

Banco de dados não relacionais



mongoDB®

Document



Tipo de dados e Operações Básicas

Prof. Me. Ricardo Resende de Mendonça
ricardo.mendonca@online.uscs.edu.br

Prof. Me. Renato Carioca Duarte
renato.duarte@online.uscs.edu.br

Introdução

Unidade 2

Um banco de dados NOSQL voltado a documentos é conhecido com *document database*.

Sendo assim ele se preocupa em armazenar documentos, que no caso do MongoDB são arquivos muito próximos ao padrão JSON.



Introdução

Unidade 2

Fazendo uma analogia ao modelo relacional

Relacional



Não Relacional



Banco de dados - - - - - → **Banco de dados**

Tabelas - - - - - → **Collections**

Tuplas - - - - - → **Documentos**

Atributos - - - - - → **Atributos**



Introdução

Unidade 2

Fazendo uma analogia ao modelo relacional

Relacional



Banco de dados

Tabelas

Tuplas

Atributos

Schema



Não Relacional



Banco de dados

Collections

Documentos

Atributos

Schemaless





MongoDB Query Language

MQL é uma linguagem usada para consultar e manipular dados armazenados no banco de dados **MongoDB**. Projetada para ser semelhante à estrutura de documentos **JSON**, o **MQL** oferece uma variedade de operadores e funcionalidades para consultas simples e avançadas, exclusão, atualizações de documentos e operações de agregação, permitindo a interação com os dados.



// Acessar o shell do mongodb
mongosh

// Exibi a lista de banco de dados do servidor
SHOW dbs

// Exibi a lista de banco de dados do servidor
SHOW databases

// Seleciona um banco de dados existente ou cria caso não exista
USE uscs

Alguns cuidados devem ser tomados ao escrevermos qualquer código que requer uma sintaxe específica.

Algumas dicas que podem tornar a escrita e leitura de código MQL mais fácil:

1. Indentação é uma necessidade, facilitando a leitura do código;
2. Padronização de nomenclaturas;
3. Uniformidade na utilização de aspas simples ' ou dupla ";
4. Os nomes dos atributos podem ser envoltos por aspas simples ou duplas, porém não há obrigatoriedade de se usar aspas;
5. Documentos que contém muitos atributos devem ser organizados de forma que cada atributo fica em uma linha;
6. Atributos do tipo array de objetos devem ser organizados de tal forma que cada objeto seja escrito em uma nova linha;




Alguns exemplos utilizando a **collection** estudantes:

- ❖ Repare que toda string está envolta por um conjunto de aspas simples ou duplas, nunca misturados.

```
1      db.estudantes.insertOne(  
2          { _id: 1, nome: 'João da Silva', sexo: 'M' }  
3      );  
4  
5      db.estudantes.insertOne(  
6          { "_id": 2, "nome": 'Mario Gomes', "sexo": 'M' }  
7      );  
8  
9      db.estudantes.insertOne(  
10         { _id: 3, nome: "Maria de Paula", sexo: "F" }  
11     );
```



❖ Exemplos de códigos ruins ou inválidos.

```
1      db.estudantes.insertOne(  
2          { _id: 1, Nome: 'João da Silva', sexo: 'M' }   
3      );  
4  
5      db.estudantes.insertOne(  
6          { "_id": 2, "nome": "Mario Gomes", "sexo": 'M' }   
7      );  
8  
9      db.estudantes.insertOne(  
10         { _id: 3, nome: "Maria de Paula", sexo: 'F' }   
11     );
```

Red arrows point to the following issues in the code:

- Line 2: `Nome` (non-standard attribute name) and `sexo` (non-standard attribute name).
- Line 6: `"sexo": 'M'` (mixed quoting).
- Line 10: `nome: "Maria de Paula", sexo: 'F'` (mixed quoting).

❖ Problemas:

- ❖ Linha 02: nomenclaturas dos atributos não padronizada;
- ❖ Linha 06: Valor do atributo iniciado com " e finalizado com ';
- ❖ Linha 10: Falta de padronização na utilização de " ou '.

❖ Exemplos de códigos ruins ou inválidos.

```
1 db.estudantes.insertOne({
2   _id: 1, nome: "João da Silva", idade: 20, sexo: "masculino",
3   curso: "Ciência da Computação", endereco: {rua: "Rua Conceição",
4   numero: "321", cidade: "São Caetano do Sul", estado: "SP",
5   cep: "09530-060"}, contatos: {email: "joao.silva@online.uscs.edu.br",
6   telefone: "+55 11 4227-7816"}, disciplinas: [
7     { nome: "Banco de dados não relacionais", carga_horaria: 80 },
8     { nome: "Algoritmos e estrutura de dados", carga_horaria: 80 },
9     { nome: "Métodos ágeis de desenv. de software", carga_horaria: 60 }
10  ], data_matricula: new Date("2030-01-01"),ativo: true
11 })
```

❖ Problemas:

- ❖ Incoerência na indentação de código.

