03

Consultando os dados

Find e *query* com navegação em documento

Já criamos uma coleção de alunos. Ela é composta de diversos elementos e podemos observar isso digitando `db.alunos.find()`. Assim, é mostrado tudo que segue um padrão vazio. Veremos todos os objetos, só que de uma maneira bastante ilegível. Para conseguirmos ler de uma forma mais clara, podemos digitar `db.alunos.find().pretty()` e serão mostrados os objetos de uma maneira mais legível:

Agora, aprenderemos que tipos de coisas trabalhar no MongoDB. Primeiro, veremos o `find`, que é como um *select* não filtrado de um banco de dados não relacional. As vezes, entretanto, queremos encontrar não tudo, mas algo com um padrão específico. Para isso, utilizamos o `db.alunos.find()` e nos parênteses passamos a referência do que queremos. O Mongo funciona no exemplo que damos para ele, imagine que buscamos tudo o que possua o nome "Felipe". No Editor, escreveremos um objeto de exemplo para passar, no Terminal, ao `find`. Essa maneira é a mais simples de encontrar chama-se *query example*. Vejamos como buscaríamos o "Felipe":

```
db.alunos.find(
    {
        nome : "Felipe"
    }
).pretty()
```

Ele irá encontrar o "Felipe" que buscamos:

```
> db.alunos.find(
... {
... nome : "Felipe"
... }
... ).pretty()
{
        "_id" : ObjectId("56cb0139b6d75ec12f75d3b6"),
        "nome" : "Felipe",
        "data_nascimento" : ISODate("1994-03-26T03:00:00Z"),
        "curso" : {
                "nome" : "Sistemas de informação"
        },
        "notas" : [
                10,
                9,
                4.5
        ],
```

Se tivéssemos mais de um objeto com o nome de "Felipe" ele traria todos os alunos chamados "Felipe". O `find` funciona, por padrão, passando por todos os objetos e procurando neles o campo que estamos buscando. Todos aqueles que possuírem a mesma referência são separados e são mostrados, isto é, devolvidos para nós. O `find` faz uma busca que por padrão é bastante simples, ele varre todos os documentos e procura todos os elementos que possuam as mesmas características que foram passadas, por exemplo, um "nome".

Vejamos, agora, algumas equivalências. Quando digitamos:

```
db.alunos.find({ nome : "Felipe"})
```

Isso é equivalente a dizer, no *SQL*,

```
SELECT * FROM alunos WHERE nome = "Felipe"
```

No dia a dia podemos buscar os alunos utilizando outras referências que não apenas os seus "nomes", por exemplo, pelas habilidades. Nesse caso, buscaremos a habilidade "inglês". Assim, buscaremos alunos cujas habilidades tenham o nome "inglês". Digitaremos, primeiro no Editor, e colaremos isso, depois, no Terminal. Usaremos o `find` , o `pretty` e especificaremos a habilidade em inglês para fazer essa busca. Observe:

```
db.alunos.find({ "habilidades.nome" : "inglês" }).pretty()
```

Teremos a seguinte resposta:

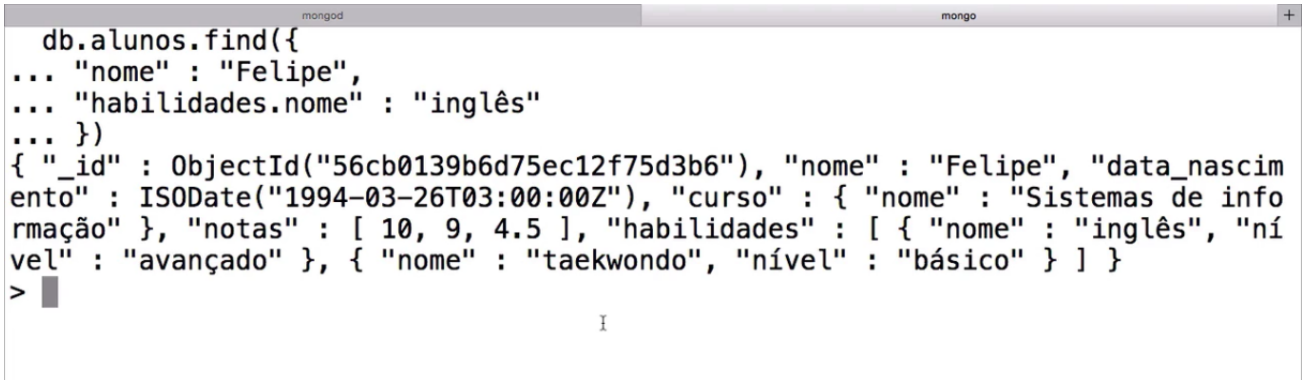
```
> db.alunos.find({ "habilidades.nome" : "inglês" }).pretty()
{
        "_id" : ObjectId("56cb0139b6d75ec12f75d3b6"),
        "nome" : "Felipe",
        "data_nascimento" : ISODate("1994-03-26T03:00:00Z"),
        "curso" : {
                "nome" : "Sistemas de informação"
        },
        "notas" : [
                10,
                9,
                4.5
        ],
```

