Digital College

FORMAÇÃO EM

DATA ANALYTICS

MÓDULO 1:

BANCO DE DADOS

UNIDADE 4:

> BANCO DE DADOS NÃO RELACIONAIS >

Sumário

1. Introdução a Banco de Dados No SQL

NoSQL significa 'não relacional'. Bancos NoSQL são comumente usados em áreas de conhecimento como Data Science. A maior diferença entre bancos NoSQL e relacionais é que bancos relacionais trabalham com tabelas, enquanto em Bancos NoSQL todos os dados constam no mesmo registro.

O termo NoSQL foi primeiramente utilizado em 1998 como o nome de um banco de dados não relacional de código aberto.

Seu autor, Carlo Strozzi, alega que o movimento NoSQL "é completamente distinto do modelo relacional e, portanto, deveria ser mais apropriadamente chamado "NoREL" ou algo que produzisse o mesmo efeito".

Com a crescente popularização da internet, diversos novos dados foram surgindo e tratar esses dados foi se tornando gradualmente mais complexo e sua manutenção foi ficando cada vez mais cara. Em 2006, o artigo "BigTable: A Distributed Storage System for Structured Data", publicado pelo Google em 2006, trouxe novamente à tona o conceito NoSQL. No início de 2009, o termo NoSQL foi reintroduzido por um funcionário do Rackspace, Eric Evans, quando Johan Oskarson da Last.fm quis organizar um evento para discutir bancos de dados open source distribuídos.

O nome era uma tentativa de descrever o surgimento de um número crescente de bancos de dados não relacionais e fazia uma referência ao esquema de atribuição de nomes dos bancos de dados relacionais mais populares do mercado, como MySQL, MS SQL, PostgreSQL etc.

A partir de então, os bancos de dados não relacionais passaram a ser conhecidos como NoSQL, e com a crescente popularização das redes sociais, a geração de conteúdo por dispositivos móveis, bem como o número cada vez maior de pessoas e de dispositivos conectados, fizeram com que o trabalho de armazenamento de dados com o objetivo de utilizá-los em ferramentas analíticas começasse a esbarrar nas questões de escalabilidade e custos de manutenção desses dados.

Bancos de dados relacionais escalam, mas quan-to maior o tamanho, mais custosa se torna essa escalabilidade, seja pelo custo de novas máquinas, seja pelo aumento de especialistas nos bancos de dados utilizados. Já os não relacionais, permitem uma escalabilidade mais barata e menos trabalhosa, pois não exigem máquinas extremamente poderosas e sua facilidade de manutenção permite que menos profissionais sejam necessários.

Assim, os bancos de dados NoSQL vão ficando mais populares entre as grandes, pois reúnem as características de poder trabalhar com dados semi-estruturados ou crus vindos de diversas origens (arquivos de log, websites, arquivos multimídia etc).

> > Podemos listar algumas dessas características abaixo:

- ***Produtividade no desenvolvimento de aplicativos:*** muito trabalho de desenvolvimento de aplicativos é gasto no mapeamento de dados entre as estruturas de dados na memória e um banco de dados relacional. Um banco de dados NoSQL pode fornecer um modelo de dados que se adapte melhor às necessidades do aplicativo, simplificando, dessa forma, essa interação e resultando em menos código a ser escrito, depurado e melhorado.

- ***Utilização do processamento paralelo para processamento das informações:*** para se atingir uma performance razoável no processamento de grandes volumes de dados, é mais eficiente dividir a tarefa em várias outras menores e que podem, assim, serem executadas ao mesmo tempo, distribuindo essas tarefas pelos vários processadores disponíveis. Para isso, os sistemas precisam atingir muita maturidade no processamento paralelo. O uso de muitos processadores baratos, não só oferece melhor performance, mas se torna também uma solução economicamente interessante, pois dessa forma é possível escalar o sistema horizontalmente, apenas adicionando hardware e não limita a empresa a poucos fornecedores de hardware mais poderoso.

- ***Distribuição em escala global:*** para atender seus usuários de forma eficiente, algumas empresas utilizam vários data centers, localizados em diversas partes do país ou do mundo. Com isso, uma série de questões sobre disponibilidade e performance são levantadas ao construir os sistemas.

A distribuição deles combinada com o hardware barato impõe ao sistema a necessidade de ser robusto o suficiente para tolerar falhas constantes e imprevisíveis, seja de hardware, seja da infraestrutura do lugar onde o data center se encontra.

> > 1.1. Tipos de Banco de Dados NoSQL

Podemos classificar os bancos de dados NoSQL conforme abaixo:

- ***Banco de dados chave/valor (Key/Value):*** sistemas distribuídos nesta categoria, também conhecidos como tabelas de hash distribuídas, armazenam objetos indexados por chaves, e possibilitam a busca por esses objetos a partir de suas chaves. Alguns bancos que utilizam esse padrão são: DynamoDb, Couchbase, Riak, Azure Table Storage, Redis, Tokyo Cabinet, Berkeley DB etc.