❖ Exemplo de código com maior legibilidade:

```
1 db.estudantes.insertOne({  
2   _id: 1,  
3   nome: "João da Silva",  
4   idade: 20,  
5   sexo: "masculino",  
6   curso: "Ciência da Computação",  
7   endereco: {  
8     rua: "Rua Conceição",  
9     numero: "321",  
10    cidade: "São Caetano do Sul",  
11    estado: "SP",  
12    cep: "09530-060"  
13  },  
14  contatos: {  
15    email: "joao.silva@online.uscs.edu.br",  
16    telefone: "+55 11 4227-7816"  
17  },  
18  disciplinas: [  
19    { nome: "Banco de dados não relacionais", carga_horaria: 80 },  
20    { nome: "Algoritmos e estrutura de dados", carga_horaria: 80 },  
21    { nome: "Métodos ágeis de desenv. de software", carga_horaria: 60 }  
22  ],  
23  data_matricula: new Date("2030-01-01"),  
24  ativo: true  
25 })
```



Data Types

MongoDB Server stores data using the **BSON** format which supports some additional data types that are not available using the **JSON** format. Compared to the legacy `mongo` shell, MongoDB Shell (`mongosh`) has type handling which is better aligned with the default types used by the MongoDB drivers.

<https://www.mongodb.com/docs/mongodb-shell/reference/data-types/>

Tipo de dados - String

Exemplos utilizando a **collection** de estudantes:

Tipo de dado utilizado para armazenar texto.

```
1 db.estudantes.insertOne({  
2     nome: "Gilberto Mendes",  
3     curso: "Ciência da Computação"  
4 })
```

Tipo de dados – Integer e Double

Unidade 2

Utilizado para armazenar números inteiros (*integer*) ou de ponto flutuante (*double*).

```
1 db.estudantes.insertOne({  
2     idade: 18,  
3     nota: 8.2  
4 })
```

Numeric Types

<https://www.mongodb.com/docs/mongodb-shell/reference/data-types/#numeric-types>



Tipo de dados – Booleano

Utilizado para armazenar um valor lógico, verdadeiro (**true**) ou falso (**false**).

```
1 db.estudantes.insertOne({  
2     situacao: true,  
3     reprovado: false  
4 })
```

Tipo de dados – Array de valores simples

Unidade 2

Utilizado para armazenar um conjunto de valores simples do mesmo tipo.

```
1 db.estudantes.insertOne({  
2     disciplinas: [  
3         "Banco de dados não relacionais",  
4         "Algoritmos de dados",  
5         "Engenharia de Software"  
6     ]  
7 })
```


Tipo de dados – Array de Objetos

Utilizado para armazenar um conjunto de objetos, não havendo necessidade dos objetos do array possuírem a mesma estrutura.

```
1 db.estudantes.insertOne({
2   telefones: [
3     {ddd: 11, numero: "4598-2562", ramal: null},
4     {ddd: 11, numero: "5587-2362", ramal: null},
5     {ddd: 11, numero: "4867-2548", observacao: "Apenas Recados"}
6   ]
7 })
```

Tipo de valor – Null

Utilizado para representar a ausência de valor de um atributo.

```
1 db.estudantes.insertOne({  
2     nome: "Leandro de Paula",  
3     email: null  
4 })
```

Tipo de dados – Objeto (Embedded Document)

Utilizado para armazenar um subdocumento como valor de um atributo.

```
1 db.estudantes.insertOne({
2     endereco: {
3         logradouro: "Rua Pereira Doria",
4         numero: "541A",
5         complemento: "Apto. 135 C",
6         cidade: "São Caetano do Sul"
7     }
8 })
```

Tipo de dados – ObjectId

Um tipo especial de dado usado como identificador exclusivo para cada documento no MongoDB.

Todos os documentos inseridos em uma **collection** que não possuírem um atributo **_id** irão receber um **objectId** como padrão.

O valor gerado pelo **objectId** não se repete internamente na **collection**.

```
1 {  
2   _id: ObjectId('65e0c3f9a4ba4898b74167ab'),  
3   endereco: {  
4     logradouro: 'Rua Pereira Doria',  
5     numero: '541A',  
6     complemento: 'Apto. 135 C',  
7     cidade: 'São Caetano do Sul'  
8   }  
9 }
```



Tipo de dados – Data

Tipo de dado usado para armazenar data ou data e hora. Ele representa uma data como um timestamp de 64 bits, que contém o número de milissegundos desde 01 de janeiro de 1970, 00:00:00 UTC.

`new Date()` é uma sintaxe do JavaScript para criar um objeto Date. Ao executar essa instrução no shell do MongoDB, uma conversão automática para ISODate ocorrerá.

```
1 db.estudantes.insertOne({
2   data_cadastro: new Date("2030-10-20"),
3   ultima_atualizacao: new Date("2030-10-21 09:40:02"),
4   inicio_n1: ISODate("2030-11-28T21:10:00Z"),
5   inicio_n3: ISODate("2030-12-14T21:10:00Z")
6 })
```

Operações básicas - CRUD

Unidade 2



CREATE



READ



UPDATE



DELETE

C R U D

<https://www.mongodb.com/docs/mongodb-shell/crud/>

INserções

insertOne & InsertMany

❖ Inserção de um único documento por vez

Curiosidade: O limite máximo de tamanho de um documento no MongoDB é de 16MB. Esse limite nos permite inserir um documento com aproximadamente mais de 15 milhões de caracteres.

```
1 db.estudantes.insertOne({  
2   _id: 159753,  
3   nome: "Paula da Silva",  
4   nascimento: ISODate("2005-10-15")  
5 })
```

