MongoDB

Aula 3:
Tipos de dados e Operadores de consulta

Banco de Dados 2 (BD2)

Instituto J&F Germinare TECH

NoSQL

Ainda estamos comparando SQL com NoSQL, já vimos que:

NoSQL	SQL
collection.find()	SELECT * FROM table
collection.find({chave:valor})	SELECT * FROM table WHERE chave=valor
collection.insertOne() ou insertMany()	INSERT INTO table VALUES ...
collection.updateOne({chave:valor}, {$set{}}) ou updateMany()	UPDATE TABLE SET ... WHERE chave=valor
collection.delete({chave:valor})	DELETE FROM table WHERE chave=valor

SQL *vs* NoSQL

Enter MySQL Query:

```
SELECT Type FROM Places
WHERE Type IN('Type1','Type 2')
ORDER BY Type;
```

MongoDB Syntax:

```
db.Places.find({
    "Type": {
        "$in": ["Type1", "Type 2"]
    }
}, {
    "Type": 1
}).sort({
    "Type": 1
});
```

Find all contacts with at least one work phone

SQL CLI

```
select * from contact A, phones B where
A.did = B.did and B.type = 'work';
```

MongoDB CLI

```
db.contact.find({"phones.type":"work"});
```

SQL in Java

```
String s = "select * from contact A, phones B
where A.did = B.did and B.type = \'work\'";
ResultSet rs = execute(s);
```

MongoDB via Java driver

```
DBObject expr = new BasicDBObject();
expr.put("phones.type", "work");
Cursor c = contact.find(expr);
```

MongoDB

O MongoDB até força a barra comparando a sintaxe SQL com a sua:

É um tutorial que apresenta diferenças entre ambas as sintaxes e diz que em 30 dias você consegue aprender o mongoDB e ainda ver suas vantagens frente a SQL.

Tire suas conclusões: https://www.mongodb.com/resources/basics/databases/nosql-explained/nosql-vs-sql

Tipos de dados

Na última aula você lembra que criamos campos como: strings, integers, floats, arrays, objects, boolean.... Mas vamos ver quais tipos são aceitos no MongoDB

Type	Number	Alias	Notes
Double	1	“double”	
String	2	“string”	
Object	3	“object”	
Array	4	“array”	
Binary Data	5	“binData”	
Undefined	6	“undefined”	Deprecated
ObjectId	7	“objectId”	
Boolean	8	“bool”	
Date	9	“date”	
Null	10	“null”	
Regular Expression	11	“regex”	
DBPointer	12	“dbPointer”	Deprecated
JavaScript	13	“javascript”	
Symbol	14	“symbol”	Deprecated
Javascript (with scope)	15	“javascriptWithScope”	
32-bit integer	16	“int”	
Timestamp	17	“timestamp”	
64-bit Integer	18	“long”	
Decimal128	19	“decimal	New in Version 3.4
Min Key	-1	“minKey”	
Max Key	127	“maxKey”	

São muitos então você pode ir na documentação:

https://www.mongodb.com/docs/manual/reference/bson-types/

Tipos de dados

No COMPASS os dados são sempre visíveis, mas na CLI você pode usar o $type ou typeof.

Quando você insere valores com tipos de dados diferentes em um mesmo campo, esse campo terá como tipo um MIXED (tipos mistos), mas ainda continua sendo possível verificar um id pelo tipo ou valor.

Mas tem como procurar algo por tipo no MongoDB? SIM!

Antes de tudo, vamos recuperar o que foi feito aula passada, se não deu o **db.alunos.drop()**, dê agora e depois cole o script que esta na aula (**script_aula3.txt**) no seu terminal.

Testando

Agora vamos testar, vamos lembrar algumas coisas, adicione na collection alunos, para documento com campo **nome:”Fernanda”** um novo campo chamado **telefone**, com o valor de **“11999887766”** (como sendo String).

Adicione agora para o documento com o campo **nome:”Juliana”** um novo campo chamado **telefone**, com o valor de **11911223344** (como sendo um número)

Testando

```
dbGerminare> db.alunos.updateOne({nome:"Fernanda"}, {$set:{telefone:"11999887766"}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

```
dbGerminare> db.alunos.updateOne({nome:"Juliana"}, {$set:{telefone:1191122334 4}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

$or

Você saberia como procurar apenas esses dois documentos (da Juliana e da Fabiana) com o **find()**?

```
dbGerminare> db.alunos.find({$or:[{nome:"Juliana"},{nome:"Fabiana"}]})
```

O

```
dbGerminare> db.alunos.find({nome:"Fernanda",nome:"Juliana"})
[
  {
    _id: ObjectId("6442e27ffdd3d9055fa8ad7e"),
    nome: 'Juliana',
    sobrenome: 'Lima',
    idade: 16,
    matricula: 1019,
    media: 7.8,
    data_matricula: ISODate("2022-02-22T03:00:00.000Z"),
    estagio: true,
    academia: {
      nome: 'Germinare Tech',
      diretor: 'Raul Moreira',
      qnt_professores: 4
    },
    teste: 15
  }
]
```

ou se colocar sem o operador lógico OU?

