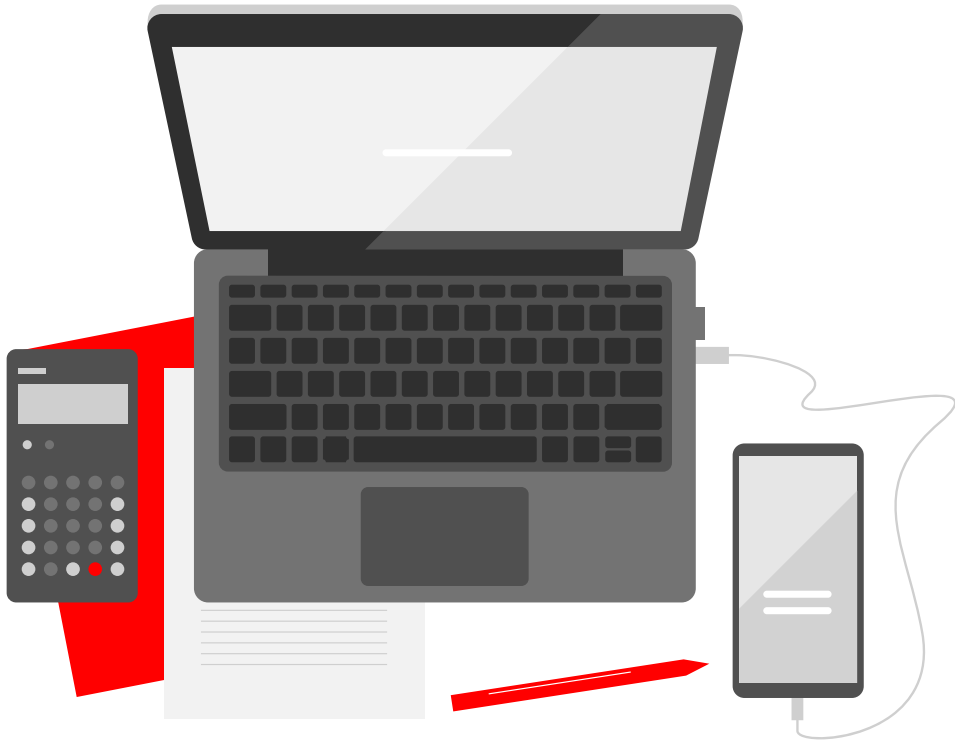
Banco de Dados
Não Relacional



mongoDB

Desenvolvimento de
Sistemas Multiplataformas

CRUD MongoDB

Independentemente de estar utilizando um **banco relacional ou não-relacional**, o aprendizado de uma nova tecnologia de persistência de dados sempre envolve iniciar pelo aprendizado do CRUD, acrônimo para:

- **C**reate (criar),
- **R**ead (ler),
- **U**pdate (atualizar)
- **D**elete (excluir), as quatro operações elementares com dados.

Enquanto que nos bancos relacionais existe a linguagem SQL e os conjuntos de comandos DDL e DML (Data Definition Language e Data Manipulation Language, respectivamente), no MongoDB nós temos um conjunto de funções com sintaxe similar à linguagem JavaScript, para realizar todas operações.

CRUD MongoDB

Além de colocar um servidor MongoDB pra rodar, executamos alguns comandos muito simples e úteis:

- **show databases**:
  - Para listar as bases de dados existentes no servidor;
- **use <nome_da_database>:**
  - Para se 'conectar' a uma base específica;
- **show collections**:
  - Para listar as coleções de documentos de uma base;
- **db.<nome_da_colecao>.find():**
  - Para retornar todos documentos de uma coleção;
- **db.<nome_da_colecao>.insert(<objeto_json>):**
  - Para inserir um ou mais documentos em uma coleção;
- **db.<nome_da_colecao>.insertOne(<objeto_json>):**
  - Para inserir um único documento em uma coleção e retornar o seu id;

CRUD MongoDB vs SQL

SQL	MongoDB - NoSQL
Select	find e findOne
Insert	insertOne e insertMany
Delete	remove
Update	update, replaceOne, updateOne
Top, Limit (dependendo do fabricante)	limit
Order By	sort
CreateIndex	createIndex
Like	find(expressão regular)
Create Table	Não precisa, a coleção é criada no momento que inserimos um documento
Create Database	Não precisa, é criada na primeira coleção criada