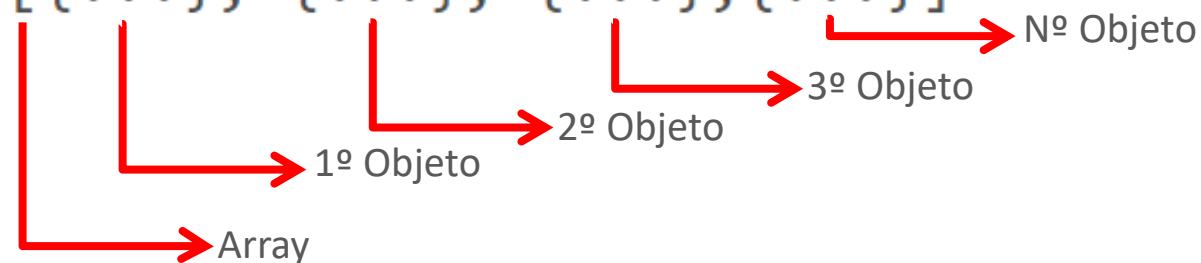

❖ Inclusão de um único documento por vez

```
1 db.estudantes.insertOne({
2     _id: 1,
3     nome: "Gustavo de Almeida",
4     nascimento: ISODate("2005-01-30"),
5     situacao: true,
6     disciplinas: ["Eng", "Alg", "NoSQL"],
7     media_geral: 8.4,
8     num_faltas: 15,
9     endereco: {
10         logradouro: "Rua dos Logaritmos",
11         numero: "124",
12         complemento: null,
13         cidade: "São Caetano do Sul"
14     }
15 })
```

❖ Inserção de múltiplos documentos de uma vez

Você pode inserir quantos documentos forem necessários, desde que não ultrapasse o limite por operação que é de 16MB.

```
1 db.estudantes.insertMany(  
2     [{...}, {...}, {...}, {...}]  
3 )
```



```
1 db.estudantes.insertMany([  
2     {_id: 159753, nome: "Daniel Augusto"},  
3     {_id: 159754, nome: "Bianca de Paula"},  
4     {_id: 159755, nome: "Sara Gomes"},  
5     {_id: 159756, nome: "Vinicius Bernardo"}  
6 ])
```

❖ Inclusão de múltiplos documentos por vez

```
1 db.estudantes.insertMany([
2     { _id: 2, nome: "Renato da Silva", ...},
3     { _id: 3, nome: "Antonio Felix", ...},
4     { _id: 4, nome: "Sara Santana", ...},
5     { _id: 5, nome: "Letícia Yuri", ...},
6     { _id: 6, nome: "Luis Henrique", ...},
7     { _id: 7, nome: "Delza Maria", ...}
8 ])
```

Exemplo apenas didático! Cada documento deve estar sintaticamente correto.

ATUALIZAÇÕES

UpdateOne & UpdateMany & ReplaceOne

❖ Atualização de documentos

Os seguintes métodos estão disponíveis para realizar atualizações em documentos do mongoDB.

- Atualização de um documento único:

```
db.collection.updateOne()
```

- Atualização de múltiplos documentos:

```
db.collection.updateMany()
```

- Substituição de um documento:

```
db.collection.replaceOne()
```

❖ Atualização de um documento

O método `updateOne()` necessita de dois documentos obrigatórios, sendo eles respectivamente: Critério de busca e Operação de atualização.

Um terceiro objeto pode ser passado, permitindo a configuração de parâmetros específicos.

```
1 db.estudantes.updateOne(  
2     {...}, → Critério de busca  
3     {...}, → Atualizações a serem realizadas  
4     {...} → Parâmetros adicionais opcionais  
5 )
```

❖ Atualização de um documento

Considerando que a collection estudantes possui um documento com o registro da estudante **'Paula da Silva'**:

```
1 db.estudantes.find()
2 [
3   {
4     _id: ObjectId('65e22586571c387e200d6738'),
5     nome: 'Paula da Silva',
6     nascimento: ISODate('2005-10-15T00:00:00.000Z')
7   },
8   {...},
9   {...}
10 ]
```

```
1 db.estudantes.updateOne(
2   {nome: "Paula da Silva"},
3   {
4     $set: {nome: "Paula da Silva Gomes", sexo: "Feminino"}
5   }
6 )
```

Operações básicas - CRUD

Unidade 2

❖ Atualização de um documento

```
1 db.estudantes.updateOne(  
2   {nome: "Paula da Silva"},  
3   {  
4     $set: {nome: "Paula da Silva Gomes", sexo: "Feminino"}  
5   }  
6 )
```

→ Critério de busca

→ Atualizações a serem realizadas

→ Operador de atualização

Atenção: Por estarmos utilizando o método `updateOne`, mesmo que o critério de seleção localize mais do que um documento, apenas o primeiro será atualizado!

MongoDB usa a notação de ponto (***dot notation***) para acessar os elementos de um documento incorporado.

```
1 db.estudantes.insertOne({
2   _id: 10,
3   nome: "Matheus Peres",
4   nascimento: ISODate("2006-02-22"),
5   endereco: {
6     logradouro: "Rua das Flores",
7     numero: "95",
8     cidade: "São Caetano do Sul"
9   }
10 })
11
12 db.estudantes.updateOne(
13   { nome: "Matheus Peres" },
14   { $set: { endereco.logradouro: "Rua das Lagrimas" } }
15 )
16
17 db.estudantes.find({ endereco.cidade: "Cidade X" })
```



<https://www.mongodb.com/docs/manual/core/document/#dot-notation>

❖ Operadores de atualização

```
1 db.estudantes.updateOne(  
2   {nome: "Paula da Silva"},  
3   {  
4     $set: {nome: "Paula da Silva Gomes"},  
5     $unset: {sexo: ""},  
6     ... ,  
7     ...  
8   }  
9 )
```

Conjunto de operações que
serão realizadas durante a
atualização do documento

Operadores:

\$set : Altera ou insere um atributo no documento;

\$unset: Exclui um atributo do documento.

