

**#4 – Instalação e Testes**

Diversos players de cloud computing fornecem versões de Mongo hospedadas e prontas para uso como [Umbler](https://www.luiztools.com.br/umbler) e [mLab](http://www.mlab.com/), no entanto é muito importante um conhecimento básico de administração local de MongoDB para entender melhor como tudo funciona. Não focaremos aqui em nenhum aspecto de segurança, de alta disponibilidade, de escala ou sequer de administração avançada de MongoDB. Deixo todas estas questões para você ver junto à documentação oficial no site oficial, onde inclusive você pode estudar e tirar as certificações.

Caso ainda não tenha feito isso, acesse o [site oficial do MongoDB](http://www.mongodb.org/) e baixe gratuitamente a versão mais recente para o seu sistema operacional, que é a versão 3.4 na data em que escrevo este artigo.

Baixe o arquivo compactado e, no caso do Windows, rode o executável que extrairá os arquivos na sua pasta de Arquivos de Programas (não há uma instalação de verdade, apenas extração de arquivos), seguido de uma pasta server/versão, o que é está ok para a maioria dos casos, mas que eu prefiro colocar em C:\Mongo ou dentro de Applications no caso do Mac.

Dentro dessa pasta do Mongo podem existir outras pastas, mas a que nos interessa é a pasta bin. Nessa pasta estão uma coleção de utilitários de linha de comando que são o coração do MongoDB (no caso do Windows, todos terminam com .exe):

* mongod: inicializa o servidor de banco de dados;
* mongo: inicializa o cliente de banco de dados;
* mongodump: realiza dump do banco (backup binário);
* mongorestore: restaura dumps do banco (restore binário);
* mongoimport: importa documentos JSON ou CSV pro seu banco;
* mongoexport: exporta documentos JSON ou CSV do seu banco;
* entre outros.

Para subir um servidor de MongoDB na sua máquina é muito fácil: execute o utilitário mongod via linha de comando  como abaixo, onde dbpath é o caminho onde seus dados serão salvos (esta pasta já deve estar criada).



|  |  |
| --- | --- |
| 1 | C:\mongo\bin> mongod --dbpath C:\mongo\data |

Isso irá iniciar o servidor do Mongo. Uma vez que apareça no prompt “[initandlisten] waiting for connections on port 27017”, está pronto, o servidor está executando corretamente e você já pode utilizá-lo, sem segurança alguma e na porta padrão 27017.

**Nota:** se já existir dados de um banco MongoDB na pasta data, o mesmo banco que está salvo lá ficará ativo novamente, o que é muito útil para os nossos testes.

Agora abra outro prompt de comando (o outro ficará executando o servidor) e novamente dentro da pasta bin do Mongo, digite:



|  |  |
| --- | --- |
| 1 | c:\mongo\bin> mongo |

Após a conexão funcionar, se você olhar no prompt onde o servidor do Mongo está rodando, verá que uma conexão foi estabelecida e um sinal de “>” indicará que você já pode digitar os seus comandos e queries para enviar à essa conexão.

Ao contrário dos bancos relacionais, no MongoDB você não precisa construir a estrutura do seu banco previamente antes de sair utilizando ele. Tudo é criado conforme você for usando, o que não impede, é claro, que você planeje um pouco o que pretende fazer com o Mongo.

O comando abaixo no terminal cliente mostra os bancos existentes nesse servidor:



|  |  |
| --- | --- |
| 1 | > show databases |

Se é sua primeira execução ele deve listar as bases admin e local. Não usaremos nenhuma delas. Agora digite o seguinte comando para “usar” o banco de dados “workshop” (um banco que você sabe que não existe ainda):



|  |  |
| --- | --- |
| 1 | > use workshop |

O terminal vai lhe avisar que o contexto da variável “db” mudou para o banco workshop, que nem mesmo existe ainda (mas não se preocupe com isso!). Essa variável “db” representa agora o banco workshop e podemos verificar quais coleções existem atualmente neste banco usando o comando abaixo:



|  |  |
| --- | --- |
| 1 | > show collections |

Isso também não deve listar nada, mas não se importe com isso também. Assim como fazemos com objetos JS que queremos chamar funções, usaremos o db para listar os documentos de uma coleção de customers (clientes) da seguinte forma:



|  |  |
| --- | --- |
| 1 | > db.customers.find() |

find é a função para fazer consultas no MongoDB e, quando usada sem parâmetros, retorna todos os documentos da coleção. Obviamente não listará nada pois não inserimos nenhum documento ainda, o que vamos fazer agora com a função insert:



|  |  |
| --- | --- |
| 1 | > db.customers.insert({ nome: "Luiz", idade: 29 }) |

A função insert espera um documento JSON por parâmetro com as informações que queremos inserir, sendo que além dessas informações o MongoDB vai inserir um campo \_id automático como chave primária desta coleção.