CRUD MongoDB vs SQL

SQL	MongoDB - NoSQL
Count	count
Group By	aggregate
Distinct	distinct

Insert

Considerando uma coleção customers (clientes), o uso do comando insertOne (precedido por db.<nome_da_collection>), como um objeto JSON sendo passado por parâmetro, faz com que o mesmo e torne um documento da referida coleção:

```
Enterprise workshop> db.clientes.insertOne({nome:"Maria Eduarda", idade:22})
{
  acknowledged: true,
  insertedId: ObjectId('65e50e4d334e07bb97397fb0')
}
```

A execução deste comando retorna **o id do documento inserido** (insertedId). Este campo '_id' único e auto incremental é um ObjectId, um objeto alfanumérico longo e complexo, e será sua chave primária.

Caso você deseje ter controle da chave primária, deverá passar o campo _id (com o valor que quiser) juntamente com seu objeto.

Insert

Além do insert de um documento passado por parâmetro, MongoDB permite realizar inserções em lote passando um array de documentos, como abaixo (o uso de colchetes indica um array de objetos, enquanto cada par de chaves indica um objeto):
Criando um ARRAY, para que possa ser inserido ao MongoDB através da função **insertMany(nomeArray)**

nomeArray = [{*chave:valor, chave:valor, chave:valor*}, {*chave:valor, chave:valor, chave:valor*}]

```
Enterprise workshop> custArray = [{nome:"Victoria Silva", idade:28, cidade:"Registro"},{nome:"Julio Cesar Rodrigues", id
ade:35, cidade:"São Caetano", estado:"SP"}]
[
  { nome: 'Victoria Silva', idade: 28, cidade: 'Registro' },
  {
    nome: 'Julio Cesar Rodrigues',
    idade: 35,
    cidade: 'São Caetano',
    estado: 'SP'
  }
]
```

Insert

Inserindo documentos criados no array usando o **db.nomeColecao.insertMany(nomeArray)**

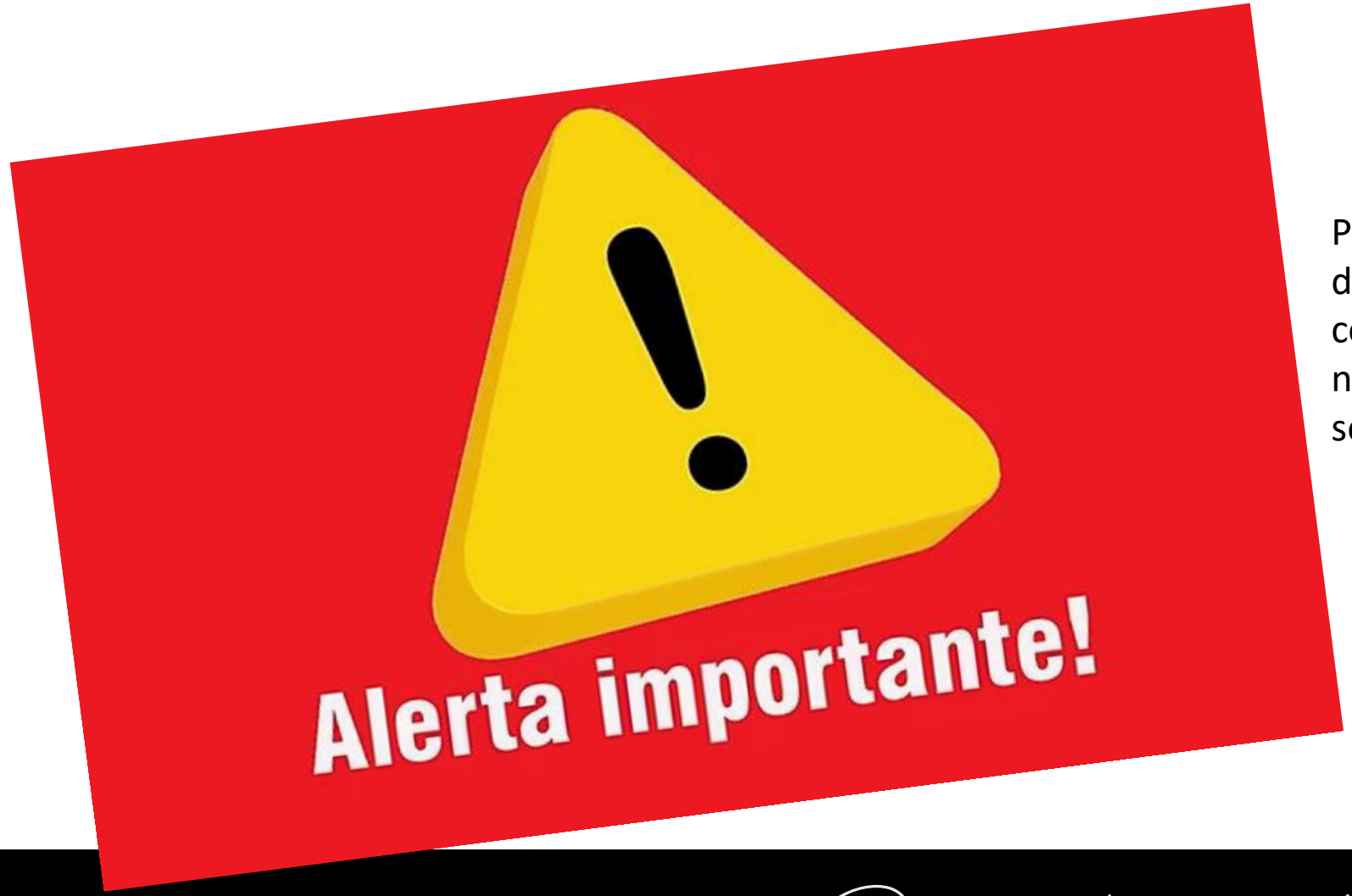
```
Enterprise workshop> db.clientes.insertMany(custArray)
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('65e51057334e07bb97397fb3'),
    '1': ObjectId('65e51057334e07bb97397fb4')
  }
}
```