Apenas o segundo campo é buscado

$exists

Mas queremos ver apenas os que tem o CAMPO telefone. Para isso usamos a query usando o operador **$exists**, observe a sintaxe:

```
dbGerminare> db.alunos.find({telefone:{$exists:true}} )
```

A busca é, campo existe?

Query:
find({telefone: {$exists: true} })

Se quiser pesquisar os que Não existem, mude para false:

Query:
find({telefone: {$exists: false} })

$type

Agora vamos entender como podemos buscar algo pelo tipo de valor de um campo.

Buscando o campo nome que tiveram valores cadastrados como STRING?

```
dbGerminare> db.alunos.find( { telefone : { $type : "string" } } )
```

Pode colocar o tipo ou número do tipo!

Query:

find({telefone: {$type: “tipo_buscado”} })

typeof

Mas e o outro? Foi cadastrado como int? integer? Tem como saber?

```
dbGerminare> typeof db.alunos.findOne({nome:"Fernanda"}).telefone
string
dbGerminare> typeof db.alunos.findOne({nome:"Juliana"}).telefone
number
```

Usando a função typeof e depois passando a query e o campo no final.
Cuidado que precisa ser no findOne, se não vai retornar **undefined**

AH, foi como “number”..... Então:

```
dbGerminare> db.alunos.find( { telefone : { $type : "number" } } )
```

typeof

Você pode também usar um forEach para varrer todos os documentos e verificar, mas isso vamos deixar para depois, ok? Só observe a query abaixo:

```
dbGerminare> db.alunos.find({}).forEach(function(doc) {
... print(typeof doc.telefone);
... })
undefined
undefined
number
string
```

$lte e $gte

Quais alunos têm médias abaixo de 7?

```
dbGerminare> db.alunos.find({media:{$lt:7}})
```

Query:

find({“media”: {$lt:7} })

$not

O operador **NÃO**, inverte uma operação lógica, por exemplo:

Pesquise todos os alunos que fazem parte de uma escola que **não** tem menos do que 10 professores, use o $not

```
dbGerminare> db.alunos.find({"escola.qnd_professores":{$not:{$lt:10}}})
[
```

Viu que nessa query a gente acessou um campo de um objeto?
Se esse for seu objetivo use a sintaxe: campo1.campo2 entre aspas duplas (“ “)

Query:

find({“academia.qnd_professores”: {$not: {$lt:5}} })

O nome do campo está errado mesmo, é **qnd_professores**, depois iremos corrigir isso!

$in e $nin

IN ou NOT IN

Este operador retorna os documentos que correspondem (ou não correspondem) aos valores especificados:

Vamos pesquisar apenas os alunos que **não** são das escolas de **Germinare Vet** e da **Germinare Tech**.

```
.find({"escola.nome":{$nin:["Germinare Vet", "Germinare Tech"]}})
```

Query:

find({“campo.subcampo”: {$nin: [“valor1”, “valor2”]} })

$ne

NOT EQUAL

O operador retorna os documentos onde o valor especificado seja igual (ou não é igual) à algo:

Todos os alunos que tem idade diferente de 16 anos.

```
dbGerminare> db.alunos.find({idade:{$ne:16}})
[
  {
    _id: ObjectId('67a24ece5fa9037b8ccb0ce3'),
    nome: 'Manoel',
    sobrenome: 'Carlos',
    idade: 15,
    matricula: 1024,
    media: 6.5,
    data_matricula: ISODate('2024-02-01T03:00:00.000Z'),
    estagio: false,
    escola: {
      nome: 'Germinare Business',
```

Trabalhando com arrays

Vamos cadastrar agora as disciplinas que esses alunos fazem, digamos que seria:

Nome	Disciplinas
Igor	[“Matemática”, “Análise de Dados”, “Banco de Dados”, “Estatística”]
Manoel	[“Vendas”, ”Português”, “Matemática”]
Juliana	[“Lógica de Programação”, “Análise de Dados”, “Desenvolvimento II”]
Fernanda	[“Banco de Dados”, “Desenvolvimento I”]

Trabalhando com arrays

```
dbGerminare> db.alunos.updateOne({nome:"Igor"}, {$set:{disciplinas:["Matemática",
... "Análise de Dados", "Banco de Dados", "Estatística"]}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

```
dbGerminare> db.alunos.updateOne({ nome: "Manoel" }, { $set: { disciplinas: ["Vendas", "Português", "Metemática"] } })
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