❖ Operadores de atualização

O operador **\$currentDate** atualiza um atributo com a data e hora corrente.

```
1 db.estudantes.updateOne(  
2     {nome: "Paula da Silva"},  
3     { $currentDate: {atualizado_em: true} }  
4 )
```

❖ Operadores de atualização

O operador **\$inc** incrementa ou decremente o valor de um atributo.

- O operador **\$inc** aceita valores positivos e negativos.
- Se o campo não existir, o operador **\$inc** cria o atributo e atribui o valor especificado.
- O uso do operador **\$inc** em um campo com valor nulo gerará um erro.

```
1 db.estudantes.updateOne(  
2     { nome: "Paula da Silva" },  
3     { $inc: {  
4         nota_n1: -1.0,  
5         nota_n3: 3.2  
6     }  
7 }  
8 )
```

<https://www.mongodb.com/docs/manual/reference/operator/update/#fields>

❖ Operadores de atualização

O operador **\$min** irá alterar o valor do atributo mantendo o menor valor entre o valor já existente no documento e o valor determinado na instrução de update.

- Se o campo não existir, o operador **\$min** cria o atributo e atribuir o valor especificado.

Exemplos:

Considerando que a estudante “Paula da Silva” possui nota N3 igual a 6.4.



```
1 db.estudantes.updateOne(  
2   { nome: "Paula da Silva" },  
3   { $min: { nota_n3: 6.5 } }  
4 )
```

O valor 6.5 não foi atribuído pois é maior do que o valor presente no banco de dados



```
1 db.estudantes.updateOne(  
2   { nome: "Paula da Silva" },  
3   { $min: { nota_n3: 6.2 } }  
4 )
```

O valor 6.2 foi atribuído pois é menor do que o valor presente no banco de dados

❖ Operadores de atualização

O operador **\$max** irá alterar o valor do atributo mantendo o maior valor entre o valor já existente no documento e o valor determinado na instrução de update.

- Se o campo não existir, o operador **\$max** cria o atributo e atribuir o valor especificado.

Exemplos:

Considerando que a estudante “Paula da Silva” possui nota N3 igual a 6.4.



```
1 db.estudantes.updateOne(  
2   { nome: "Paula da Silva" },  
3   { $max: { nota_n3: 6.2 } }  
4 )
```

O valor 6.2 não foi atribuído pois é menor do que o valor presente no banco de dados



```
1 db.estudantes.updateOne(  
2   { nome: "Paula da Silva" },  
3   { $max: { nota_n3: 6.5 } }  
4 )
```

O valor 6.5 foi atribuído pois é maior do que o valor presente no banco de dados



Dica: Os operadores **\$min** e **\$max** podem ser utilizados com outros tipos de dados, como data por exemplo.

<https://www.mongodb.com/docs/manual/reference/operator/update/#fields>

❖ Operadores de atualização

O operador **\$mul** irá multiplicar o valor existente na base de dados em função do valor determinado no comando de update.

Exemplos:

Considerando que a estudante “Paula da Silva” possui nota N3 igual a 3.2.

```
1 db.estudantes.updateOne(  
2     { nome: "Paula da Silva" },  
3     { $mul: { nota_n3: 2 } }  
4 )
```

Após a execução da instrução de update acima o valor do atributo nota_n3 será alterado para 6.4.

❖ Operadores de atualização

Você pode executar várias multiplicações dentro do valor da chave **\$mul**, essa característica também se aplica aos outros operadores.

```
1 db.estudantes.updateOne(  
2     { nome: "Paula da Silva" },  
3     { $mul:  
4         {  
5             nota_n1: 2,  
6             nota_n2: 1.5  
7         }  
8     }  
9 )
```

<https://www.mongodb.com/docs/manual/reference/operator/update/#fields>

❖ Operadores de atualização




O operador **\$rename** é capaz de alterar o nome de um atributo, Atenção! Caso o nome de origem não exista no documento, nenhuma alteração será realizada.

```
1 db.estudantes.updateOne(  
2     { nome: "Paula da Silva" },  
3     { $rename: { "nota_n1": "n1" } }  
4 )
```

O comando acima alterou o nome do atributo “nota_n1” para “n1”.

❖ Atualização de um documento

Como mencionado anteriormente o método `updateOne()` pode possuir um terceiro objeto como parâmetro.

```
1 db.estudantes.updateOne(  
2     { ... },  Critério de busca  
3     { ... },  Atualizações a serem realizadas  
4     { upsert: true }  Parâmetros adicionais opcionais  
5 )
```

O parâmetro mais utilizado é o **upsert**, ele é utilizado para criar uma condicional lógica, caso nenhum documento seja encontrado com base no critério de seleção, um novo documento será criado levando em consideração o segundo objeto do comando.

❖ Operadores de atualização

O operador **\$setOnInsert** irá ser executado apenas quando um comando de update não localizar nenhum documento com base no critério de seleção.