CRUD MongoDB

Insert

```
Enterprise workshop> arrayClientes = [{nome:"Antonio Henrique Guedes", idade:28},{nome:"Sabrina Rodrigues", idade:32, cidade:"Iguape",estado:"SP"},{nome:"Izadora Pompeu", idade:21, email:"izapompeu123@outlool.com"}]
[
  { nome: 'Antonio Henrique Guedes', idade: 28 },
  {
    nome: 'Sabrina Rodrigues',
    idade: 32,
    cidade: 'Iguape',
    estado: 'SP'
  },
  {
    nome: 'Izadora Pompeu',
    idade: 21,
    email: 'izapompeu123@outlool.com'
  }
]
Enterprise workshop> db.clientes.insertMany(arrayClientes)
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('65e51285334e07bb97397fb5'),
    '1': ObjectId('65e51285334e07bb97397fb6'),
    '2': ObjectId('65e51285334e07bb97397fb7')
  }
}
```

CRUD MongoDB



Para o nome dos campos dos seus documentos e até mesmo para o nome das coleções do seu banco, use o padrão de nomes JavaScript (camel-case, sem acentos, sem espaços dentro do nome, etc).

Filtrar Dados

Filtrar uma consulta baseado em um campo do documento, a função find() pode receber um documento por parâmetro representando o filtro a ser aplicado sobre a coleção para retornar documentos específicos.

db.nomeColecao.find({chave:valor})

```
Enterprise workshop> db.clientes.find({cidade:"Iguape"})
[
  {
    _id: ObjectId('65e51285334e07bb97397fb6'),
    nome: 'Sabrina Rodrigues',
    idade: 32,
    cidade: 'Iguape',
    estado: 'SP'
  }
]
```

Para que haja o retorno da pesquisa, os parâmetros da pesquisa deverão ser escritos como foram registrados, obedecendo maiúsculo e minúsculo.

Filtrar Dados

Filtro independente da ordem das chave:valor

```
Enterprise workshop> db.clientes.find({cidade:"Iguape",nome:"Sabrina Rodrigues"})
[
  {
    _id: ObjectId('65e51285334e07bb97397fb6'),
    nome: 'Sabrina Rodrigues',
    idade: 32,
    cidade: 'Iguape',
    estado: 'SP'
  }
]
```

Filtro realizado com valor escrito diferente que como foi cadastrado.

```
Enterprise workshop> db.clientes.find({cidade:"IGUAPE",nome:"Sabrina Rodrigues"})

Enterprise workshop>
```

Filtrar Dados

- **findOne**
  - Além do find comum, o MongoDB possui uma variante que é o findOne.
    A única diferença prática em relação ao find normal é que o findOne retorna somente a primeira ocorrência de documento que atenda aos filtros.
    Para consultas que você só vai usar um documento do retorno, o ideal é usar o findOne, visto que ele tende a ser mais rápido pois ao encontrar um documento que atenda aos filtros, ele pára de buscar no restante da coleção.

