MongoDB

Aula 2:
Primeiros comandos: CRUD e Operadores

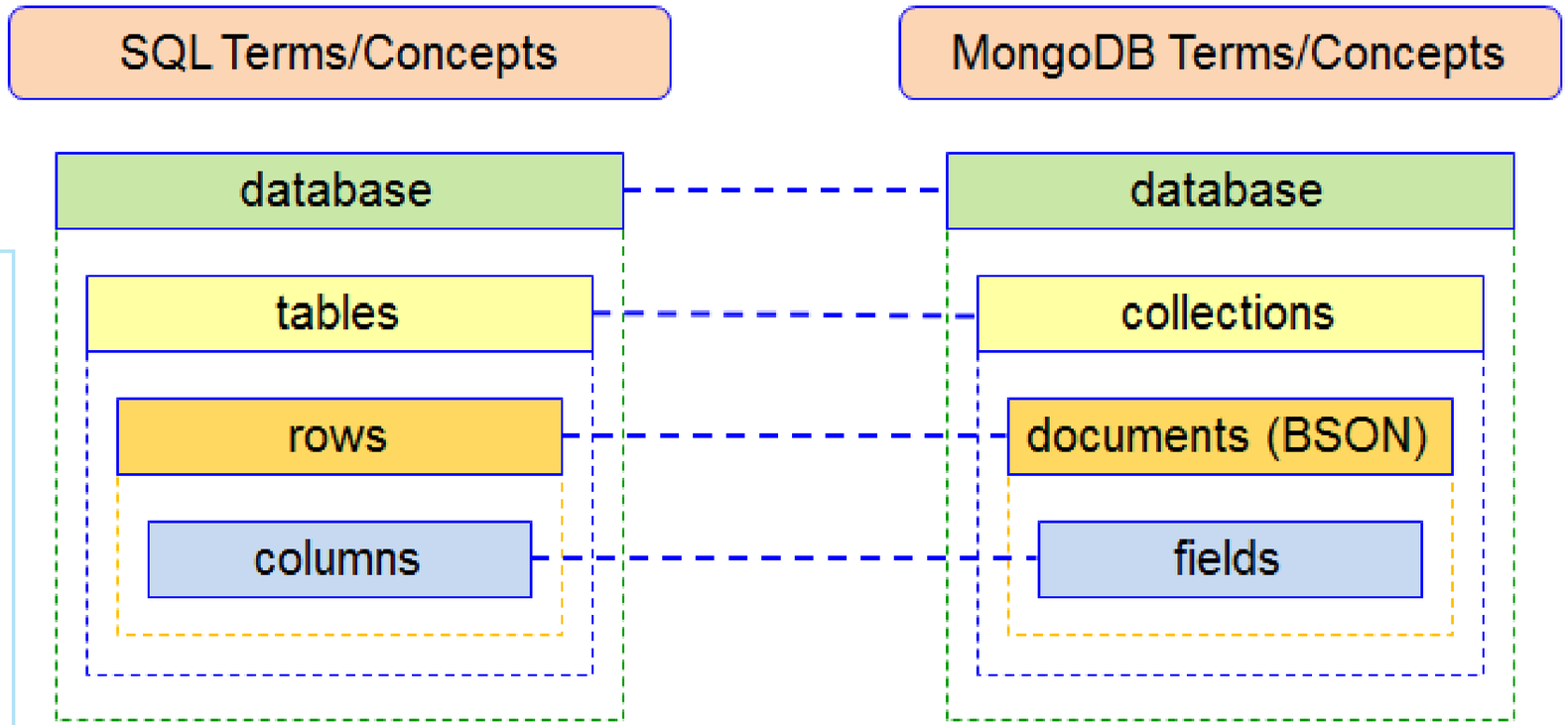
Banco de Dados 2 (BD2)

Instituto J&F Germinare TECH

NoSQL

Antes de começar a aprender os comandos MongoDB e utilizar queries, precisamos mudar um pouco a mentalidade.