Temos os três alunos cujas habilidades são inglês.

É simples realizar uma busca em um elemento interno da árvore e do documento. Fomos entrando nesses elementos através do `find` e da referência que passamos, a `"habilidades.nome"`. Podemos buscar, inclusive, além da habilidade também por um "nome" específico, por exemplo, os alunos chamados "Felipe" que fazem inglês. Para isso, podemos escrever o seguinte:

```
db.alunos.find({
    "nome" : "Felipe",
    "habilidades.nome" : "inglês"
})
```

Ele nos devolverá um objeto:



Conseguimos fazer uma busca usando um campo diferente do documento, simplesmente passando um exemplo, um valor, do que estamos buscando.

Seria um pouco diferente fazer essa mesma busca passando dois valores, no **SQL**. O que queremos é buscar todas as habilidade que sejam "inglês", para isso buscaremos em habilidades (`FROM habilidades as h`), aquelas que sejam inglês (`WHERE H.nome="inglês"`), em todos os alunos (`*`). Assim, queremos pegar a tabela habilidade e juntar com a tabela Alunos(`JOIN alunos as a ON a.id = h.aluno_id`). No caso, como queremos uma busca da habilidade inglês, mas também que o nome seja "Felipe" vamos, para isso, acrescentar `AND a.nome = "Felipe"` . Teremos o seguinte:

```
SELECT a.(*)
    FROM habilidades as h
    JOIN alunos as a ON a.id = h.aluno_id
    WHERE h.nome="inglês"
    AND a.nome = "Felipe";
```

Lembrando que não estamos em uma competição de digitação. Não é por que digitamos menos usando o `db.alunos.find` que ele é melhor. Em ambos os casos temos vantagens e também desvantagens. A *query* `db.alunos.find` é simples para trazer igualdades, ela traz o documento inteiro, por padrão, mesmo que nós queiramos um pedacinho. O `SELECT a.*` traz apenas um pedacinho, mesmo que quisessemos inteiro. É importante lembrar, entretanto, ambos fazem as mesmas coisas. Tudo depende do que precisamos e como usamos. Mas, repare que fica mais fácil de fazer a *query* `db.alunos.find` do que a *SQL*, pois esta parece mais complexa.

O objetivo foi, também, mostrar o `find` . Vamos trabalhar alguns tipos de *query* no **MongoDB** e vamos aprender diversos usos do `find` . Se entrarmos na documentação do **MongoDB Collection find** veremos que a quantidade de coisas que podemos fazer no `find` é gigante. Observe em [docs.mongodb.org/v3.0/reference/method/db.collection.find (http://docs.mongodb.org/v3.0/reference/method/db.collection.find)](http://docs.mongodb.org/v3.0/reference/method/db.collection.find).

Find com *or* e *in*

Imagine que queremos buscar alunos que estejam matriculados em cursos de tecnologia, como o curso de "Sistema de informação" e "Engenharia química". Poderíamos fazer uma *query* diferente para cada um desses curso.