Como sabemos se funcionou? Além da resposta ao comando insert (nInserted indica quantos documentos foram inseridos com o comando), você pode executar o find novamente para ver que agora sim temos customers no nosso banco de dados. Além disso se executar o “show collections” e o “show databases” verá que agora sim possuímos uma coleção customers e uma base workshop nesse servidor.

Tudo foi criado a partir do primeiro insert e isso mostra que está tudo funcionando bem no seu servidor MongoDB!

E assim encerra a primeira parte do nosso tutorial de MongoDB para iniciantes. Na parte 2, [que você confere neste link](https://www.luiztools.com.br/post/tutorial-mongodb-para-iniciantes-em-nosql-2/), falo dos comandos elementares do MongoDB (CRUD) e de modelagem orientada à documentos.

# Tutorial MongoDB para iniciantes em NoSQL – Parte 2

MongoDB Logo

***Atualizado em 23/11/2017!***

Na [parte anterior deste tutorial](https://www.luiztools.com.br/post/tutorial-mongodb-para-iniciantes-em-nosql/) sobre MongoDB para iniciantes em NoSQL, falei sobre os conceitos mais elementares deste banco de dados, sobre quando usar e quando não usar esta tecnologia, quais as principais diferenças dele para outros bancos de dados e deixamos o servidor e o cliente prontos para receber comandos, sendo que inclusive executamos alguns para testar tudo.

Se não fez a primeira parte do tutorial, [faça](https://www.luiztools.com.br/post/tutorial-mongodb-para-iniciantes-em-nosql/), no mínimo a seção final onde configuramos o ambiente. Continuaremos exatamente de onde a última parte parou.

Nesta segunda parte, falaremos dos comandos elementares (CRUD) do MongoDB. Veremos:

1. [Insert Avançado](https://www.luiztools.com.br/post/tutorial-mongodb-para-iniciantes-em-nosql-2/#1)
2. [Find Avançado](https://www.luiztools.com.br/post/tutorial-mongodb-para-iniciantes-em-nosql-2/#2)
3. [Update](https://www.luiztools.com.br/post/tutorial-mongodb-para-iniciantes-em-nosql-2/#3)
4. [Delete](https://www.luiztools.com.br/post/tutorial-mongodb-para-iniciantes-em-nosql-2/#4)

Caso prefira assistir a um vídeo ao invés de ler o longo post, o vídeo abaixo é uma aula gratuita do meu [curso de Node.js e MongoDB](https://www.luiztools.com.br/curso-nodejs). Aproveite!

Vamos lá!

### #1 – Insert “Avançado”

Na seção anterior aprendemos a fazer um find() que retorna todos os documentos de uma coleção e um insert que insere um novo documento em uma coleção, além de outros comandos menores. Agora vamos adicionar mais alguns registros no seu terminal cliente  mongo:



|  |  |
| --- | --- |
| 1  2 | > custArray = [{ nome : "Fernando", idade : 29 }, { nome : "Teste", "uf" : "RS" }]  > db.customers.insert(custArray) |

**Atenção:**para o nome dos campos dos seus documentos e até mesmo para o nome das coleções do seu banco, use o padrão de nomes de variáveis JS (camel-case, sem acentos, sem espaços, etc).

**Nota:** no exemplo acima a variável custArray passa a existir durante toda a sessão do terminal a partir do comando seguinte.

Nesse exemplo passei um array com vários documentos para nossa função insert inserir na coleção customers e isso nos trás algumas coisas interessantes para serem debatidas. Primeiro, sim, você pode passar um array de documentos por parâmetro para o insert. Segundo, você notou que o segundo documento não possui “idade”? E que ele possui um campo “uf”?

### #2 – Find “avançado”

Para se certificar que todos documentos foram realmente inseridos na coleção, use o seguinte comando:



|  |  |
| --- | --- |
| 1 | > db.customers.find().pretty() |

É o mesmo comando find() que usamos anteriormente, mas com a função pretty() no final para identar o resultado da função no terminal, ficando mais fácil de ler. Use e você vai notar a diferença, principalmente em consultas com vários resultados.

Mas voltando à questão do “uf”, ao contrário dos bancos relacionais, o MongoDB possui schema variável, ou seja, se somente um customer tiver “uf”, somente ele terá esse campo, não existe um schema pré-definido compartilhado entre todos os documentos, cada um é independente. Obviamente considerando que eles compartilham a mesma coleção, é interessante que eles possuam coisas em comum, caso contrário não faz sentido guardar eles em uma mesma coleção.