Primeiro: TERMOS



SQL

Digamos que queremos criar um banco **dbGerminare**, para armazenar informações dos alunos com algumas características como: nome, sobrenome, idade, matricula, média, data_matricula, se está estagiando, escola.
Pensaríamos já em criar no mínimo 2 tabelas, lembra que fizemos isso no Postgres? Algo assim:

Nome	Sobrenome	Idade	Matricula	Média	Data_matricula	Escola (FK)	Estagio
Igor	José	16	1023	7.5	2025-02-21	1	False
Luciano	Gomes	15	1011	6.9	2025-02-22	2	True
...	...	...	...	...	...	...	...

aluno

escola

Codigo (PK)	Nome	Diretor	Qnd_prof
1	Germinare Tech	João Pilla	4
2	Germinare Business	Paulo Gualtiere	8
...	...	...	...

MongoDB

Isso exige a modelagem do banco inicialmente. Uma vantagem do MongoDB é a flexibilidade, os dados podem ser armazenados de maneira mais ‘livre’. Sem tanta preocupação com as chaves primárias, estrangeiras, e a modelagem inicial.

Vamos ver como esses dados abaixo podem ser inseridos no MongoDB?

aluno

Nome	Sobrenome	Idade	Matricula	Média	Data_matricula	Escola (FK)	Estagio
Igor	José	16	1023	7.5	2025-02-21	1	False
Luciano	Gomes	15	1011	6.9	2025-02-22	2	True
...	...	...	...	...	..	...	...

escola

Codigo (PK)	Nome	Diretor	Qnd_prof
1	Germinare Tech	João Pilla	4
2	Germinare Business	Paulo Gualtiere	8
...	...	...	...

MongoDB

Vamos usar o CMD primeiro, eu sei.... É chato? Mas vai ajudar a entender os comandos e operadores mais rápido. Depois a gente vai para o Compass, eu prometo.

Abra o CMD e ative o servidor.

Depois de iniciar o servidor, abra o **mongosh** em outro terminal....

Lembra como? Você fez na aula passada, mas tem no material também!

Comandos MongoDB

Quando você entrar, vai entrar no db teste>

```
test> cls
```

cls pode ser usado para limpar a tela

```
test> show dbs
PrimeiraAula   48.00 KiB
admin          40.00 KiB
config        108.00 KiB
local          84.00 KiB
test          120.00 KiB
test>
```

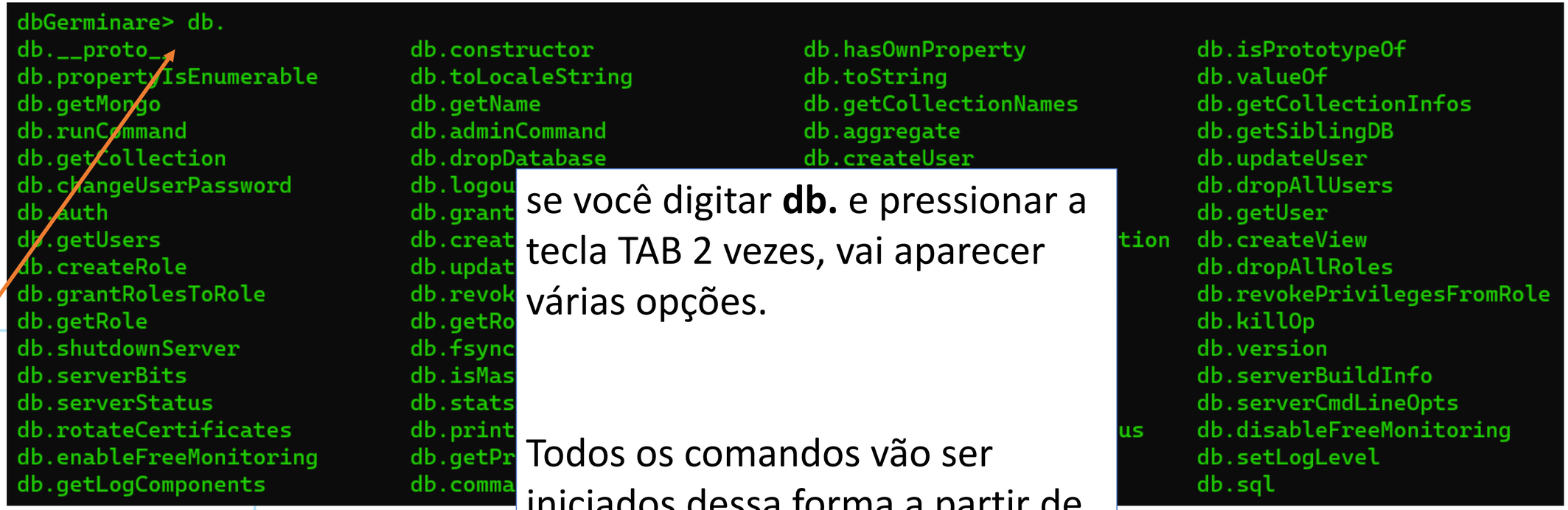
show dbs vai mostrar todos os bancos que você tem

```
test> use dbGerminare
switched to db dbGerminare
dbGerminare> show dbs
PrimeiraAula   48.00 KiB
admin          40.00 KiB
config        108.00 KiB
local          84.00 KiB
test          120.00 KiB
dbGerminare>
```