O atributo “**observacao**” será criado no documento apenas se “**Renato Mendes**” não existir na collection “**estudantes**”

```
1 db.estudantes.updateOne(  
2     { nome: "Renato Mendes" },  
3     {  
4         $set: {nota_n1: 10.0 },  
5         $setOnInsert: {observacao: "Criado no update"}  
6     },  
7     { upsert: true }  
8 )
```

<https://www.mongodb.com/docs/manual/reference/operator/update/#fields>

❖ Operadores de atualização para **Arrays**

Os próximos operadores de atualização são exclusivos para operações em arrays, para exemplificar a utilização deles você deve considerar o seguinte documento presente em sua collection:

```
1 db.estudantes.insertOne({  
2     nome: "João Penedos",  
3     disciplinas: ["Eng", "Ágil", "NoSQL"]  
4 })
```

❖ Operadores de atualização

O operador **\$push** irá adicionar um novo elemento a um array. Atenção! O operador \$push não realiza nenhum controle sobre os valores a serem adicionados, sendo assim, o valor será duplicado caso o valor a ser adicionado já exista no array .

```
1 db.estudantes.updateOne(  
2   { nome: "João Penedos" },  
3   { $push: { disciplinas: "Log" } }  
4 )
```

❖ Operadores de atualização

O operador **\$pull** é responsável por remover os elementos de um array que satisfazerem o critério de seleção determinado.

```
1 db.estudantes.updateOne(  
2   { nome: "João Penedos" },  
3   { $pull: { disciplinas: "Eng" } }  
4 )
```

Você pode utilizar qualquer operador de seleção disponível no MongoDB.

❖ Operadores de atualização

O operador **\$pullAll** é responsável por remover todas as instâncias presentes no array do seu documento em função de todos elementos presentes na instrução de update.

```
1 db.estudantes.updateOne(  
2   { nome: "João Penedos" },  
3   { $pullAll: { disciplinas: ["Ágil", "NoSQL"]} }  
4 )
```

❖ Operadores de atualização

O operador **\$pop** é utilizado para remover o primeiro (-1) ou o último elemento (1) de um array.

Considerando o seguinte cenário:

```
1 {  
2   nome: "João Penedos",  
3   disciplinas: ["Eng", "Ágil", "NoSQL", "Alg"]  
4 }
```

→ Documento Original

```
1 db.estudantes.updateOne(  
2   { nome: "João Penedos" },  
3   { $pop: { disciplinas: -1 } }  
4 )
```

→ Atualização

```
1 {  
2   nome: "João Penedos",  
3   disciplinas: ["Ágil", "NoSQL", "Alg"]  
4 }
```

→ Resultado Final



❖ Operadores de atualização

O operador **\$pop** é utilizado para remover o primeiro (-1) ou o último elemento (1) de um array.

Considerando o seguinte cenário:

```
1 {  
2   nome: "João Penedos",  
3   disciplinas: ["Ágil", "NoSQL", "Alg"]  
4 }
```

→ Documento Original

```
1 db.estudantes.updateOne(  
2   { nome: "João Penedos" },  
3   { $pop: { disciplinas: 1 } }  
4 )
```

→ Atualização

```
1 {  
2   nome: "João Penedos",  
3   disciplinas: ["Ágil", "NoSQL"]  
4 }
```

→ Resultado Final

❖ Operadores de atualização

O operador **\$addToSet** irá adicionar um novo elemento a um array caso seu valor ainda não esteja presente.

✓

```
1 db.estudantes.updateOne(  
2     { nome: "João Penedos" },  
3     { $addToSet: { disciplinas: "Alg" } }  
4 )
```

!

```
1 db.estudantes.updateOne(  
2     { nome: "João Penedos" },  
3     { $addToSet: { disciplinas: ["Alg", "Estat"] }}  
4 )
```

Um array com dois elementos será adicionado no array disciplinas.


disciplinas: ["Eng" , "Ágil" , "NoSQL" , ["Alg" , "Estat"]]



❖ Operadores de atualização

O operador **\$addToSet** pode ser utilizado em conjunto com o operador **\$each**.

```
1 db.estudantes.updateOne(  
2   { nome: "João Penedos" },  
3   { $addToSet: {  
4       disciplinas: { $each: ["Alg", "Estat"] }  
5   }  
6 }  
7 )
```



Cada elemento do array é tratado como um elemento único,
aplicando as regras de inserção do operador \$addToSet
individualmente.

❖ Atualização de múltiplos documentos

O método `updateMany()` possui funcionamento muito similar ao `updateOne()`, porém, todos os documentos que atenderem seu critério de busca irão ser atualizados.

```
1 db.estudantes.updateMany(  
2     { nota_n1: 5.0 },  
3     { $set: { nota_n1: 9.5 } }  
4 )
```

Neste exemplo todos os documentos que possuírem nota N1 igual a 5.0 serão atualizados.

❖ Substituição de um documento inteiro

O método `replaceOne()` tem por finalidade substituir um documento sem alterar seu identificador único (`_id`).

```
1 db.estudantes.replaceOne(  
2   { _id: 10 },  
3   { nome: "Jorge Peres",  
4     nascimento: ISODate("2016-02-22"),  
5     endereco: {  
6       logradouro: "Rua das Margaridas",  
7       numero: "125",  
8       cidade: "São Caetano do Sul"  
9     }  
10  }  
11 )
```

Atenção!!! Não é possível indicar um novo `_id` no segundo parâmetro, isso porque o `_id` é imutável.

