



INSTITUTO FEDERAL  
PIAUÍ  
Campus Parnaíba

# MongoDB - Operações Básicas

Prof. Msc Denival A. dos Santos

# Introdução

- Não existe uma linguagem separada para descrever as operações de CRUD no MongoDB.
- As operações existem como métodos/funções dentro da API.

CRUD	SQL	MongoDB
Create	Insert	Insert
Read	Select	Find
Update	Update	Update
Delete	Delete	Remove

# Identificador

- Cada documento deve conter um **\_id** único associado, especificado pelo campo `_id`.
- Caso o usuário não especifique um valor para esse campo, ele é gerado automaticamente pelo MongoDB, definido como um `ObjectId()`.
- O MongoDB cria também um índice para o campo `_id`, a fim de tornar as consultas mais eficientes.
- Podemos especificar um `_id` (que será único na coleção)

```
db.pessoas.insert({_id:1, a:1})
```

# Inserindo um documento

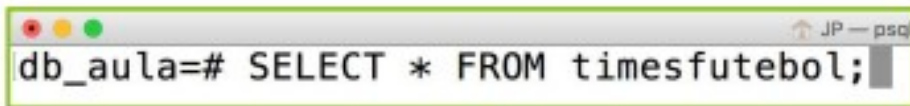
- Existem duas operações de inserção no MongoDB.
  - Inserção de um único documento:
    - insertOne.
  - Inserção de múltiplos documentos de uma só vez:
    - insertMany.
- Sintaxe



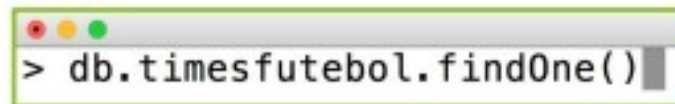
```
JP — mongo — 80x24
> db.timesfutebol.insertOne({"_id": 1, "nome": "Cruzeiro", "pais": "Brasil"})
{ "acknowledged" : true, "insertedId" : 1 }
> db.timesfutebol.insertMany([{"nome": "Barcelona", "pais": "Espanha"}, {"nome": "Palmeiras", "mundial": 0}])
{
  "acknowledged" : true,
  "insertedIds" : [
    ObjectId("5911b28009702042baa255e2"),
    ObjectId("5911b28009702042baa255e3")
  ]
}
```

# Localizando documentos

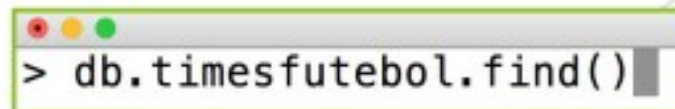
- O MongoDB possui dois métodos principais para retornar informações de documentos.
  - O método `find()` retorna um ponteiro para todos os documentos que atendem aos critérios especificados.
  - O método `findOne()` retorna um único documento que atende aos critérios especificados. Caso exista mais de um documento atendendo aos critérios, o método `findOne()` retorna apenas o primeiro.



```
db_aula=# SELECT * FROM timesfutebol;
```



```
> db.timesfutebol.findOne()
```