use dbGerminare vai mudar você para um novo banco de dados. Por mais que ele mude, esse banco ainda não foi criado na memória, pode dar o **show dbs** que não vai aparecer. Isso porque precisa ter uma **collection** para aparecer.

Comandos MongoDB

Vamos aprender fazendo uma analogia a uma operação CRUD
CREATE – READ – UPDATE – DELETE



se você digitar **db.** e pressionar a tecla TAB 2 vezes, vai aparecer várias opções.

Todos os comandos vão ser iniciados dessa forma a partir de agora.

db.collection.comando()

CREATE (C)

SQL: CREATE TABLE

SQL: INSERT INTO

- Criando uma **collection.**
- Vamos inserir o seguinte registro em uma collection chamada **alunos**:

Nome	Sobrenome	Idade	Matricula	Media	Data_matricula	Estagio
Igor	José	16	1023	7.5	2025-01-25	False

CREATE (C)

SQL: CREATE TABLE
SQL: INSERT INTO

- Criando uma **collection.**
- Vamos inserir o seguinte registro em uma collection chamada **alunos**:

Nome	Sobrenome	Idade	Matricula	Media	Data_matricula	Estagio
Igor	José	16	1023	7.5	2025-01-25	False

```
dbGerminare> db.alunos.insertOne({
... nome:"Igor",
... sobrenome:"José",
... idade:16,
... matricula:1023,
... media:7.5,
... data_matricula: new Date(2025,01,25),
... estagio:false
... })
{
  acknowledged: true,
  insertedId: ObjectId('679aa3aca6ea82e9f6cb0ce2')
}
```

Para continuar na linha de baixo digite ENTER, até fechar o), não precisa de ;

Mas nós criamos a collection **alunos** antes? Não. Mas para o MongoDB, isso não importa. Caso o usuário possua as credenciais necessárias, você pode mencionar **collections** e até mesmo **databases** inexistentes, e o banco de dados se encarrega de criá-los automaticamente. Isso só é possível graças a flexibilidade de **collections** ao não pré-determinar a estrutura dos dados antes de inseri-los.

CREATE (C)

```
dbGerminare> db.alunos.
db.alunos.__proto__                          db.alunos.constructor                db.alunos.hasOwnProperty
db.alunos.isPrototypeOf                      db.alunos.propertyIsEnumerable       db.alunos.toLocaleString
db.alunos.toString                           db.alunos.valueOf                    db.alunos.aggregate
db.alunos.bulkWrite                          db.alunos.countDocuments             db.alunos.deleteMany
db.alunos.deleteOne                          db.alunos.distinct                   db.alunos.estimatedDocumentCount
db.alunos.find                               db.alunos.findOne                    db.alunos.renameCollection
db.alunos.findOneAndDelete                   db.alunos.findOneAndReplace          db.alunos.findOneAndUpdate
db.alunos.insertMany                         db.alunos.insertOne                  db.alunos.isCapped
db.alunos.replaceOne                         db.alunos.updateMany                 db.alunos.updateOne
db.alunos.compactStructuredEncryptionData    db.alunos.convertToCapped            db.alunos.createIndexes
db.alunos.createIndex                        db.alunos.ensureIndex                db.alunos.getIndexes
db.alunos.getIndexSpecs                      db.alunos.getIndices                 db.alunos.getIndexKeys
db.alunos.dropIndexes                                                             db.alunos.totalIndexSize
db.alunos.getDB                                                                   db.alunos.dataSize
db.alunos.storageSize                                                             db.alunos.drop
db.alunos.exists                                                                  db.alunos.getName
db.alunos.runCommand                                                              db.alunos.stats
db.alunos.latencyStats                                                            db.alunos.initializeUnorderedBulkOp
db.alunos.getPlanCache                                                            db.alunos.hideIndex
db.alunos.unhideIndex
```

