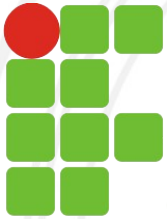
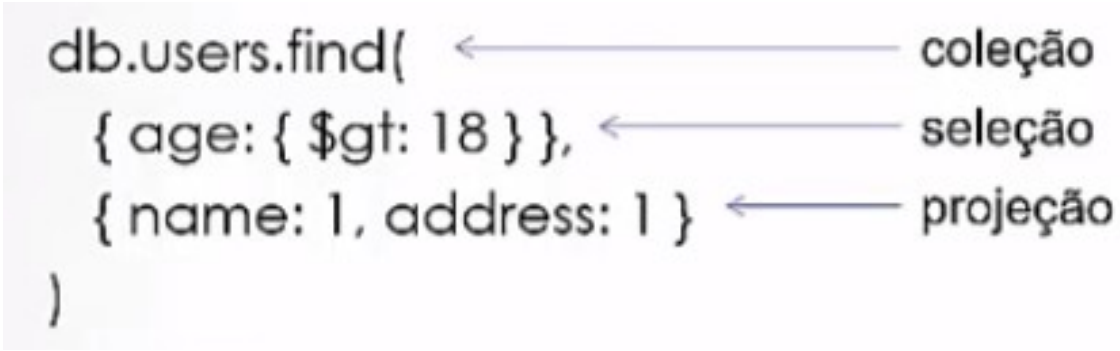


INSTITUTO FEDERAL
PIAUÍ
Campus Parnaíba

MongoDB – Find (continuação)

Prof. Msc Denival A. dos Santos

Find() - Leitura de dados



- Deve referir-se a uma coleção especifica.
- Pode incluir a seleção (documentos a serem exibidos).
- Pode incluir a projeção (campos a serem exibidos).

Find() - Projeção de dados

```
{ "_id" : ObjectId("5c9d203959dc3195b757c8ad"), "marca" : "vw", "modelo" : "Fusca", "ano" : 1970 }
{ "_id" : ObjectId("5c9d203959dc3195b757c8ae"), "marca" : "fiat", "modelo" : "147", "ano" : 1985 }
{ "_id" : ObjectId("5c9d1ede59dc3195b757c8ac"), "marca" : "Chevrolet", "modelo" : "Prisma", "ano" : 2018 }
{ "_id" : ObjectId("5c9d1e9959dc3195b757c8ab"), "marca" : "Hyundai", "modelo" : "HB20", "ano" : 2019 }
```

- Projeção de dados no mongoDB significa selecionar apenas os dados necessários ao invés de selecionar todo os dados de um documento.
- Para limitar quais os campos que deseja exibir você precisa setar a lista de campos com os valores 1 ou 0.
  - 1 é usado para mostrar o campo,
  - 0 é usado para esconder o campo.

```
> db.carro.find({},{_id:0, marca:1, modelo:1})
{ "marca" : "Hyundai", "modelo" : "HB20" }
{ "marca" : "Chevrolet", "modelo" : "Prisma" }
{ "marca" : "vw", "modelo" : "Fusca" }
{ "marca" : "fiat", "modelo" : "147" }
```

Find() - Seleção de dados

```
db.inventory.insertMany([
   { item: "journal", qty: 25, size: { h: 14, w: 21, uom: "cm" }, status: "A" },
   { item: "notebook", qty: 50, size: { h: 8.5, w: 11, uom: "in" }, status: "A" },
   { item: "paper", qty: 100, size: { h: 8.5, w: 11, uom: "in" }, status: "D" },
   { item: "planner", qty: 75, size: { h: 22.85, w: 30, uom: "cm" }, status: "D" },
   { item: "postcard", qty: 45, size: { h: 10, w: 15.25, uom: "cm" }, status: "A" }
]);
```

- A seleção lista todos os documentos que atendam a uma condição.
- Para exibir todos os documentos: db.inventory.find({}).
- Para especificar uma condição utilizamos <campo>:<valor>.
- Exemplo:

```
db.inventory.find( { status: "D" } )
```

```
SELECT * FROM inventory WHERE status = "D"
```

