

Day - 02

```
db.inventory.find( [tags: { $exists: false }],  
                   [tags: 1, -id: false])
```

for composite attributes we use `concat.name`

```
41  
42 // $eq, $in, $gt (inventory)  
43 // quantity 30, 50  
44 db.inventory.find({quantity: {$in: [30, 50]}})  
45  
46 // Text (status)  
47
```

→ return rows it's quantity
30 or 50
✓

it acts same as or

we use `or` operator if the `or` between 2 different fields
not solved by an array

`$` in operator takes array not single value

Logical Query operator

Logical Query Operators	
<code>db.inventory.find({ \$Operator: [{ price: { \$ne: 1.99 } }, { price: { \$exists: true } }] })</code>	
Name	Description
<u>\$and</u>	Joins query clauses with a logical AND returns all documents that match the conditions of both clauses.
<u>\$not</u>	Inverts the effect of a query expression and returns documents that do not match the query expression.
<u>\$nor</u>	Joins query clauses with a logical NOR returns all documents that fail to match both clauses.
<u>\$or</u>	Joins query clauses with a logical OR returns all documents that match the conditions of either clause.

```
db.inventory.find ( { $and: [
```

but all the conditions you want in the and
#number of { } is the number of conditions

```
] }
```

```
db.inventory.find ( { $or: [
```

```
{ qty : { $gte : 25 } },
```

```
{ sal : { $in : [50, 20] } }
```

```
] }
```

db.products.find ({ product: { \$regex: "(?i)pc" } })

↳ ignore the case sensitive

db.inventory.find ({ tags: { \$size: 2 }, tags: { \$in: ['red', 'black'] } })

↳ tag size is equal 2

db.inventory.find ({ tags: { \$size: 2 }, tags: { \$all: ['red', 'black'] } })

↳ at least red & black available

if there're more components no problem but it should have both → if one of them only it will not give it

⇒ there's a question in the interview about this point

↳ find data has at least red and black

⇒ Comparison ⇒ between in & all ↳ size is additional in the all condition

```
db.books.find ( {
```



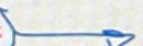
```
  tag : { $all : ["red", "blank"], size : 1 }
```

```
})
```

it will not give an error because it's not syntax error, it's logical error so it will return empty list

update & insert has 3 options one, many, just insert or update
↓
in the last versions

they keep it to help
conflicts if moving from old
version into new one

Update Operators	
Name	Description
<u>\$currentDate</u>	Sets the value of a field to current date, either as a Date or a Timestamp.
<u>\$inc</u>	Increments the value of the field by the specified amount.
<u>\$min</u>	Only updates the field if the specified value is less than the existing field value.
<u>\$max</u>	Only updates the field if the specified value is greater than the existing field value.
<u>\$mul</u>	Multiplies the value of the field by the specified amount.
<u>\$rename</u> 	Renames a field.
<u>\$set</u> 	Sets the value of a field in a document.
<u>\$setOnInsert</u>	Sets the value of a field if an update results in an insert of a document. Has no effect on update operations that modify existing documents.
<u>\$unset</u> 	Removes the specified field from a document.

min & max operators are different than SQL in NoSQL

```
db.updateOne({  
  $rename: {  
    "dep": "department"  
  }  
})
```

```
db.employee.updateMany({  
  $unset: {  
    "name": "" // will drop the column  
  }  
})
```

```
db.employee.updateMany({  
  $set: {  
    "code": "A"  
  }  
})
```



```

80 db.employee.updateOne({_id:10} ,
81 {
82   $set:{
83     "code" : "not",
84     name:"eman"
85   }
86 },
87 {
88   upsert:true
89 })

```

upsert works as insert or update

if not available

if available

we can add column to let me know if the record is inserted by the upsert or not

```

84   name:"eman"
85 },
86 $setOnInsert:{
87   dataSource:"new upsert"
88 }
89 },
90 {
91   upsert:true
92 })
93

```

by \$setOnInsert: {
data source

data Source is added just in the Case of insert

Schema validation

Constraint

Imagine that we have a field (column) called code and it has a null value

if you created a non NULL Constraint it will give an error

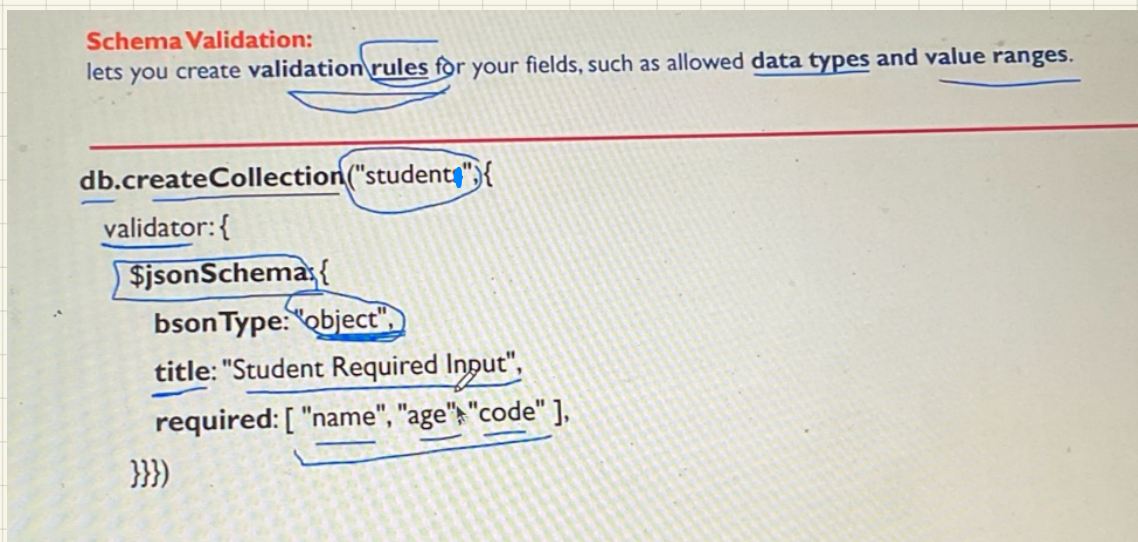
So you need to change the Null value to non null first then do the Constraint

vs.

Rule

what if you don't know what data was in the field and you want the data before know to stay the same but make a Constraint on the data coming in the future.

we can create a Rule



After this I should insert name, age, code for each insertion in student

properties: {

name: {

bsonType: "string",

description: "'name' must be a string and is required"

},

year: {

bsonType: "int",

minimum: 2017,

maximum: 3017,

description: "'year' must be an integer in [2017, 3017] and is required"

},

gpa: {

bsonType: ["double"],

description: "'gpa' must be a double if the field exists"

}

}