Para criar uma collection, basta digitar o comando **db.nome_collection.insertOne()** ou **insertMany()** que vai ser criado a collection e o banco vai aparecer no **show dbs**

Datas no MONGODB

- Primeiro problema:

```
dbGerminare> db.alunos.insertOne({
... nome:"Igor",
... sobrenome:"José",
... idade:16,
... matricula:1023,
... media:7.5,
... data_matricula: new Date(2025,01,25),
... estagio:false
... })
{
  acknowledged: true,
  insertedId: ObjectId('679aa3aca6ea82e9f6cb0ce2')
}
```

Mas nem tudo é bom, viu como foi para inserir a data? O MongoDB segue a especificação BSON e, por isso, o valor das datas é armazenado internamente como um inteiro de 64 bits (com valores positivos e negativos) que contém a quantidade de milisegundos desde primeiro de janeiro de 1970. Na prática, trabalhamos com a função **Date()** ou **ISODate()** de maneira semelhante à manipulações de datas no JavaScript.

Datas no MONGODB

- Mas piora, calma.... Vamos ver a informação cadastrada usando o **find().**

```
dbGerminare> db.alunos.find()
[
  {
    _id: ObjectId('679aa3aca6ea82e9f6cb0ce2'),
    nome: 'Igor',
    sobrenome: 'José',
    idade: 16,
    matricula: 1023,
    media: 7.5,
    data_matricula: ISODate('2025-02-25T03:00:00.000Z'),
    estagio: false
  }
]
```

Foi essa data que a gente cadastrou?

NÃO MESMO!

Quando utilizamos o **new Date()** devemos informar o mês entre 0 á 11, onde 0 é começa em Janeiro e 11 termina em Dezembro.

AFF! Pelo menos é só para o mês ☺

Daqui a pouco a gente atualiza isso, vamos seguir....

CREATE (C)

- Vamos criar mais esses documentos na nossa coleção:

nome	sobrenome	idade	matricula	media	data_matricula	estagio
Manoel	Carlos	15	1024	6.5	2024-02-01	False
Juliana	Lima	16	1019	6.0	2025-01-24	True
Fernanda	Matos	16	1025	5.9	2025-01-25	True

- Para inserir mais de um registro ao mesmo tempo, usamos o **insertMany()** com uma lista de documentos, observe o comando a seguir:

Coloque as datas corretamente agora...

CREATE (C)

```
dbGerminare> db.alunos.insertMany([
...     {nome:"Manoel", sobrenome: "Carlos", idade:15, matricula:1024, media:6.5, data_matricula:new Date(2024, 01, 01), estagio:false},
...     {nome:"Juliana", sobrenome: "Lima", idade:16, matricula:1019, media:6.0, data_matricula:new Date(2024, 00, 24), estagio:true},
...     {nome:"Fernanda", sobrenome: "Matos", idade:16, matricula:1025, media:5.9, data_matricula:new Date(2024, 00, 25), estagio:true},
... ])
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('679aa566a6ea82e9f6cb0ce3'),
    '1': ObjectId('679aa566a6ea82e9f6cb0ce4'),
    '2': ObjectId('679aa566a6ea82e9f6cb0ce5')
  }
}
dbGerminare>
```

Perceba que quando usa o **insertMany()** você deve colocar os documentos dentro de uma array separados por virgula, algo como:

.insertMany([{doc1}, {doc2}, {doc3}])

READ (R)