EXCLUSÕES

DeleteOne & DeleteMany

❖ Exclusão de um único documento por vez

```
1 db.estudantes.deleteOne( { nome: "João Penedos" } )
```

❖ Exclusão de múltiplos documentos por vez

```
1 db.estudantes.deleteMany( { nota_n3: 6.0 } )
```

CONSULTAS

Find()

❖ Consulta de documentos

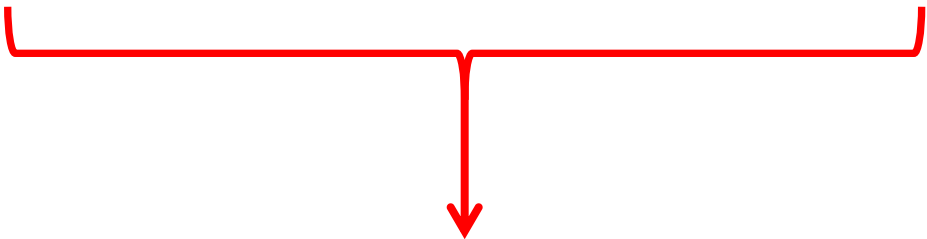
O método `find()` pode receber até três documentos opcionais, sendo eles respectivamente: critério de busca, projeção e configuração de parâmetros específicos.

```
1 db.estudantes.find(  
2   { ... }, → Critério de busca  
3   { ... }, → Projeção  
4   { ... } → Parâmetros adicionais  
5 );
```

❖ Consulta de documentos – Critérios de Busca

O primeiro documento do método `find()` permite que você determine condicionais lógicas que serão utilizadas para filtra um subconjunto de documentos desejados.

```
1 db.estudantes.find(  
2     {nome: "Jorge Peres"}  
3 )
```



Esse critério irá selecionar todos os estudantes existentes na collection **“estudantes”** com valor do atributo **“nome”** como **“Jorge Peres”**

❖ Consulta de documentos – Projeção

O segundo documento do método `find()` permite que você determine quais atributos deverão ser exibidos.

```
1 db.estudantes.find(  
2     {nome: "Marcelo Rocha"},  
3     {nome: true, sexo: 1}  
4 )
```

→ Critério de busca

→ Projeção

Atenção:

1. Você pode utilizar o valor **true** ou **1** para determinar se um atributo deve ser exibido e **false** ou **0** para ocultar.
2. O MongoDB é Case-Sensitive.
3. **Não crie o caos em seu código!** Padronize tudo com **true** ou **1** e seja feliz =]



❖ Consulta de documentos – Projeção

O resultado padrão da sua consulta será os atributos escolhidos para projeção mais o atributo `_id`, caso não deseje exibir o `_id`, inclua ele na projeção com o valor **false** ou **0**.

```
1 db.estudantes.find(  
2     {nome: "Marcelo Rocha"},  
3     {_id: false, nome: true, sexo: true}  
4 )
```

❖ Consulta de documentos – Ordenação

O resultado de uma consulta pode ser ordenado por um ou mais atributos presentes nos documentos por meio do método `sort()`.

Ordem ascendente

```
1 db.estudantes.find(  
2   { },  
3   { _id: false, nome: true }  
4 ).sort({ nome: 1 })
```



```
1 [  
2   { nome: 'Alexandre Martins' },  
3   { nome: 'Aline Oliveira' },  
4   { nome: 'Amanda Ferreira' },  
5   { nome: 'Ana Costa' },  
6   { nome: 'André Lima' },  
7   { nome: 'Beatriz Carvalho' },  
8   { nome: 'Camila Nunes' },  
9   { nome: 'Carla Martins' },  
10  { nome: 'Carolina Silva' },  
11  { nome: 'Carolina Silva' },  
12  ...  
13 ]
```

Ordem descendente

```
1 db.estudantes.find(  
2   { },  
3   { _id: false, nome: true }  
4 ).sort({ nome: -1 })
```



```
1 [  
2   { nome: 'Vinicius Pereira' },  
3   { nome: 'Vanessa Gonçalves' },  
4   { nome: 'Tatiane Sousa' },  
5   { nome: 'Roberto Silva' },  
6   { nome: 'Roberto Santos' },  
7   { nome: 'Ricardo Lima' },  
8   { nome: 'Renata Almeida' },  
9   { nome: 'Rafael Ferreira' },  
10  { nome: 'Rafael Costa' },  
11  { nome: 'Pedro Santos' },  
12  ...  
13 ]
```