Trabalhando com arrays

```
dbGerminare> db.alunos.updateOne({ nome: "Juliana" }, { $set: { disciplinas: ["Lógica de Programação", "Análise de Dados", "Desenvolvimento II"] } }
)
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
dbGerminare> db.alunos.updateOne({ nome: "Fernanda" }, { $set: { disciplinas: ["Banco de Dados", "Desenvolvimento I"] } })
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

$elemMatch

O operador **$elemMatch** combina documentos que contêm um campo de matriz com pelo menos um elemento que corresponde a todos os critérios de consulta especificados:

Para entender ele, vamos criar essa nova collection abaixo:

```
dbGerminare> db.match.insertMany([{vetor:[92,89,98]}, {vetor:[85, 99, 99]}])
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId("6452839e3e6cdca8f454b89c"),
    '1': ObjectId("6452839e3e6cdca8f454b89d")
  }
}
dbGerminare> db.match.find()
[
  { _id: ObjectId("6452839e3e6cdca8f454b89c"), vetor: [ 92, 89, 98 ] },
  { _id: ObjectId("6452839e3e6cdca8f454b89d"), vetor: [ 85, 99, 99 ] }
]
```

$elemMatch

A consulta a seguir corresponde apenas aqueles documentos onde a matriz de resultados contém pelo menos um elemento que é maior ou igual a 90 e é menor que 95:

```
dbGerminare> db.match.find({vetor: {$elemMatch: {$gte: 90, $lt: 95}}})
[
  { _id: ObjectId("6452839e3e6cdca8f454b89c"), vetor: [ 92, 89, 98 ] }
]
```

$elemMatch

Embora possamos utilizar operadores de comparação como **$lte** e **$gte**, se especificarmos apenas uma única condição de consulta dentro de **$elemMatch**, e não estivermos utilizando o **$not** ou os operadores de **$ne**, a utilização do **$elemMatch** pode ser omitida, pois ele estaria essencialmente desempenhando a mesma função.

Há mais algumas coisas para se ter em mente ao usar este operador, principalmente:

- Você não pode especificar uma expressão **$where** em uma operação **$elemMatch**.
- Você não pode especificar uma expressão de consulta **$text** em uma operação **$elemMatch**.

$size

O operador $size retorna aqueles documentos onde o tamanho da matriz corresponde ao número de elementos especificados no argumento:

Quem são os alunos que estão matriculados em 4 disciplinas?

Query:

find({disciplinas: {$size: 4} })

```
dbGerminare> db.alunos.find({disciplinas:{$size:4}})
[
  {
    _id: ObjectId('67a24ece5fa9037b8ccb0ce2'),
    nome: 'Igor',
    sobrenome: 'José',
    idade: 16,
    matricula: 1023,
    media: 7.5,
    data_matricula: ISODate('2025-01-25T03:00:00.000Z'),
    estagio: false,
    escola: { nome: 'Germinare Vet', diretor: 'João Audi', qnd_professores: 5 },
    disciplinas: [
      'Matemática',
      'Análise de Dados',
      'Banco de Dados',
      'Estatística'
    ]
  }
]
```

$expr

Mas e se agora queremos ver quem são os alunos que estão matriculados em pelo menos 3 disciplinas.

```
db.alunos.find({$expr:{$gte:[{$size:"$disciplinas"}, 3]}})
```

Nesse caso precisa também do **$expr**
O operador **$expr** permite que você utilize expressões de agregação na linguagem da consulta.

Query:

find({$expr: {$gte: [{$size:”$disciplinas”}, 3] } })

$expr

Mostre quem são os alunos cuja idade seja maior que a quantidade de professores da sua escola:

```
dbGerminare> db.alunos.find({
...       $expr: { $gt: ["$idade", "$escola.qnd_professores"] }
... })
[
```

$regex

O operador $regex oferece habilidades de expressão regular para combinar strings em consultas. MongoDB alavanca expressões regulares compatíveis com o Perl:

https://perldoc.perl.org/perlre

https://www.mongodb.com/docs/manual/reference/operator/query/regex/

Vamos procurar todos os alunos matriculados em uma disciplina com nome Desenvolvimento.

```
dbGerminare> db.alunos.find({disciplinas:{$regex: /Desenvolvimento/}})
```

$Operadores aprendidos

Operador	Função	EX
$type	Buscar pelo tipo	{campo: {**$type**: “BSON type” }}
$or	Lógico -> Ou	{**$or**:[{cond1}, {cond2}]}
$exists	Um campo existe?	{campo: {**$exists**: true}}
$in $nin	Está em Não está em (IN , NOT IN)	{campo: {**$nin**: [valor1, valor2, ...]} }
$elemMatch	Possibilidade de fazer combinações em Arrays	{array: {**$elemMatch**: {condições1, condição2 } } }
$size	Retornar resultados que tenham um tamanho específico	{campo_array: {**$size**: tamanho} }
$expr	Usar uma expressão em uma consulta	{**$exp**: {condições.....} }
$regex	Uso de expressões regulares	{campo: {**$regex**: “/operadoresPERL/” }}