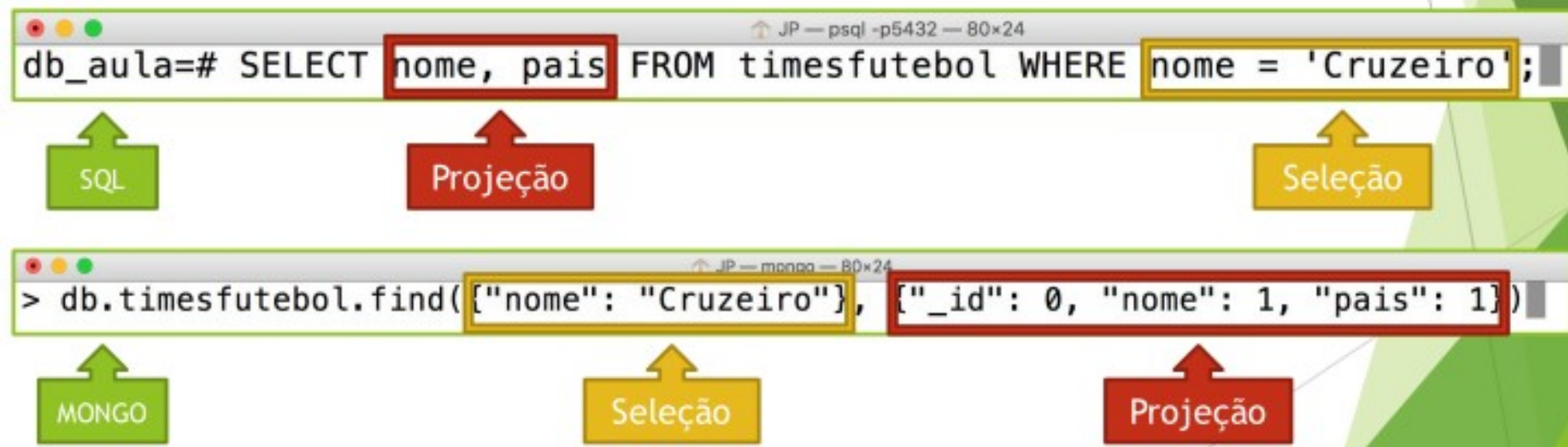


```
> db.timesfutebol.find()
```



# Localizando documentos

- Ambos os métodos find() e findOne() permitem especificar, da mesma forma, critérios de seleção e projeção para o resultado.



# Operadores

- **\$gt**: verifica se um atributo é maior que um valor
- **\$gte**: verifica se um atributo é maior ou igual que um valor
- **\$lt**: verifica se um atributo é menor que um valor
- **\$lte**: verifica se um atributo é menor ou igual que um valor
- **\$exists**: verifica se um atributo existe
- **\$type**: verifica se existe um atributo de um tipo determinado
- **\$regex**: se o atributo corresponde à expressão regular
- **\$or**: compara duas condições com o operador ou
- **\$and**: compara duas condições com o operador and
- **\$in**: verifica se um atributo contém um dos valores de um array
- **\$all**: verifica se um atributo contém todos os valores de um array

# Operadores

**\$gt, \$gte, \$lt, \$lte:** {campo: {\$operator:valor}}

```
db.notas.find({nota:{$gt:95})  
db.notas.find({nota:{$gte:95, $lte:98},  
tipo:"ejercicio"})  
db.notas.find({score:{$lt:95})
```



# Operadores \$or

**\$or**: retorna todos os documentos que atendam a uma duas condições.

```
db.estoque.find({$or: [{qtde: {$lt: 20}},  
  {preço: 10}]})
```

# Operadores \$and

**\$and**: retorna todos os documentos que atendam às duas condições.

```
//explicitamente (reparem no [])
db.pessoas.find( { $and: [ { nome: { $gt: "C" } },
{ nome: { $regex: "a" } } ] } );

//implicitamente
db.pessoas.find( { nome: { $gt: "C", $regex: "a" } });

//explicitamente (reparem no [])
db.notas.find( { $and: [ { nota: { $gt: 50 } },
{ tipo: "prova" } ] } );

//implicitamente
db.notas.find( { nota: { $gt: 50 }, tipo: "prova" } );
```

# Modificar documentos

- O MongoDB possui três métodos (update, updateOne, updateMany) para atualização de dados em um documento.
- Os métodos **updateOne()** e **updateMany()** localizam o documento segundo os critérios especificados e fazem as alterações descritas.
  - Diferença: quantidade de documentos afetada.
  - Enquanto o updateOne() afeta somente um documento que atenda os critérios, o updateMany() afeta todos.
- O método replaceOne() localiza um único documento que atenda aos critérios especificados e o substitui por um novo documento.
  - O atributo \_id do documento permanece o mesmo.

```
db.<nome-da-collection>.update(  
    {<critérioDeBusca1>:<valor1>,...},  
    {<campoParaAtualizar1>:<novoValor1>,...})  
));
```

# Modificar documentos

- Ejemplo

```
db.carro.updateOne(  
  {modelo: "Siena"},  
  {$set: {modelo: "Grand Siena"}}  
)
```

# Deletar documentos

- O MongoDB possui dois métodos para a remoção de documentos.
- Os métodos `deleteOne()` e `deleteMany()` localizam o documento segundo os critérios especificados e o removem da base de dados.
  - Diferença: quantidade de documentos afetada.
  - Enquanto o `deleteOne()` afeta somente um documento que atenda os critérios, o `deleteMany()` afeta todos.

**SQL** → JP — psql -p5432 — 80×24

```
db_aula=# DELETE FROM timesfutebol WHERE nome = "Palmeiras";
```

← Seleção

**MONGO** → JP — mongo — 80×24

```
> db.timesfutebol.deleteOne({"nome": "Palmeiras"})
{ "acknowledged" : true, "deletedCount" : 1 }
> db.timesfutebol.deleteMany({})
{ "acknowledged" : true, "deletedCount" : 2 }
```

← Seleção

← Seleção

# Operadores \$exists e \$type

**\$exists:** retorna os documentos que contenha o atributo especificado.

```
db.pessoas.find({"profissão":{"$exists:true"}});
```

**\$type:** retorna os documentos cujo atributo seja do tipo especificado (de acordo com a notação BSON).

```
db.pessoas.find({nome:{$type:2}});
```