SQL: SELECT * FROM

- Para visualizar todos os documentos, pode usar o **.find()**
  - Pode inclusive usar o **.find().pretty()** <- A versão mais nova já vem com o pretty padrão.

```
dbGerminare> db.alunos.find()
[
  {
    _id: ObjectId('679aa3aca6ea82e9f6cb0ce2'),
    nome: 'Igor',
    sobrenome: 'José',
    idade: 16,
    matricula: 1023,
    media: 7.5,
    data_matricula: ISODate('2025-02-25T03:00:00.000Z'),
    estagio: false
  },
  {
    _id: ObjectId('679aa566a6ea82e9f6cb0ce3'),
    nome: 'Manoel',
    sobrenome: 'Carlos',
    idade: 15,
    matricula: 1024,
    media: 6.5,
    data_matricula: ISODate('2024-02-01T03:00:00.000Z'),
    estagio: false
  },
  {
```

```
  {
    _id: ObjectId('679aa566a6ea82e9f6cb0ce4'),
    nome: 'Juliana',
    sobrenome: 'Lima',
    idade: 16,
    matricula: 1019,
    media: 6,
    data_matricula: ISODate('2024-01-24T03:00:00.000Z'),
    estagio: true
  },
  {
    _id: ObjectId('679aa566a6ea82e9f6cb0ce5'),
    nome: 'Fernanda',
    sobrenome: 'Matos',
    idade: 16,
    matricula: 1025,
    media: 5.9,
    data_matricula: ISODate('2024-01-25T03:00:00.000Z'),
    estagio: true
  }
]
```

READ (R)

- Se você quiser ver apenas 1 registro pode usar o **findOne().** Vai mostrar o primeiro que encontrar. Se não colocar nenhum filtro vai ser o primeiro criado mesmo.

```
dbGerminare> db.alunos.findOne()
{
  _id: ObjectId('679aa3aca6ea82e9f6cb0ce2'),
  nome: 'Igor',
  sobrenome: 'José',
  idade: 16,
  matricula: 1023,
  media: 7.5,
  data_matricula: ISODate('2025-02-25T03:00:00.000Z'),
  estagio: false
}
```

Filtros no find()

- Mas e se quisermos ver todos que estão estagiando? Observe a query abaixo:

```
dbGerminare> db.alunos.find({estagio:true})
[
  {
    _id: ObjectId('679aa566a6ea82e9f6cb0ce4'),
    nome: 'Juliana',
    sobrenome: 'Lima',
    idade: 16,
    matricula: 1019,
    media: 6,
    data_matricula: ISODate('2024-01-24T03:00:00.000Z'),
    estagio: true
  },
  {
    _id: ObjectId('679aa566a6ea82e9f6cb0ce5'),
    nome: 'Fernanda',
    sobrenome: 'Matos',
    idade: 16,
    matricula: 1025,
    media: 5.9,
    data_matricula: ISODate('2024-01-25T03:00:00.000Z'),
    estagio: true
  }
]
```

Precisa especificar a chave e o valor que vai buscar.

Se não tiver nenhuma correspondência, vai aparecer em branco.

Filtros no find()

- E se....

```
dbGerminare> db.alunos.find( {estagios: true} )

dbGerminare> db.alunos.find( {estagio: } )
Uncaught:
SyntaxError: Unexpected token (1:26)

> 1 | db.alunos.find( {estagio: } )
    |                           ^
  2 |

dbGerminare> db.alunos.find( {: true} )
Uncaught:
SyntaxError: Unexpected token (1:17)

> 1 | db.alunos.find( {: true} )
    |                 ^
  2 |
```

Errar o nome da chave ou valor: **EM BRANCO**

Deixar um campo em branco: **ERRO**

Filtros no find()

- Se tiverem muitos e você quiser saber apenas quantos tem? Use o **count()** depois.

```
dbGerminare> db.alunos.find( {estagio: true} ).count()
2
```

- Procure todos que tem 16 anos.

```
dbGerminare> db.alunos.find({idade:16})
```

- Para procurar o primeiro aluno cadastrado com idade de 16 anos, basta usar o **findOne()**

```
dbGerminare> db.alunos.findOne({idade:16})
```

Filtros no find()

- Procurando todos que tem 16 anos e estão estagiando?

A essa é fácil, só colocar 2 documentos, separados por virgula....

```
dbGerminare> db.alunos.find({idade:16},{estagio:true})
[
  { _id: ObjectId("6442e196fdd3d9055fa8ad7c"), estagio: false },
  { _id: ObjectId("6442e27ffdd3d9055fa8ad7e"), estagio: true },
  { _id: ObjectId("6442e27ffdd3d9055fa8ad7f"), estagio: true }
]
```

WRONG!

Todos que tem idade = 16

Valor do campo estagio

Filtros no find()

- Procurando todos que tem 16 anos e estão estagiando?