❖ Consulta de documentos – Ordenação

O método `sort()` permite a organização do resultado em ordem ascendente utilizando o valor 1 ou descendente com o valor -1. É possível ordenar o resultado da consulta usando vários níveis de ordenação.

```
1 db.estudantes.find(  
2     { peso: { $gte: 70, $lte: 80} },  
3     {_id: false, nome: true, sexo: true, peso: true}  
4 ).sort({sexo: 1, nome: 1})
```

```
1 [  
2   { nome: 'Carla Martins', sexo: 'F', peso: 70 },  
3   { nome: 'Laura Santos', sexo: 'F', peso: 71 },  
4   { nome: 'Luana Oliveira', sexo: 'F', peso: 70 },  
5   { nome: 'Renata Almeida', sexo: 'F', peso: 72 },  
6   { nome: 'André Lima', sexo: 'M', peso: 80 },  
7   { nome: 'Daniel Alves', sexo: 'M', peso: 79 },  
8   { nome: 'Diego Silva', sexo: 'M', peso: 77 },  
9   { nome: 'João Silva', sexo: 'M', peso: 80 },  
10  { nome: 'Lucas Oliveira', sexo: 'M', peso: 78 },  
11  { nome: 'Lucas Oliveira', sexo: 'M', peso: 79 },  
12  { nome: 'Lucas Pereira', sexo: 'M', peso: 75 },  
13  { nome: 'Marcelo Rocha', sexo: 'M', peso: 73 },  
14  { nome: 'Marcos Ferreira', sexo: 'M', peso: 76 },  
15  { nome: 'Rafael Ferreira', sexo: 'M', peso: 78 },  
16  { nome: 'Ricardo Lima', sexo: 'M', peso: 77 }  
17 ]
```

→ Agrupamento feminino e nomes em ordem crescente

→ Agrupamento masculino e nomes em ordem crescente

❖ Consulta de documentos – Limit

O método `limit()` permite controlar a quantidade máxima de documentos que devem ser retornados na consulta.

```
1 db.estudantes.find(  
2     { },  
3     { _id: false, nome: true, sexo: true }  
4 ).sort({ nome: 1 }).limit(5)
```

```
1 [  
2   { nome: 'Alexandre Martins', sexo: 'M' },  
3   { nome: 'Aline Oliveira', sexo: 'F' },  
4   { nome: 'Amanda Ferreira', sexo: 'F' },  
5   { nome: 'Ana Costa', sexo: 'F' },  
6   { nome: 'André Lima', sexo: 'M' }  
7 ]
```

❖ Consulta de documentos – Operadores de Comparação

- **\$gt:** "greater than" - "maior que".
- **\$gte:** "greater than or equal to" - "maior que ou igual a".
- **\$lt:** "less than" - "menor que".
- **\$lte:** "less than or equal to" - "menor que ou igual a".
- **\$eq:** "equal" - "igual a".
- **\$ne:** "not equal" - "diferente de".

❖ Consulta de documentos – Operadores de Comparação

- **\$eq:** "equal" - "igual a". Seleciona documentos onde o valor do campo é igual ao valor especificado. É o operador padrão quando você simplesmente especifica um valor para um campo.

```
1 db.estudantes.find(  
2     {nome: "Marcelo Rocha"}  
3 )
```



```
1 db.estudantes.find({  
2     nome: {$eq: "Marcelo Rocha"}  
3 })
```

❖ Consulta de documentos – Operadores lógicos

- **\$and:** Este operador combina múltiplas expressões em uma única expressão lógica e retorna os documentos que atendem a todas as condições especificadas.

```
1 db.estudantes.find(  
2     { $and : [{sexo: "M"}, {peso: 80}]},  
3     {nome: true, sexo: true, peso: true}  
4 )
```



```
1 db.estudantes.find(  
2     {sexo: "M", peso: 80},  
3     {nome: true, sexo: true, peso: true}  
4 )
```

❖ Consulta de documentos – Operadores lógicos

- **\$or:** Este operador combina múltiplas expressões em uma única expressão lógica e retorna os documentos que atendem a pelo menos uma das condições especificadas.

```
1 db.estudantes.find(  
2     { $or : [{sexo: "M"}, {peso: 80}]},  
3     {nome: true, sexo: true, peso: true}  
4 )
```

❖ Consulta de documentos – Operadores lógicos

- **\$not:** Este operador inverte a lógica de uma expressão e retorna os documentos que não atendem à condição especificada.

```
1 db.estudantes.find(  
2     { peso: {$not: {$eq: 80}} },  
3     { nome: true, sexo: true, peso: true }  
4 )
```

❖ Consulta de documentos – Operadores lógicos

- **\$nor:** Este operador combina múltiplas expressões em uma única expressão lógica e retorna os documentos que não atendem a nenhuma das condições especificadas.

```
1 db.estudantes.find(  
2     { $nor: [{ peso: 70 }, { peso: 80 } ] },  
3     { nome: true, sexo: true, peso: true }  
4 )
```

❖ Consulta de documentos – Operadores lógicos

- **\$in:** Este operador é útil quando você precisa fazer consultas com uma lista de valores possíveis para um determinado campo.

No exemplo abaixo serão exibidos todos estudantes que possuírem peso igual a 70 ou igual a 80 ou igual a 75.

```
1 db.estudantes.find(  
2     { peso: { $in: [70, 80, 75] } },  
3     { nome: true, sexo: true, peso: true }  
4 )
```

❖ Consulta de documentos – Operadores lógicos

- **\$nin:** Este operador é utilizado quando você deseja desconsiderar uma lista de valores possíveis, ele é o inverso do operador \$in.

No exemplo abaixo serão exibidos todos estudantes que não possuírem peso igual a 70 ou igual a 80 ou igual a 75.

```
1 db.estudantes.find(  
2     { peso: { $nin: [70, 80, 75] } },  
3     { nome: true, sexo: true, peso: true }  
4 )
```

❖ Consulta de documentos – Operadores lógicos

- **\$exists:** Este operador verifica se um atributo existe ou não dentro de um documento.

```
1 db.estudantes.find(  
2     { telefone: { $exists: true } },  
3     { nome: true, sexo: true, peso: true }  
4 )
```


CONSULTAS EM ARRAYS

Find()

❖ Consulta de documentos – Operadores lógicos

- **\$elemMatch:** Este operador irá validar buscas complexas em subdocumentos.