# Operadores \$regex

**\$regex**: retorna os documentos que satisfazem a expressão regular

```
// nomes que possuam a letra a em qualquer posição  
// Em SQL... WHERE (nome like %a%)  
db.pessoas.find({"nome":{"$regex":"a"}});  
  
// nomes que terminem com a  
// Em SQL... WHERE (nome like %A)  
db.pessoas.find({nome:{'$regex':'a\\$'}});  
  
// nomes que comecem com A  
// Em SQL... WHERE (nome like A%)  
db.pessoas.find({nome:{'$regex':'^A'}});
```

# Operadores \$or

**\$or**: retorna todos os documentos que atendam a uma duas condições.

```
db.estoque.find({$or: [{qtde: {$lt: 20}},  
  {preço: 10}]})
```



# Operadores \$or and \$and

Combinando **\$or** e **\$and**:

```
db.estoque.find( {  
  $and : [  
    { $or : [{preço: 0.99},{preço: 1.99}]},  
    { $or : [ {a_venda : true},  
      { qtde:{$lt:20}}] }  
  ]  
} )
```

# Ordenação

Protótipo: `db.<collection>.find(<condição>).sort(<ordenação>)`

`<ordenação>`: {nomecampo: direção}

Direção:

- ASC: 1
- DESC: -1

```
db.notas.find().sort({"estudante":1})
db.notas.find().sort({"estudante":-1})
db.notas.find({tipo:"prova", nota:{$gte:50}},
{_id:0, estudante:1}).sort({"estudante":-1})
```

# Contagem

**.count()**: conta o número de documentos retornados na consulta

```
db.pessoas.count()  
db.pessoas.find().count()  
db.pessoas.find({"nome": "Ana"}).count()  
db.pessoas.find({"profissão": "Estudante"}).count()
```

# Distintos

- \$distinct
  - db.carros.distinct("marca")

# Renomear

**\$rename** é utilizado para renomear atributos.

É possível passar uma lista de atributos a serem renomeados.

```
db.students.insert({_id:1, name:"Ana",  
nickname: "Aninha", celular:"99999-9999"});  
db.students.update( { _id: 1 },  
{ $rename: { 'nickname': 'alias', 'cell': 'mobile' } });
```

# Outros operadores

Como poderíamos fazer buscas em favoritos (array)?

**\$in**: retorna todos os documentos cujo atributo contenha pelo menos um dos valores especificados no array

- **\$nin**: seleciona documentos em que o valor do campo especificado não está no array ou documentos que não possuam o campo
- **\$eq**: seleciona documentos em que o valor do campo é igual ao valor especificado.
- **\$ne**: seleciona documentos em que o valor é diferente (*not equal*) do valor especificado
- **\$not**: operador NOT, seleciona documentos que não satisfazem a expressão especificada, incluindo documentos que não contenham o atributo especificado

# Busca em arrays

Como poderíamos fazer buscas em favoritos (array)?

**\$in**: retorna todos os documentos cujo atributo contenha pelo menos um dos valores especificados no array

**\$all**: retorna todos os documentos cujo atributo contenha todos os valores especificados no array

# Busca em arrays

```
// todos os documentos que tenham pizza como favorito  
db.pessoas.find({favoritos:"pizza"});  
// retorna João, Pedro e Luís
```

```
// todos os documentos que contenham pizza  
// OU refrigerante como favorito  
db.pessoas.find({favoritos:  
{$in:["pizza","refrigerante"]}});  
// retorna João, Pedro e Luís
```

```
// todos os documentos que contenham pizza  
// E refrigerante como favorito  
db.pessoas.find({favoritos:  
{$all:["pizza","refrigerante"]}});//Pedro e Luís
```



# Agregação

- **\$min**

```
db.carros.aggregate([  
    {  
        $group:{_id:"$marca", minimo:{$min:"$valor"}}  
    }  
])
```

- **\$max**

```
db.carros.aggregate([  
    {  
        $group:{_id:"$marca", maximo:{$max:"$valor"}}  
    }  
])
```

# Agregação

- **\$avg**

```
db.carros.aggregate([  
    {  
        $group:{_id:"$marca", media:{$avg:"$valor"}}  
    }  
])
```

- **\$sum**

```
db.carros.aggregate([  
    {  
        $group:{_id:"$marca", soma:{$sum:1}}  
    }  
])
```

# Operações matemáticas

- **\$add (soma)**

```
db.carros.aggregate([  
    {  
        $project:{marca:1, soma:{$add:["$valor",10]}}  
    }  
])
```

- **\$multiply (multiplicação)**

```
db.carros.aggregate([  
    {  
        $project:{marca:1, multiplicação:{$multiply:["$valor",10]}}  
    }  
])
```

# Operações matemáticas

- **\$subtract (subtração)**

```
db.carros.aggregate([
    {
        $project:{marca:1, subtracao:{$subtract:["$valor",10]}}
    }
])
```