Filtrar Dados

find({chave:valor}) - Comum

```
Enterprise workshop> db.clientes.find({estado:"SP"})
[
  {
    _id: ObjectId('65e51057334e07bb97397fb4'),
    nome: 'Julio Cesar Rodrigues',
    idade: 35,
    cidade: 'São Caetano',
    estado: 'SP'
  },
  {
    _id: ObjectId('65e51285334e07bb97397fb6'),
    nome: 'Sabrina Rodrigues',
    idade: 32,
    cidade: 'Iguape',
    estado: 'SP'
  }
]
```

findOne({chave:valor}) – Usando One

```
Enterprise workshop> db.clientes.findOne({estado:"SP"})
{
  _id: ObjectId('65e51057334e07bb97397fb4'),
  nome: 'Julio Cesar Rodrigues',
  idade: 35,
  cidade: 'São Caetano',
  estado: 'SP'
}
```

Filtrar Dados

- **Filter Operators**
  - Além de campos com valores literais, o parâmetro do find permite usar uma infinidade de operadores.
  - Cada operador possui um funcionamento próprio e todos eles são compostos de uma palavra e iniciam com o símbolo $.
  - Um deles, bastante utilizado em buscas com campos de texto é o operador **$regex** que permite, por exemplo, trazer todos documentos que possuam a letra '**a**' no nome:

Filtrar Dados

db.nomeColecao.find({chave:{$regex:/a/i}})

Ou através do atalho abaixo, que faz exatamente a mesma coisa:

db.nomeColecao.find({chave: /a/i })

```
Enterprise workshop> db.clientes.find({nome:{$regex:/a/i}})
[
  {
    _id: ObjectId('65e505f0334e07bb97397fad'),
    nome: 'Wagner Toth',
    idade: 43
  },
  {
    _id: ObjectId('65e508c0334e07bb97397faf'),
    nome: 'João da Silva',
    idade: 34
  },
  {
    _id: ObjectId('65e50e4d334e07bb97397fb0'),
    nome: 'Maria Eduarda',
    idade: 22
  },
  {
    _id: ObjectId('65e51057334e07bb97397fb3'),
    nome: 'Victoria Silva',
    idade: 28,
    cidade: 'Registro'
  },
  {
    _id: ObjectId('65e51057334e07bb97397fb4'),
    nome: 'Julio Cesar Rodrigues',
    idade: 35,
    cidade: 'São Caetano',
```

Filtrar Dados

Mas e se eu quiser trazer todos os cliente maiores 35 anos? Use o operador **$gte**:

O operador **$gte (Greater Than or Equal)** retorna todos os documentos que possuam o campo idade e que o valor do mesmo seja igual ou superior à 35.

db.nomeColecao.find({idade: {$gte: 35}})

```
Enterprise workshop> db.clientes.find({idade:{$gte:35}})
[
  {
    _id: ObjectId('65e505f0334e07bb97397fad'),
    nome: 'Wagner Toth',
    idade: 43
  },
  {
    _id: ObjectId('65e51057334e07bb97397fb4'),
    nome: 'Julio Cesar Rodrigues',
    idade: 35,
    cidade: 'São Caetano',
    estado: 'SP'
  }
]
```

Filtrar Dados

Podemos facilmente combinar filtros absolutos e operadores usando vírgulas dentro do documento passado por parâmetro, assim como citado anteriormente:

db. clientes.find({nome: "Luiz", idade: {$gte: 18}})

O que a expressão acima irá retornar?

Se você disse ***clientes cujo nome sejam Luiz e que sejam maiores de idade***, você **acertou!**

E a expressão abaixo?

db.clientes.find({nome: { $regex: /a/i }, idade: {$gte: 18}})

Clientes cujo nome contenham a letra '**a**' e que sejam maiores de idade, é claro!

Filtrar Dados

Alguns operadores que você pode usar junto ao filtro do find são:

- **$eq**: exatamente igual (=)
- **$ne**: diferente (<> ou !=)
- **$gt**: maior do que (>)
- **$lt**: menor do que (<)
- **$lte**: menor ou igual a (<=)
- **$in**: o valor está contido em um array de possibilidades, como em um OU.
- **$all**: MongoDB permite campos com arrays.