Para exemplificar a utilização do **\$elemMatch** você deverá considerar o seguinte documento presente na collection “**estudantes**”.

```
1 { _id: 50,  
2   nome: "Mariana Costa",  
3   sexo: "F",  
4   peso: 66,  
5   nascimento: ISODate("2000-06-12"),  
6   disciplinas: [  
7     {nome: "Engenharia de Soft", carga_horaria: 80},  
8     {nome: "Métodos Ágeis", carga_horaria: 40},  
9     {nome: "NoSQL", carga_horaria: 80},  
10  ]  
11 }
```



❖ Consulta de documentos – \$elemMatch

```
1 { _id: 50,  
2   nome: "Mariana Costa",  
3   sexo: "F",  
4   peso: 66,  
5   nascimento: ISODate("2000-06-12"),  
6   disciplinas: [  
7     {nome: "Engenharia de Soft", carga_horaria: 80},  
8     {nome: "Métodos Ágeis", carga_horaria: 40},  
9     {nome: "NoSQL", carga_horaria: 80},  
10  ]  
11 }
```

```
1 db.estudantes.find(  
2   { "disciplinas.nome" : "Métodos Ágeis",  
3     "disciplinas.carga_horaria" : 80},  
4   { nome: true, disciplinas: true }  
5 )
```

Ao realizarmos a consulta abaixo utilizando o **Dot Notation** o documento da “Mariana Costa” será exibido e talvez não seja isso que você está desejando.



Isso ocorre pois o **dot notation** está procurando qualquer documento que possua um subdocumento que dentro de seu array “disciplinas” possua um atributo “nome” igual a “Métodos Ágeis” e um atributo “carga_horaria” igual a 80, não necessariamente dentro do mesmo elemento do array.

❖ Consulta de documentos – \$elemMatch

```
1 { _id: 50,  
2   nome: "Mariana Costa",  
3   sexo: "F",  
4   peso: 66,  
5   nascimento: ISODate("2000-06-12"),  
6   disciplinas: [  
7     {nome: "Engenharia de Soft", carga_horaria: 80},  
8     {nome: "Métodos Ágeis", carga_horaria: 40},  
9     {nome: "NoSQL", carga_horaria: 80},  
10  ]  
11 }
```

```
1 db.estudantes.find(  
2   { "disciplinas.nome" : "Métodos Ágeis",  
3     "disciplinas.carga_horaria" : 80},  
4   { nome: true, disciplinas: true }  
5 )
```

Diagram illustrating the \$elemMatch query. A yellow arrow points from the query condition `"disciplinas.carga_horaria" : 80` to the corresponding element in the `disciplinas` array: `{nome: "Engenharia de Soft", carga_horaria: 80}`. Another yellow arrow points from the query condition `{ nome: true, disciplinas: true }` to the `disciplinas` array.

Note que os valores existem, porém eles não pertencem ao mesmo elemento do array.



❖ Consulta de documentos – \$elemMatch

Devemos usar o operador **\$elemMatch** quando desejamos por exemplo localizar todos os estudantes que estão cursando a disciplina de Métodos Ágeis com carga horário de 40 horas

```
1 db.estudantes.find({
2   disciplinas: {
3     $elemMatch: {
4       nome: "Métodos Ágeis",
5       carga_horaria: 40
6     }
7   }
8 })
```

```
1 { _id: 50,
2   nome: "Mariana Costa",
3   sexo: "F",
4   peso: 66,
5   nascimento: ISODate("2000-06-12"),
6   disciplinas: [
7     {nome: "Engenharia de Soft", carga_horaria: 80},
8     {nome: "Métodos Ágeis", carga_horaria: 40},
9     {nome: "NoSQL", carga_horaria: 80},
10  ]
11 }
```



❖ Consulta de documentos – Operadores lógicos

- **\$all:** Este operador permitir encontrar um conjunto de valores em um array, não importando a ordem dos elementos.

Para exemplificar a utilização do **\$all** você deverá considerar o seguinte documento presente na collection “**estudantes**”.

```
1 db.estudantes.insertOne({
2   _id: 100,
3   nome: "Mariano Silva",
4   emails: ["mc@gmail.com",
5            "silva@outlook.com",
6            "silva.m@abc.com",
7            "msilva@def.com"]
8 })
```

❖ Consulta de documentos – Operadores lógicos

- \$all

```
1 db.estudantes.find(  
2   { emails: { $all: [ "mc@gmail.com", "msilva@def.com" ] } }  
3 )
```

Essa instrução é equivalente ao comando:

```
1 db.estudantes.find(  
2   { $and: [ { emails: "mc@gmail.com" }, { emails: "msilva@def.com" } ] }  
3 )
```

❖ Consulta de documentos – Operadores lógicos

- **\$size:** Este operador permitir verificar a quantidade de elementos em um array.

```
1 db.estudantes.find(  
2     { emails: { $size: 4 } }  
3 )
```


❖ Collections

Excluir uma collection:

```
db.estudantes.drop()
```

Renomear uma collection

```
db.estudantes.renameCollection("students")
```

❖ Banco de dados

```
db.dropDatabase()
```

Atenção: Não é possível renomear um banco de dados por meio de um método. Você deve considerar uma estratégia de backup e restore!

Dúvidas?

