Notation می‌گوییم.
اگر قصد داریم با یک محیط گرافیکی با Mongo DB کار کنیم می‌توانیم از Mongo DB compass استفاده کنیم.

# اصطلاحات

document : مجموعه‌ای از کلید-مقدارها است که یک object را مشخص می‌کند.
collection : گروهی از document ها collection را ایجاد می‌کند.
database : گروهی از collection ها با هم database را ایجاد می‌کنند.




# دیتا تایپ ها

| Type | Description |
|---|---|
| string | put it between “ “ or ‘ ‘ |
| integer | any number without floating point |
| double | number with floating point |
| boolean | values could be true or false |

| | |
|---|---|
| date object | define date. for example “new Date()” object will return current date. |
| null | |
| array | it is an array. one field in Mongo DB can store multiple value |
| document | we can create a document inside another document (nested document). |

# دستورات

بعد از نصب Mongo DB می‌توانیم Mongosh که یک Command line interface برای Mongo DB است را نصب کنیم و با استفاده از کامندهای زیر با Mongo DB ارتباط برقرار کنیم.

| Category | Command | Description |
|---|---|---|
| Database | cls | clear screen |
| | show dbs | show list of databases |
| | use <db name> | select a DB to work, and if the db name does not exist it will create it. |
| | db.dropDatabase() | drop current database |
| Collection | show collections | show list of collections inside a database |
| | db.<collection name>.drop() | drop a collection |
| | db.createCollection(“ <collection name> ”) | create a collection inside of selected db |
| Create | db.<collection name>.insertOne({name:”sina”, age: 26}) | “insert One” method creates a new document inside the collection.<br><br>If a collection with a defined name does not exist, it creates it too. |
| | db.<collection name>.insertMany([ {name:”sina”, age: 26}, | Insert more than one document to collection. |

| | | |
|---|---|---|
| | {name:”ali”, age: 30}<br>]) | consider that we pass an array to method |
| Read | db.<collection name>.find() | gives a list of documents inside a collection. |
| | db.<collection name>.find(<br>{query},<br>{projection}<br>) | we can pass “query” and “projection” to find method.<br><br>example: {name:”sina”}, {name:true, age:true}<br>description: return students whose names is “sina” and return just “name” and “age” fields. |
| | db.<collection name>.find().sort({<br>name: 1<br>}) | sort all returning results of the “find” method.<br>This method get an object as input for example in this sample we are saying sort by name.<br><br>1 = ASC<br>-1 = DESC |
| | db.<collection name>.find().limit(2) | defined how many of the results should be returned. |
| Update | db.<collection name>.updateOne(<br>{filter},<br>{update}<br>) | update one document. it gets two parameters, the filter define which object should be update and update define which fields should update.<br><br>in “update” object we should define operators:<br>$set: add or replace the value of field<br>$unset: remove field<br><br>example : {name: sina}, {$set:{age: 25}}<br>description: update documents whose name is sina and set its age to 25 |

| | | |
|---|---|---|
| Delete | db.<collection name>.deleteOne({<br>name: “sina”<br>}) | will delete one collection based on passed conditions. |
| | db.<collection name>.deleteMany({<br>name: “sina”<br>}) | delete more than one. |

نکته : وقتی یک database میسازیم اگر دستور show dbs رو بزنیم دیتابیسی که ساختیم را به ما نشان نمی‌دهد به این خاطر که آن دیتابیس خالی است.

نکته : در Mongo DB امکان method chaining داریم مثلا متد find فراخوانی می‌شود و سپس متد sort فراخوانی می‌شود و سپس متد limit.

نکته: وقتی یک collection میسازیم می‌توانیم محدودیت‌هایی نیز مشخص کنیم. مثلا بگوییم در یک collection چند document می‌توانیم داشته باشیم یا حجم collection چند بایت باشد.