Se existirem ambas as chaves e valores, use apenas um documento mesmo.

```
dbGerminare> db.alunos.find({idade:16,estagio:true})
[
  {
    _id: ObjectId('679aa566a6ea82e9f6cb0ce4'),
    nome: 'Juliana',
    sobrenome: 'Lima',
    idade: 16,
    matricula: 1019,
    media: 6,
    data_matricula: ISODate('2024-01-24T03:00:00.000Z'),
    estagio: true
  },
  {
    _id: ObjectId('679aa566a6ea82e9f6cb0ce5'),
    nome: 'Fernanda',
    sobrenome: 'Matos',
    idade: 16,
    matricula: 1025,
    media: 5.9,
    data_matricula: ISODate('2024-01-25T03:00:00.000Z'),
    estagio: true
  }
]
```

DONE!

Query:

find({idade:16 , estagio:true})

Filtros no find()

- Vamos complicar um pouco?

Você pode até usar os dois documentos, mas precisaria usar um operador lógico **AND**:

```
dbGerminare> db.alunos.find({$and:[{idade:16},{estagio:true}]})
[
  {
    _id: ObjectId('679aa566a6ea82e9f6cb0ce4'),
    nome: 'Juliana',
    sobrenome: 'Lima',
    idade: 16,
    matricula: 1019,
    media: 6,
    data_matricula: ISODate('2024-01-24T03:00:00.000Z'),
    estagio: true
  },
  {
    _id: ObjectId('679aa566a6ea82e9f6cb0ce5'),
    nome: 'Fernanda',
    sobrenome: 'Matos',
    idade: 16,
    matricula: 1025,
    media: 5.9,
    data_matricula: ISODate('2024-01-25T03:00:00.000Z'),
    estagio: true
  }
]
```

Query:

find({$*and*: [{idade:16},{estagio:true}] })

Operadores

- É possível usar vários operadores em um filtro, todos devem começar com o caractere do cifrão ($).

Operador	Função
$sum	Utilizado para a soma de valores.
$avg	Utilizado para encontrar a média entre os valores.
$gte	Utilizado como critério de "maior ou igual".
$lte	Utilizado como critério de "menor ou igual".
$min	Utilizado para recuperar o "menor valor".
$max	Utilizado para recuperar o "maior valor".
$in	Utilizado para localizar dentro de um array qualquer um dos parâmetros.
$all	Utilizado para localizar dentro de um array todos os elementos do parâmetro.
$regex	Utilizado para o uso de expressões regulares.
$and	Idêntico ao operador "and" do SQL.
$or	Idêntico ao operador "or" do SQL.

Documentação oficial:

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

Operadores

- Vamos pesquisar todos os alunos que estão com média acima de 7, ou “aprovados”.

```
dbGerminare> db.alunos.find({media:{$gte:7}})
[
  {
    _id: ObjectId('679aa3aca6ea82e9f6cb0ce2'),
    nome: 'Igor',
    sobrenome: 'José',
    idade: 16,
    matricula: 1023,
    media: 7.5,
    data_matricula: ISODate('2025-02-25T03:00:00.000Z'),
    estagio: false
  }
]
```

Query:

find({media: {*$gte*:7} })

$gte = greater than or equal

- Vamos voltar em operadores e filtros nas próximas aulas... Continuando o CRUD

UPDATE (U)

SQL: UPDATE ... SET

- Atualizar registros precisa do SET, mas assim como os operadores, precisa do cifrão antes.
- Digamos que “Juliana Lima” teve sua nota revisada e foi para 7.8. Como atualizar?

```
dbGerminare> db.alunos.updateOne({nome:"Juliana"}, {$set:{media:7.8}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

Êxito: 1

Query:

update({nome: “Juliana”}, {$set: {media:7.8} } })

O que vai pesquisar

O que vai mudar

UPDATE (U)

- O update não altera a ordem de visualização de um **find()** como era no SQL.

Se colocar apenas o $set, sem um filtro antes, é o mesmo que UPDATE sem WHERE.