Mas como fica isso nas consultas? E se eu quiser filtrar por “uf”? Não tem problema!

Essa é uma boa deixa para eu mostrar como filtrar um find() por um campo do documento:



|  |  |
| --- | --- |
| 1 | > db.customers.find({uf: "RS"}) |

Note que a função find pode receber um documento por parâmetro representando o filtro a ser aplicado sobre a coleção para retornar documentos. Nesse caso, disse ao find que retornasse todos os documentos que possuam o campo uf definido como “RS”. O resultado no seu terminal deve ser somente o customer de nome “Teste” (não vou falar do \_id dele aqui pois o valor muda completamente de um servidor MongoDB para outro).

**Atenção:** MongoDB é case-sensitive ao contrário dos bancos relacionais, então cuidado!

Experimente digitar outros valores ao invés de “RS” e verá que eles não retornam nada, afinal, não basta ter o campo uf, ele deve ser exatamente igual a “RS”.

Além de campos com valores específicos, esse parâmetro do find permite usar uma infinidade de operadores como por exemplo, trazer todos documentos que possuam a letra ‘a’ no nome:



|  |  |
| --- | --- |
| 1 | > db.customers.find({nome: { $regex: /a/ }}) |

Se você já mexeu com expressões regulares (regex) em JS antes, sabe exatamente como usar e o poder desse recurso junto a um banco de dados, sendo um equivalente muito mais poderoso ao LIKE dos bancos relacionais.

Mas e se eu quiser trazer todos os customers maiores de idade?



|  |  |
| --- | --- |
| 1 | > db.customers.find({idade: {$gte: 18}}) |

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 18. E podemos facilmente combinar filtros usando vírgulas dentro do documento passado por parâmetro, assim como fazemos quando queremos inserir campos em um documento:



|  |  |
| --- | --- |
| 1 | > db.customers.find({nome: "Luiz", idade: {$gte: 18}}) |

O que a expressão acima irá retornar?

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

E a expressão abaixo:



|  |  |
| --- | --- |
| 1 | > db.customers.find({nome: { $regex: /a/ }, idade: {$gte: 18}}) |

Customers cujo nome contenham a letra ‘a’ e que sejam maiores de idade, é claro!

Outros 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. Ex: {idade: {$in: [10,12] }}
* **$all:** MongoDB permite campos com arrays. Ex: { tags: [“NodeJS”, “MongoDB”] }. Com esse operador, você compara se seu campo multivalorado possui todos os valores de um array específico. Ex: {tags: {$all: [“NodeJS”, “Android”]}}

Entre outros!

Você também pode usar findOne ao invés de find para retornar apenas o primeiro documento, ou ainda as funções limit e skip para limitar o número de documentos retornados e para ignorar alguns documentos, especificamente, da seguinte maneira:



|  |  |
| --- | --- |
| 1 | > db.customers.find().skip(1).limit(10) |

No exemplo acima retornaremos 10 customers ignorando o primeiro existente na coleção.

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 descrescente (-1), como abaixo em que retorno todos os customers ordenados pela idade:



|  |  |
| --- | --- |
| 1 | > db.customers.find().sort({idade: 1}) |

**Nota:**assim como nos bancos relacionais, os métodos de consulta retornam em ordem de chave primária por padrão, o que neste caso é o \_id.

Gravei recentemente um vídeo pra Umbler onde falo bastante de consultas em MongoDB, que você pode conferir abaixo:

Já os slides do vídeo podem ser conferidos abaixo:

[**Introdução ao MongoDB II**](https://www.slideshare.net/secret/Ckmgi13Md23iaB)de [**Luiz Fernando Duarte Jr**](https://www.slideshare.net/luizfduartejr)

Ok, vimos como usar o find de maneiras bem interessantes e úteis, mas e os demais comandos de manipulação do banco?

[[](https://www.luiztools.com.br/livro-mongodb)](https://www.luiztools.com.br/livro-mongodb)

### #3 – Update

Além do insert que vimos antes, também podemos atualizar documentos já existentes, por exemplo usando o comando update e derivados. O jeito mais simples (e mais burro) de atualizar um documento é chamando a função update na coleção com 2 parâmetros:

* documento de filtro para saber qual(is) documento(s) será(ão) atualizado(s);
* novo documento que substituirá o antigo;

Como em:



|  |  |
| --- | --- |
| 1 | > db.customers.update({nome: "Luiz"}, {nome: "Luiz", idade: 29, uf: "RS"}) |

Como resultado você deve ter um nModified maior do que 1, mostrando quantos documentos foram atualizados.