Find() - ordenação

- .sort(<campo>:<valor>)
- Para ordenar os dados que desejamos exibir utilizamos:
  - ASC: 1 - usado para mostrar em ordem crescente,
  - DESC: -1 - usado para esconder o decrescente

```
> db.carro.find().sort({ano:1})
{ "_id" : ObjectId("5c9d203959dc3195b757c8ad"), "marca" : "vw", "modelo" : "Fusca", "ano" : 1970 }
{ "_id" : ObjectId("5c9d203959dc3195b757c8ae"), "marca" : "fiat", "modelo" : "147", "ano" : 1985 }
{ "_id" : ObjectId("5c9d1ede59dc3195b757c8ac"), "marca" : "Chevrolet", "modelo" : "Prisma", "ano" : 2018 }
{ "_id" : ObjectId("5c9d1e9959dc3195b757c8ab"), "marca" : "Hyundai", "modelo" : "HB20", "ano" : 2019 }
```

```
> db.carro.find().sort({ano:-1})
{ "_id" : ObjectId("5c9d1e9959dc3195b757c8ab"), "marca" : "Hyundai", "modelo" : "HB20", "ano" : 2019 }
{ "_id" : ObjectId("5c9d1ede59dc3195b757c8ac"), "marca" : "Chevrolet", "modelo" : "Prisma", "ano" : 2018 }
{ "_id" : ObjectId("5c9d203959dc3195b757c8ae"), "marca" : "fiat", "modelo" : "147", "ano" : 1985 }
{ "_id" : ObjectId("5c9d203959dc3195b757c8ad"), "marca" : "vw", "modelo" : "Fusca", "ano" : 1970 }
```

Find() - busca de substring

- **/a/i - contendo uma substring em qualquer parte da string.**
  - Exemplos:

```
db.pessoa.find({nome:/ld/i})
```

- **/^a/i - string que começa uma substring qualquer.**
  - **Exemplo:**

```
db.pessoa.find({nome:/^a/i})
```

- **/a$/i - string que termina com uma substring qualquer**
  - **Exemplo:**

```
db.pessoa.find({nome:/do$/i})
```

Find() - Operadores relacionais

- Para comparação de diferentes valores do tipo BSON, utilizamos:
  - **$eq** – corresponde a valores iguais a um valor especifico.
  - **$ne** – corresponde a valores que não são iguais a um valor.
  - **$gt** – corresponde a valores que são maiores que um valor.
  - **$gte** – corresponde a valores que são maiores ou iguais a um valor.
  - **$lt** – corresponde a valores que são menores que um valor.
  - **$lte** – corresponde a valores que são menores ou iguais a um valor.
  - **$in** – corresponde a um valor especificado em uma matriz.
  - **$nin** – corresponde a valores não especificados em uma matriz.

Find() - Operadores de comparação

- **$eq - {campo:valor} ou {<campo:{$eq:valor}}**
  - Exemplos:

```
db.carro.find({ano:1985})
```

```
db.carro.find({ano:{$eq:1985}})
```

- **$ne**
  - **Exemplo:**

```
db.carro.find({ano:{$ne:1985}})
```

Find() - Operadores de comparação

- **$gt**
  - Exemplo:

```
db.carro.find({ano:{$gt:2000}})
```

- **$gte**
  - **Exemplo:**

```
db.carro.find({ano:{$gte:2000}})
```

Find() - Operadores de comparação

- $lt
  - Exemplo:

```
db.carro.find({ano:{$lt:2000}})
```

- $lte
  - **Exemplo:**

```
db.carro.find({ano:{$lte:2000}})
```

Find() - Operadores lógicos

- Para comparações lógicas utilizamos:
  - **$and - E**
  - **$or - Ou**
  - **$not - Negação**

Find() - Operadores lógicos

- **$or**
  - Exemplo:

```
db.Vendas.find(
  {
   $or: [
          {
            Qtde: { $eq:4 }
          },
          {
             Qtde: {$eq:5}
           }
        ]
    });
```

Find() - Operadores lógicos

- **$and**
  - Exemplo:

```
db.Vendas.find(
  {
   $and: [
          {
            Qtde: { $eq:4 }
          },
          {
             Cliente: {$eq:'Henrique'}
           }
        ]
    });
```

Find() - Operadores lógicos

- **$not**
  - Exemplo:

```
db.Vendas.find({Qtde: {$not: {$eq:1} }});
```

Operadores $exists

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

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

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'}});
```

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-Documentos).

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>",
    ...
  ]
}
```


https://docs.mongodb.com/manual/reference/method/db.updateUser/

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.getUsers()

- Exemplo

```
use admin
db.createUser(
    {
      user:"denival",
      pwd:"1234",
      roles:[ "readWrite", { role:"readWrite", db:"leilao" } ],
      passwordDigestor:"server"
    }
)
```

- Para remover um usuário utilizamos

```
use leilao
db.dropUser("teste", {w: "majority", wtimeout: 5000})
```

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:\backup**16/1/18**

  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)