# اپراتور ها

در Mongo DB مجموعه‌ای از اپراتورها را داریم که به ما کمک می‌کنند امکاناتی را داشته باشیم. این اپراتورها با علامت $ شروع می‌شوند.

## Comparison operators

### ne

معادل با not equal است.
در مثال زیر گفته‌ایم که document هایی را برگردان که نام آنها sina نیست.

```
db.<collection name>.find(
	{name: {$ne:”sina”}}
)
```

### lt و gt

معادل با Less than و greater than است.
در مثال زیر گفته‌ایم که document هایی که سن آنها کمتر از 20 را برگردان.

```
db.<collection name>.find(
	{age: {$lt:20}}
)
```

## lte و gte

معادل با less than or equal و greater than or equal است.
در مثال زیر گفته‌ایم که document هایی که سن آنها کمتر یا مساوی از 20 را برگردان.

```
db.<collection name>.find(
	{age: {$lte:20}}
)
```

## in

اگر مقدار داده شده با مقدار تعیین شده در in با هم match باشند در نظر گرفته می‌شود.
در مثال زیر گفته شده هرکسی که نامش در این آرایه باشد.

```
db.<collection name>.find(
	{
		name: {$in:[“sina”, “ali”]}
	}
)
```

## nin

معادل با not in است
در مثال زیر گفته شده هرکسی که نامش در این آرایه نباشد.

```
db.<collection name>.find(
	{
		name: {$nin:[“sina”, “ali”]}
	}
)
```

نکته : می‌توانیم همزمان از چند اپراتور استفاده کنیم. مثلا :

```
db.<collection name>.find(
	{
		age: {
			$lte: 20,
			$gte: 30
		}
	}
)
```

# Logical operators

## and

یک آرایه می‌گیرد و بررسی می‌کند که همه شرط‌های داخل آرایه برقرار باشند.

```
db.<collection name>.find(
	{
		$and: [
			{name:”sina”},
			{age: {$gt: 22}}
		]
	}
)
```

## not

عمل not را انجام میدهد. مثلا در کد زیر گفته شده سن بزرگتر از 22 نباشد.

```
db.<collection name>.find(
	{
		age: {$not:{$gt: 22}}
	}
)
```

## nor

مانند شرط and است یا این تفاوت که به جای اینکه همه شرط ها برقرار باشند، همه شرط ها باید برقرار نباشند.
در مثلا زیر گفته ایم که نه نامش "sina" باشد و نه سنش بزرگتر از 22 باشد.

```
db.<collection name>.find(
	{
		$nor: [
			{name:”sina”},
			{age: {$gt: 22}}
		]
	}
)
```

## or

یک آرایه می‌گیرد و بررسی می‌کند که حداقل یکی از شرط‌های داخل آرایه برقرار باشند.

```
db.<collection name>.find(
	{
		$or: [
			{name:”sina”},
			{age: {$gt: 22}}
```

```
            ]
        }
)
```

## Element operators

### exists

بررسی می‌کند که یک فیلد وجود دارد یا خیر. مثلا در کوئری زیر گفته ایم اگر فیلد name وجود نداشت آپدیت شود :

```
db.<collection name>.updateMany(
        {name: {$exists:false}},
        {$set: {name: “something”}}
)
```

# ایجاد ارتباط بین Document ها

در MongoDB دو راه برای ایجاد ارتباط بین Document ها وجود دارد :

- روش Referencing : در این روش دو document جداگانه داریم و اگر document اول بخواهد به document دوم ارتباط داشته باشد کافی است id آن را در داخل خودش داشته باشد.
- روش Embedded : در این روش دیگر دو document جداگانه نداریم بلکه یک document داریم که در خودش document دیگری را دارد.

نکته : هر دو روش بالا قابل استفاده هستند و هر کدام از این روش‌ها مزایا و معایب خود را دارند که باید با توجه به نیاز انتخاب کنیم ولی به شکل کلی اگر document ها بزرگ هستند باید به شکل مجزا ذخیره شده و از روش referencing استفاده کنیم ولی اگر نوع استفاده ما از document ها به این شکل است که با فراخوانی یک document نیاز داریم document مرتبط با آن را هم لود کنیم روش embedded بهتر است چرا که در mongo db امکان join نداریم و برای لود کردن دیتا باید کوئری مجزا بزنیم.