Entretanto, isso acaba sendo um pouco limitado. Na prática, o que queremos é executar uma *query* onde tenhamos uma condição que seja **ou** um valor **ou** o outro valor. Se tentássemos em uma *query* colocar as opções "Sistema de Informação" e "Engenharia Química" teríamos o seguinte:

```
> db.alunos.find({
    "curso.nome" : "Sistemas de informação",
    "curso.nome" : "Engenharia Química"
    })
{ "_id" : ObjectId("56cb03e9b6d75ec12f75d3b8"), "nome" : "Alberto", "data_nascimento" : ISODate
```

Observe, nos é mostrado apenas as informações referentes ao aluno que cursa "Engenharia Química", porém, a primeira informação que passamos é, totalmente ignorada.

Estamos usando um *javascript* e, por isso, estamos sensíveis a suas regras. O dicionário de *javascript* é um conjunto de chave e valor. A chave é única e ela possui apenas um valor por vez. Quando colocamos em uma chave dois nomes, o segundo subscreve o primeiro, ou seja, é como se não tivéssemos escrito o primeiro. Pela linguagem, temos que fazer outro procedimento, podemos isolar os objetos, isto é, colocar eles em dicionários distintos. Observe:

```
{"curso.nome" : "Sistemas de informação"}
{"curso.nome" : "Engenharia Química"}
```

Quando temos diversos objetos e queremos agrupar eles podemos usar uma *Array* de *javascript*. Temos que falar que gostaríamos que qualquer uma das *query* que estão dentro da *Array*. Para isso, usamos o `or` para dizer, "**ou**" uma coisa "**ou**" a outra:

```
db.alunos.find({
    $or : [
        {"curso.nome" : "Sistemas de informação"},
        {"curso.nome" : "Engenharia Química"}
    ]
})
```

Copiando isso do editor e colando em nosso Terminal, veremos que esse processo é sucesso!

```
  db.alunos.find({
... $or : [
... {"curso.nome" : "Sistemas de informação"},
... {"curso.nome" : "Engenharia Química"}
... ]
... })
{ "_id" : ObjectId("56cb0139b6d75ec12f75d3b6"), "nome" : "Felipe", "data_nascim
ento" : ISODate("1994-03-26T03:00:00Z"), "curso" : { "nome" : "Sistemas de info
rmação" }, "notas" : [ 10, 9, 4.5 ], "habilidades" : [ { "nome" : "inglês", "ní
vel" : "avançado" }, { "nome" : "taekwondo", "nível" : "básico" } ] }
{ "_id" : ObjectId("56cb03e9b6d75ec12f75d3b8"), "nome" : "Alberto", "data_nasci
mento" : ISODate("1972-10-30T03:00:00Z"), "curso" : { "nome" : "Engenharia Quím
ica" }, "habilidades" : [ { "nome" : "inglês", "nível" : "intermediário" } ] }
>
```

E, agora, se quiséssemos buscar apenas as alunas "Danielas" que estão nesses cursos? Temos duas maneiras de tentar alterar esse código. Podemos introduzir uma terceira condição com o nome "Daniela" (`{"nome" : Daniela}`), logo abaixo dos cursos.

Mas isso não funcionará pois o `or` faz com que qualquer uma das opções sejam verdadeiras. Portanto, não faz sentido colocarmos o nome "Daniela" junto com os nomes dos cursos. Na verdade, o que queremos dizer é que buscamos qualquer pessoa que esteja em um dos dois cursos e, cujo nome do aluno seja "Daniela". Ou seja, com a condição de ter o nome "Daniela". Isso, portanto, é algo externo, não colocaremos dentro das chaves, mas sim fora:

```
db.alunos.find({
    $or : [
        {"curso.nome" : "Sistemas de informação"},
        {"curso.nome" : "Engenharia Química"}
    ],
    "nome" : "Daniela"
})
```

Se colarmos isso no Terminal veremos que estará funcionando, como não temos "Danielas" cursando engenharia ou sistemas, ele não nos trará nada. Podemos testar colocando mais um curso , o de "moda":