- **\$divide (divisão)**

```
db.carros.aggregate([
    {
        $project:{marca:1, divisao:{$divide:["$valor",10]}}
    }
])
```

# Relacionamentos no MongoDB

- O MongoDB não implementa integridade referencial e nem operações de junção.
  - Logo, não existe o conceito de chave estrangeira para documentos.
- Existem duas maneiras de se expressar relacionamentos entre documentos no MongoDB.
  - **Referências entre documentos:** é possível guardar o `_id` de um documento como um atributo em outro documento. Não é o mesmo que guardar uma chave estrangeira.
  - **Documentos embutidos:** o MongoDB permite guardar um documento inteiro como um atributo em um documento (Sub-Documents).

# Relacionamentos no MongoDB

- Referencia entre documento (normalizado) - Um para muitos

```
// inserir post
> db.posts.insert({titulo:'Analise e Projeto de Sistemas'});
> var post = db.posts.findOne({titulo:'Analise e Projeto de
Sistemas'});

// inserir comentario
> db.comentarios.insert({
  nome:'Jefferson',
  corpo:'Awo Boer, eu nao tenho mais APS!',
  post_id : post._id
});
> var comentario = db.comentarios.findOne({nome:'Jefferson'});

> db.comentarios.find({post_id: post._id})
{
  "_id" : ObjectId("4d5955f1e0ab4d700255d83e"),
  "nome" : "Jefferson",
  "corpo" : "Awo Boer, eu nao tenho mais APS!",
  "post_id" : ObjectId("4d5955a5e0ab4d700255d83d")
}
```

# Relacionamentos no MongoDB

- Documentos embutidos (Embedded) - um para muitos

```
-- inserir posts, comentários
db.posts.insert({
  titulo: 'Aula',
  comentarios: [
    {nome: 'Diego', corpo: 'Reprovei'},
    {nome: 'Antonio', corpo: 'Passei!'}
  ]
})
```

```
db.posts.find({"comentarios.nome": "Antonio"})
// ou
db.posts.find({
  comentarios: {
    $elemMatch: {nome: 'Antonio'}
  }
})
```

# Usuários

- Por padrão o MongoDB fornece acesso o servidor sem autenticação.
- Para a criação de um novo usuário para uma base de dados utilizamos o comando:

Db.createUser()

- Sintaxe

```
{ user: "<name>",  
  pwd: "<cleartext password>",  
  customData: { <any information> },  
  roles: [  
    { role: "<role>", db: "<database>" } | "<role>",  
    ...  
  ]  
}
```



# Usuários

- Os campos nome, senha e roles são obrigatórios, já o campo customData é opcional.
- Em roles iremos informar quais os privilégios para esse usuário e em qual database ele irá ter tais privilégios.
- Todas as roles estão descritas no manual do MongoDB no link <https://docs.mongodb.com/manual/reference/built-in-roles/#built-in-roles>.
- Para criar um usuário sem roles, basta passar um vetor em vazio.
- As informações de usuários ficam armazenadas na base de dados admin.
- Para acessar com usuário criado:

**mongo -u nombreUsuarioAdmin -p --authenticationDatabase base\_Dados**

# Usuários

- Para pesquisar os usuários existente temos o comando:

`db.system.users.find()`

- Exemplo

```
use admin
db.createUser(
  {
    user: "aluno",
    pwd: "aluno",
    roles: [{role:"readWrite", db:"aula" }]
  }
)
```

- Para remover um usuário utilizamos

`db.dropUser("aluno")`

# Backup

- Backups são fundamentais e são parte de um eficiente plano de **disaster recovery**. Mais do que apenas fazer uma cópia do arquivo, um bom backup garante a integridade dos dados, funcionamento em mais de um sistema e está alocado em diversos lugares.
- O **mongodump** é utilizado para exportar o conteúdo de sua base de dados, no formato BSON (representação binária de estruturas de dados - Binary JSON).

- Sintaxe

mongodump

- Comandos:

mongodump --out c:\backup16/1/18

mongodump --collections carros --db leilao -out c:\backup

mongodump --host mongodb1.example.net --port 3017 --username user  
--password "pass" --out /opt/backup/mongodump-2013-10-24

# Restauração

- O **mongorestore** é utilizado em conjunto com o mongodump para restaurar (“importar”) o arquivo BSON, recriando os índices criados na base que foi exportada. Com ele é possível criar uma nova base de dados ou adicionar os dados a uma base já existente.

- Comandos

mongorestore c:\backup16/1/18

mongorestore --collection carros --db leilao C:\Users\denival\dump\leilao\carros.bson (Refaz a estrutura)

mongorestore --collection testado --db testando C:\Users\denival\dump\leilao\carros.bson (neste caso cria uma nova coleção)