```
dbGerminare> db.alunos.find()
[
  {
    _id: ObjectId('679aa3aca6ea82e9f6cb0ce2'),
    nome: 'Igor',
    sobrenome: 'José',
    idade: 16,
    matricula: 1023,
    media: 7.5,
    data_matricula: ISODate('2025-02-25T03:00:00.000Z'),
    estagio: false
  },
  {
    _id: ObjectId('679aa566a6ea82e9f6cb0ce3'),
    nome: 'Manoel',
    sobrenome: 'Carlos',
    idade: 15,
    matricula: 1024,
    media: 6.5,
    data_matricula: ISODate('2024-02-01T03:00:00.000Z'),
    estagio: false
  },
  {
    _id: ObjectId('679aa566a6ea82e9f6cb0ce4'),
    nome: 'Juliana',
    sobrenome: 'Lima',
    idade: 16,
    matricula: 1019,
    media: 7.8,
    data_matricula: ISODate('2024-01-24T03:00:00.000Z'),
    estagio: true
  },
  {
```

UPDATE (U)

- Lembra que a data do Igor José está errada? Use uma query para alterar essa data, na busca, procure pelo nome e sobrenome dele.

```
dbGerminare> db.alunos.updateOne({nome:"Igor", sobrenome:"José"}, {$set: {data_matricula:new Date(2025,00,25)} })
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

Query:

updateOne({nome:"Igor", sobrenome:"José"}, {$set: {data_matricula:new Date(2025,00,25)} })

UPDATE (U)

- No MongoDB para inserir um novo campo, também é utilizado o **update().**
- Mas nesse caso, podemos usamos o **updateMany().**

Digamos que queremos adicionar as informação das escolas dos alunos, mas lembra que isso era outra tabela no SQL? No mongoDB isso vai ser um outro documento como valor de uma chave... Vamos ver.

nome	sobrenome	idade	matricula	media	data_matricula	escola (FK)	estagio
Igor	José	16	1023	7.5	2022-02-21	1	False
Luciano	Gomes	15	1011	6.9	2022-02-22	2	True
...	...	...	...	...	...	...	...

Codigo (PK)	Nome	Diretor	Qnd_prof
1	Germinare Tech	João Pilla	4
2	Germinare Business	Paulo Gualtiere	8
...	...	...	...

UPDATE (U)

Vamos inserir a seguinte estrutura:

Nome	Sobrenome	Idade	Matricula	Media	Data_matricula	Estagio
Igor	José	16	1023	7.5	2025-01-25	False
Manoel	Carlos	15	1024	6.5	2024-02-01	False
Juliana	Lima	16	1019	7.8	2024-01-24	True
Fernanda	Matos	16	1025	5.9	2024-01-25	True

```
escola:{
    nome:”Germinare Vet”,
    diretor: “João Audi”,
    qnt_professores: 5
}
```

```
escola:{
    nome:”Germinare Business”,
    diretor: “Paulo Gualtiere”,
    qnt_professores: 14
}
```

```
escola:{
    nome:”Germinare Tech”,
    diretor: “João Pilla”,
    qnt_professores: 7
}
```

UPDATE (U)

- 1º vamos adicionar em todos a mesma informação:

```
dbGerminare> db.alunos.updateMany({},
... {$set: {escola: {nome:"Germinare Tech", diretor: "João Pilla", qnd_professores:4 } } }
... )
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 4,
  modifiedCount: 4,
  upsertedCount: 0
}
dbGerminare>
```

SQL: INSERT INTO

Query:

```
updateMany({}, {$set:
                    {escola:{   nome:”Germinare Tech”,
                                diretor: “João Pilla”,
                                qnd_professores:4 }
                    }
                }
)
```

UPDATE (U)

- Visualize o que aconteceu com o **.find()**

- Agora modifique para que esses 2 documentos fiquem com:

Nome	Sobrenome	Idade	Matricula	Media	Data_matricula	Estagio
Igor	José	16	1023	7.5	2025-01-25	False
Manoel	Carlos	15	1024	6.5	2024-02-01	False

```
escola:{
    nome:”Germinare Vet”,
    diretor: “João Audi”,
    qnt_professores: 5
}
```

```
escola:{
    nome:”Germinare Business”,
    diretor: “Paulo Gualtiere”,
    qnt_professores: 14
}
```

UPDATE (U)

```
dbGerminare> db.alunos.updateOne({nome:"Igor"},
... {$set: {escola: {nome:"Germinare Vet", diretor: "João Audi", qnd_professores:5 } } }
... )
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
dbGerminare> db.alunos.updateOne({nome:"Manoel"},
... {$set: {escola: {nome:"Germinare Business", diretor: "Paulo Gualtiere", qnd_professores:14 } } }
... )
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
dbGerminare>
```