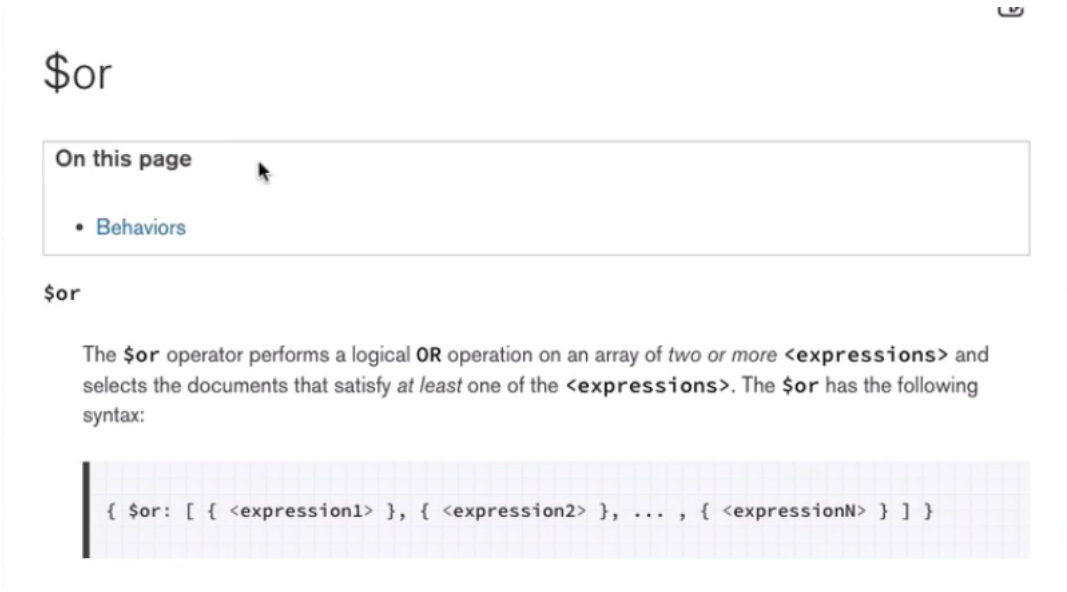
```
db.alunos.find({
    $or : [
        {"curso.nome" : "Sistemas de informação"},
        {"curso.nome" : "Engenharia Química"},
        {"curso.nome" : "Moda"}
    ],
    "nome" : "Daniela"
})
```

Se testarmos isso em nosso Terminal, ele nos mostrará "Daniela" no curso de "Moda".

```
... $or : [
... {"curso.nome" : "Sistemas de informação"},
... {"curso.nome" : "Engenharia Química"},
... {"curso.nome" : "Moda"}
... ],
... "nome" : "Daniela"
... })
{ "_id" : ObjectId("56cb043fb6d75ec12f75d3b9"), "nome" : "Daniela", "data_nasci
mento" : ISODate("1995-10-30T02:00:00Z"), "curso" : { "nome" : "Moda" }, "habil
idades" : [ { "nome" : "alemão", "nível" : "básico" } ] }
>
```

O que fizemos foi uma *query* utilizando o `or` e colocando dentro dele condições e fora temos as condições normais, sem sua interferência. O `find` permite que façamos as *query* navegando em nossos documentos com condições do tipo "e" (,) e condições do tipo "ou" (`or`) .

Temos diversas outras condições e isso pode lido na documentação do **MongoDB**, referente ao método `find` , passível de ser encontrada em: Documentação MongoDB (http://docs.mongodb.org/v3.0/reference/method/db.collection.find) . Se quisermos saber ainda mais detalhes sobre o **MongoDB**, no caso o `or` , basta jogarmos no *google*, buscarmos a documentação e veremos que condições devem ser passadas quando utilizamos ele.



Vamos observar uma última variação. Buscamos, antes, os alunos cujos nomes eram "Daniela" e que estão em um dos três cursos mencionados. Vamos fazer a mesma busca, mas utilizando o `in` .