# استفاده از Mongodb در Spring Boot

## اضافه کردن وابستگی

در هنگام ساخت پروژه وابستگی Spring Data MongoDB را به پروژه اضافه می‌کنیم.

## اتصال به MongoDB

متغیرهای زیر را در پروژه مقداردهی می‌کنیم.

```
spring.data.mongodb.authentication-database=admin
spring.data.mongodb.username=root
spring.data.mongodb.password=root
spring.data.mongodb.host=root
spring.data.mongodb.port=root
spring.data.mongodb.database=root
spring.data.mongodb.auto-index-creation=true
```

در mongoDB یک دیتابیس به نام admin داریم که user های سیستم را در خودش نگه می‌دارد. متغیر authentication-database مشخص می‌کند اینکه اطلاعات کاربر را بررسی کند باید به دیتابیس admin متصل شود.

نکته : متغیر auto-index-creation این امکان را می‌دهد که بتوانیم در داخل کلاس جاوای خود از انوتیشن Indexed استفاده کنیم و در سطح MongoDB آن Index ساخته شود.

## مدل کردن داده و استفاده از Annotation

برای مدل کردن داده یک کلاس جاوایی با فیلد های متناظر می‌سازیم :

```
@Document
public class Student {
	@Id
	private String id;

	private String name;

	private String lastName;

	@Indexed(unique=true)
	private String email;

	private LocalDate birthDate;
}
```

نکته : برای اینکه مشخص کنیم این کلاس معادل با یک document در mongoDB است از انوتیشن Document استفاده می‌کنیم. معادل انوتیشن Entity در دیتابیس‌های رابطه‌ای است.

نکته : هر document یک id دارد پس با انوتیشن Id آن را مشخص می‌کنیم.

نکته : انوتیشن Indexed روی یک فیلد ایندکس تعریف می‌کند که باعث می‌شود اجرای کوئری‌ها روی آن فیلد سرعت بیشتری داشته باشد همچنین این امکان را می‌دهد که بگوییم آن فیلد در collection ما باید unique باشد.

نکته : نام collection از روی نام کلاس انتخاب می‌شود مثلا در این مثال نام collection ما در Mongo DB برابر با student خواهد بود. اگر نام collection ما با نام کلاس متفاوت است می‌توانم در انوتیشن Document نام دلخواه خود را مشخص کنیم.

## ساخت repository

برای ساخت repository و کار با دیتابیس یک interface میسازیم که از اینترفیس MongRepository ارث بری می‌کند.

```
public interface StudentRepository extends MongoRepository<Student, String> {

}
```

در این interface متدهای آماده برای کار با MongoDB از قبل قرار دارد.

## اجرای کوئری دلخواه

### استفاده از کلاس Query

اگر بخواهیم می‌توانیم با استفاده از کلاس Query یک کوئری دلخواه را اجرا کنیم مثلا :

```
Query query = new Query();
query.addCriteria(
        Criteria.where(“email”).is(“myemail.com”)
);
var result = mongoTemplate.find(query, Student.class);
```

نکته : کلاس MongoTemplate به شکل پیشفرض در کانتکس برنامه وجود دارد و کافی است فقط آن را inject کنیم.

### استفاده از Method Name

می‌توانیم نام متد در اینترفیس مربوط به repository را نوعی انتخاب کنیم که خودش کوئری را ایجاد کند. مثلا :

```
var result = repository.findStudentByEmail(“myEmail.com”);
```

### استفاده از MongoDB Query

می‌توانیم با استفاده از انوتیشن Query در repository کوئری دلخواه خود را بنویسیم :

```
@Query(“”my query is here“”)
void myQuery();
```