Como devem estar os dados:

```
dbGerminare> db.alunos.find()
[
  {
    _id: ObjectId('679aa3aca6ea82e9f6cb0ce2'),
    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 }
  },
  {
    _id: ObjectId('679aa566a6ea82e9f6cb0ce3'),
    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',
      diretor: 'Paulo Gualtiere',
      qnd_professores: 14
    }
  },
  {
    _id: ObjectId('679aa566a6ea82e9f6cb0ce4'),
    nome: 'Juliana',
    sobrenome: 'Lima',
    idade: 16,
    matricula: 1019,
    media: 7.8,
    data_matricula: ISODate('2024-01-24T03:00:00.000Z'),
    estagio: true,
    escola: {
      nome: 'Germinare Tech',
      diretor: 'João Pilla',
      qnd_professores: 7
    }
  },
  {
    _id: ObjectId('679aa566a6ea82e9f6cb0ce5'),
    nome: 'Fernanda',
    sobrenome: 'Matos',
    idade: 16,
    matricula: 1025,
    media: 5.9,
    data_matricula: ISODate('2024-01-25T03:00:00.000Z'),
    estagio: true,
    escola: {
      nome: 'Germinare Tech',
      diretor: 'João Pilla',
      qnd_professores: 7
    }
  }
]
```

DELETE (D)

- Vamos deletar o aluno de matricula: **1024**

```
dbGerminare> db.alunos.deleteOne({matricula:1024})
{ acknowledged: true, deletedCount: 1 }
dbGerminare> db.alunos.find().count()
3
dbGerminare>
```

Deletando um DB

- Vai apagar tudo que tem no banco, claro!

```
dbGerminare> db.drop
db.dropDatabase  db.dropUser      db.dropAllUsers  db.dropRole      db.dropAllRoles

dbGerminare> db.dropDatabase()
{ ok: 1, dropped: 'dbGerminare' }
dbGerminare>
```

CRUD mongoDB

- Hoje aprendemos os comandos para realizar um CRUD no mongoDB, são eles:

CREATE:
- insertMany()
- insertOne()

READ
- find()

UPDATE
- updateOne()
- updateMany()

DELETE:
- delete()
- deleteOne()
- deleteMany()

*Além do **find()**, **update()** e **delete()** tem:*

findOne()
findOneAndDelete()
findOneAndReplace()
findOneAndUpdate()

updateOne() e findOneAndUpdate()

No MongoDB, tanto **updateOne()** quanto **findOneAndUpdate()** são usados para atualizar um único documento em uma coleção, mas há diferenças importantes entre eles:

- **updateOne():** Retorna um objeto contendo informações sobre a operação, como o número de documentos modificados, mas não retorna o documento atualizado.

- **findOneAndUpdate():** Retorna o próprio documento antes ou depois da atualização, dependendo do parâmetro *returnDocument*.

updateOne() e findOneAndUpdate()

- **updateOne():**

```
dbGerminare> db.alunos.insertOne({nome:"Grilo"})
{
  acknowledged: true,
  insertedId: ObjectId('67a25cd05fa9037b8ccb0ce8')
}
dbGerminare> db.alunos.updateOne({nome:"Grilo"}, {$set:{idade:35}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

- **findOneAndUpdate():**

```
dbGerminare> db.alunos.findOneAndUpdate({nome:"Grilo"}, {$set:{idade:37}}, {returnDocument:"after"})
{ _id: ObjectId('67a25cd05fa9037b8ccb0ce8'), nome: 'Grilo', idade: 37 }
dbGerminare> db.alunos.findOneAndUpdate({nome:"Grilo"}, {$set:{idade:40}}, {returnDocument:"before"})
{ _id: ObjectId('67a25cd05fa9037b8ccb0ce8'), nome: 'Grilo', idade: 37 }
```

findOneAndUpdate() and ..Replace()

No MongoDB, tanto **updateOne()** quanto **findOneAndUpdate()** são usados para atualizar um único documento em uma coleção, mas há diferenças importantes entre eles:

- **findOneAndUpdate()** requer operadores de atualização, como $set, $unset, $inc, etc.
- **findOneAndReplace()** não aceita operadores de atualização. Ele substitui todo o documento diretamente. Substituir completamente o documento mantendo apenas o _id.