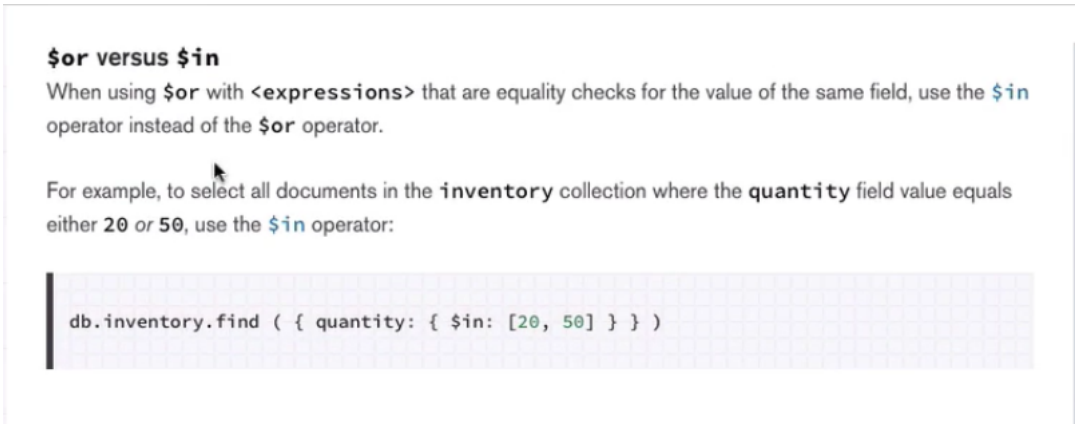
Antes, vamos observar como escreveríamos no *SQL*:

```
SELECT * FROM cursos WHERE nome = "Sistemas de informação" OR nome ="Engenharia Química" OR nome
```

Essa *query* funciona, mas ela é confusa. Se colocarmos um `AND` ao final isso não significa que ele estará ligado aos `OR` anteriores e teremos que lembrar de colocar um parenteses do primeiro "nome" até "Moda" para termos a certeza de que as três regras andam juntas. Ou seja, começam os detalhes capciosos! Na prática, no *SQL*, ao em vez de ficarmos usando o `OR` , podemos usar o `in` , que indica "estar dentro de algo". Repare:

```
SELECT * FROM cursos WHERE nome in ("Sistemas de informação", "Engenharia Química","Moda");
```

Temos o `in` no *SQL* e podemos usar ele, também, junto ao `db`. A documentação nos fala que para utilizá-lo devemos usar uma *Arrey*:



$or versus $in

When using **$or** with **<expressions>** that are equality checks for the value of the same field, use the **$in** operator instead of the **$or** operator.

For example, to select all documents in the **inventory** collection where the **quantity** field value equals either **20** *or* **50**, use the **$in** operator:

```
db.inventory.find ( { quantity: { $in: [20, 50] } } )
```

O `in` pode ser mais fácil de usar, em alguns casos. Se vc estiver com dúvidas, pode buscar a documentação do Mongo onde fala sobre o `in`. Como no *SQL* temos uma maneira diferente de usar as coisas, vamos ver como usamos o `in` no `db`. Queremos dizer que o `"curso.nome"` estará `"in"` (dentro) de "Sistemas de informação", "Engenharia Química". Escreveremos o seguinte:

```
db.alunos.find({
    "curso.nome" : {
        $in : ["Sistema de informação", "Engenharia Química"]
        }
})
```

Copiando isso que está no Editor e rodamos no Terminal. Veremos que estará tudo certo, ele nos trará, exatamente, o que pedimos:

```
> db.alunos.find({
... "curso.nome" : {
... $in : ["Sistemas de informação", "Engenharia Química"]
... }
... })
{ "_id" : ObjectId("56cb0139b6d75ec12f75d3b6"), "nome" : "Felipe", "data_nascim
ento" : ISODate("1994-03-26T03:00:00Z"), "curso" : { "nome" : "Sistemas de info
rmação" }, "notas" : [ 10, 9, 4.5 ], "habilidades" : [ { "nome" : "inglês", "ní
vel" : "avançado" }, { "nome" : "taekwondo", "nível" : "básico" } ] }
{ "_id" : ObjectId("56cb03e9b6d75ec12f75d3b8"), "nome" : "Alberto", "data_nasci
mento" : ISODate("1972-10-30T03:00:00Z"), "curso" : { "nome" : "Engenharia Quím
ica" }, "habilidades" : [ { "nome" : "inglês", "nível" : "intermediário" } ] }
>
```

Com isso, vimos tanto o `in` quanto o `or` e mostramos também o quanto é válido buscar informações na documentação e que não devemos ter medo de recorrer a isso no dia a dia.