Filtrar Dados

O operador **$or** permite que você crie mais de um filtro para seu find, podendo ser completamente diferente um do outro e serão retornados os documentos que atendam qualquer um deles ou até mesmo os dois:

db.nomeColecao.find({$or: [{chave:valor}, {chave: valor}]})

```
Enterprise workshop> db.clientes.find({$or: [{idade: 32}, {nome: "Wagner Toth"}]})
[
  {
    _id: ObjectId('65e505f0334e07bb97397fad'),
    nome: 'Wagner Toth',
    idade: 43
  },
  {
    _id: ObjectId('65e51285334e07bb97397fb6'),
    nome: 'Sabrina Rodrigues',
    idade: 32,
    cidade: 'Iguape',
    estado: 'SP'
  }
]
```

Filtrar Dados

Existe uma variação deste operador que é o **$nor,** utilizado quando temos um array de filtros que os documentos que vão ser retornados **NÃO devem atender** (***ao contrário do $or***).

E por fim, o operador **$not** serve para negar o resultado de uma outra expressão de filtro, ou seja, algo que seria verdadeiro na consulta, se tornará falso.

Não é algo muito recomendado de se utilizar por questões práticas de lógica (negações costumam gerar confusão nas discussões) e por questões de performance também, pois negações sempre exigem full scan na coleção.

Filtrar Dados - Personalizado

Existem diversas funções e opções super úteis de serem adicionadas em finds visando personalizar o resultado dos mesmos.

Por exemplo, temos no MongoDB duas funções para fazer paginação de resultados: **skip e limit**:

db.nomeColecao.find().skip(1).limit(10)

```
Enterprise workshop> db.clientes.find().skip(1).limit(4)
[
  {
    _id: ObjectId('65e508c0334e07bb97397faf'),
    nome: 'João da Silva',
    idade: 34
  },
  {
    _id: ObjectId('65e50e4d334e07bb97397fb0'),
    nome: 'Maria Eduarda',
    idade: 22
  },
  {
    _id: ObjectId('65e51057334e07bb97397fb3'),
    nome: 'Victoria Silva',
    idade: 28,
    cidade: 'Registro'
  },
  {
    _id: ObjectId('65e51057334e07bb97397fb4'),
    nome: 'Julio Cesar Rodrigues',
    idade: 35,
    cidade: 'São Caetano',
    estado: 'SP'
  }
]
```

Filtrar Dados - Personalizado

db.clientes.find().skip(1).limit(4)

No exemplo retornaremos quatro clientes (limit) ignorando o primeiro existente na coleção (skip).

```
Enterprise workshop> db.clientes.find().skip(1).limit(4)
[
  {
    _id: ObjectId('65e508c0334e07bb97397faf'),
    nome: 'João da Silva',
    idade: 34
  },
  {
    _id: ObjectId('65e50e4d334e07bb97397fb0'),
    nome: 'Maria Eduarda',
    idade: 22
  },
  {
    _id: ObjectId('65e51057334e07bb97397fb3'),
    nome: 'Victoria Silva',
    idade: 28,
    cidade: 'Registro'
  },
  {
    _id: ObjectId('65e51057334e07bb97397fb4'),
    nome: 'Julio Cesar Rodrigues',
    idade: 35,
    cidade: 'São Caetano',
    estado: 'SP'
  }
]
```

Filtrar Dados - Personalizado

E para ordenar? Usamos a função **sort** no final de todas as outras, com um documento indicando quais campos e se a ordenação por aquele campo é **crescente (1)** ou **decrescente (-1).**

Ordem Crescente

Db.nomeColecao.find().sort({chave: 1})

Ordem Decrescente

Db.nomeColecao.find().sort({chave: -1})

```
Enterprise workshop> db.clientes.find().skip(1).limit(3).sort({idade:1})
[
  {
    _id: ObjectId('65e50e4d334e07bb97397fb0'),
    nome: 'Maria Eduarda',
    idade: 22
  },
  {
    _id: ObjectId('65e51285334e07bb97397fb5'),
    nome: 'Antonio Henrique Guedes',
    idade: 28
  },
  {
    _id: ObjectId('65e51057334e07bb97397fb3'),
    nome: 'Victoria Silva',
    idade: 28,
    cidade: 'Registro'
  }
]
```

Filtrar Dados - Personalizado

Agora, se quer saber quantos documentos serão retornados por uma consulta, aplique a função count após o find:

db.nomeColecao.find({chave:valor}).count()

```
Enterprise workshop> db.clientes.find({estado:"SP"}).count()
2
Enterprise workshop>
```

Obrigado!!!!