Por que essa é a maneira mais burra de fazer um update? Porque além de perigosa ela exige que você passe o documento completo a ser atualizado no segundo parâmetro, pois ele substituirá o original!

**Primeira regra do update inteligente:** se você quer atualizar um documento apenas, comece usando updateOne ao invés de update. O updateOne vai te obrigar a usar operadores ao invés de um documento inteiro para a atualização, o que é muito mais seguro.

**Segunda regra do update inteligente:** sempre que possível, use a chave primária (\_id) como filtro da atualização, pois ela é sempre única dentro da coleção.

**Terceira regra do update inteligente:** sempre use operadores ao invés de documentos inteiros no segundo parâmetro, independente do número de documentos que serão atualizados.

Mas que operadores são esses?

Assim como o find possui operadores de filtro, o update possui operadores de atualização. Se eu quero, por exemplo, mudar apenas o nome de um customer, eu não preciso enviar todo o documento do respectivo customer com o nome alterado, mas sim apenas a expressão de alteração do nome, como abaixo (já usando o \_id como filtro, que é mais seguro):



|  |  |
| --- | --- |
| 1 | > db.customers.updateOne({\_id: ObjectId("59ab46e433959e2724be2cbd")}, {$set: {idade: 28}}) |

**Nota:** para saber o \_id correto do seu update, faça um find primeiro e não tente copiar o meu pois não vai repetir.

Esta função vai alterar (operador $set) a idade para o valor 28 do documento cujo \_id seja “59ab46e433959e2724be2cbd” (note que usei uma função ObjectId para converter, pois esse valor não é uma string).

**Nota:**você pode usar null se quiser “limpar” um campo.

O operador $set recebe um documento contendo todos os campos que devem ser alterados e seus respectivos novos valores. Qualquer campo do documento original que não seja indicado no set continuará com os valores originais.

**Atenção:** se o campo a ser alterado não existir no documento, ele será criado.

Não importa o valor que ela tenha antes, o operador $set vai sobrescrevê-lo. Agora, se o valor anterior importa, como quando queremos incrementar o valor de um campo, não se usa o operador $set, mas sim outros operadores. A lista dos mais úteis operadores de update estão abaixo:

* **$unset:** remove o respectivo campo do documento;
* **$inc:** incrementa o valor original do campo com o valor especificado;
* **$mul:** multiplica o valor original do campo com o valor especificado;
* **$rename:** muda o nome do campo para o nome especificado;

Além disso, existe um terceiro parâmetro oculto no update que são as opções de update. Dentre elas, existe uma muito interessante do MongoDB: upsert, como abaixo:



|  |  |
| --- | --- |
| 1 | > db.customers.updateOne({nome: "LuizTools"}, {$set: {uf: "RS"}}, {upsert: true}) |

Um upsert é um update como qualquer outro, ou seja, vai atualizar o documento que atender ao filtro passado como primeiro parâmetro, porém, se não existir nenhum documento com o respectivo filtro, ele será inserido, como se fosse um insert.

upsert = update + insert

Eu já falei como amo esse banco de dados? 

### #4 – Delete

Pra encerrar o nosso conjunto de comandos mais elementares do MongoDB falta o delete, ops, deleteOne e deleteMany na verdade. Tem também o remove, que possui mais opções, mas é pouco usado.

Existe uma função deleteOne e uma deleteMany, o que a essa altura do campeonato você já deve saber a diferença. Além disso, assim como o find e o update, o primeiro parâmetro do remove é o filtro que vai definir quais documentos serão deletados e todos os operadores normais do find são aplicáveis.

Sendo assim, de maneira bem direta:



|  |  |
| --- | --- |
| 1 | > db.customers.deleteOne({nome: "Luiz"}) |

Vai excluir todos os clientes cujo nome seja igual a “Luiz”.

Simples, não?!

Obviamente existem coisas muito mais avançadas do que esse rápido tópico de MongoDB. Lhe encorajo a dar uma olhada no [site oficial do banco de dados](http://mongodb.org/) onde há a seção de documentação, vários tutoriais e até mesmo a possibilidade de tirar certificações online para garantir que você realmente entendeu a tecnologia.

Na [terceira parte deste tutorial](https://www.luiztools.com.br/post/tutorial-mongodb-para-iniciantes-em-nosql-3/), você verá como fazer backup e restaurar bases de dados MongoDB!

E o vídeo? Já viu o vídeo com Introdução ao MongoDB que gravei pra [Umbler](https://www.luiztools.com.br/umbler)?

Referencias

<https://www.luiztools.com.br/post/tutorial-mongodb-para-iniciantes-em-nosql/#4>

<https://www.luiztools.com.br/post/tutorial-mongodb-para-iniciantes-em-nosql-2/>