- ***Bancos de dados orientados a documentos:*** os documentos dos bancos dessa categoria são coleções de atributos e valores, onde um atributo pode ser multivalorado. Em geral, os bancos de dados orientados a documentos não possuem esquema, ou seja, os documentos armazenados não precisam possuir estrutura em comum. Essa característica faz deles boas opções para o armazenamento de dados semi-estruturados. Alguns bancos que utilizam esse padrão são: MongoDb, CouchDB, RavenDb etc.

- ***Bancos de dados de famílias de colunas:*** bancos relacionais normalmente guardam os registros das tabelas contiguamente no disco. Por exemplo, caso se queira guardar id, nome e endereço de usuários em um sistema de cadastro, os registros seriam: Id1, Nome1, Endereço1; Id2, Nome2, Endereço2.

Essa estrutura torna a escrita muito rápida, pois todos os dados de um registro são colocados no disco com uma única escrita no banco. Essa estrutura também é eficiente caso se queira ler registros inteiros. Já para situações em que se quer ler algumas poucas colunas de muitos registros, essa estrutura é pouco eficiente, pois muitos blocos do disco terão de ser lidos. Para esses casos em que se quer otimizar a leitura de dados estruturados, bancos de dados de famílias de colunas são mais interessantes, pois eles guardam os dados contiguamente por coluna.

O exemplo anterior em um banco de dados dessa categoria ficaria: Id1, Id2; Nome1, Nome2; Endereço1, Endereço2. Por esse exemplo é possível perceber a desvantagem de um banco de dados de famílias de colunas: a escrita de um novo registro é bem mais custosa do que em um banco de dados tradicional. Assim, num primeiro momento, os bancos tradicionais são mais adequados ao processamento de transações on-line (OLTP), enquanto os bancos de dados de famílias de colunas são mais interessantes para processamento analítico on-line (OLAP). O Bigtable é uma implementação da Google dessa categoria de bancos de dados. Outros bancos de dados que são orientados a coluna: Hadoop, Cassanda, Hypertable, Amazon SimpleDB etc.

- ***Bancos de dados de grafos:*** diferentemente de outros tipos de bancos de dados NoSQL, esse está diretamente relacionado a um modelo de dados estabelecido, o modelo de grafos. A ideia desse modelo é representar os dados e/ou o esquema dos dados como grafos dirigidos, ou como estruturas que generalizem a noção de grafos.

O modelo de grafos é mais interessante que outros quando "informações sobre a interconectividade ou a topologia dos dados são mais importantes, ou tão importante quantos os dados propriamente ditos. O modelo orientado a grafos possui três componentes básicos: os nós (são os vértices do grafo), os relacionamentos (são as arestas) e as propriedades (ou atributos) dos nós e relacionamentos.

Neste caso, o banco de dados pode ser visto como um multigrafo rotulado e direcionado, em que cada par de nós pode ser conectado por mais de uma aresta. Um exemplo pode ser: "Quais cidades foram visitadas anteriormente (seja residindo ou viajando) por pessoas que viajaram para o Rio de Janeiro?" No modelo relacional, esta consulta poderia ser muito complexa devido a necessidade de múltiplas junções, o que poderia acarretar uma diminuição no desempenho da aplicação. Porém, por meio dos relacionamentos inerentes aos grafos, essas consultas tornam-se mais simples e diretas. Alguns bancos que utilizam esse padrão são: Neo4J, Infinite Graph, InforGrid, HyperGraphDB etc.

> > 1.2. Qual o melhor banco NoSQL?

Os Bancos de Dados NoSQL referem-se a uma série de tecnologias diferentes, que não são essencialmente relacionais. Mas o que os melhores bancos de dados NoSQL têm em comum?

> > 1.2.1. MongoDB

Quando falamos em MongoDB estamos falando de um líder de mercado dos bancos de dados NoSQL. O MongoDB também possui *features* bem legais para produção. São eles: replicação, indexação e balanceamento de carga.

Para armazenar dados, o MongoDB utiliza alguns documentos muito similares ao formato JSON*. O melhor de tudo – e talvez a razão de ser líder de mercado – é que o MongoDB é open source, o que contribui muito para a evolução da sua tecnologia.

> > 1.2.2. Amazon DynamoDB

Mais um produto excelente da AWS (Amazon Web Services). O banco de dados DynamoDB é totalmente cloud e viabiliza um desempenho confiável e em escala. Um ponto bem importante: a Amazon confirma que a latência é consistente e fica abaixo de 10 milissegundos. Além disso, tem recursos valiosos de segurança, baseados em cache de memória, backup e restauração de dados.

O DynamoDB também funciona por meio de vários mestres. Este banco de dados já é amplamente utilizado, assim como o MongoDB e pode ser utilizado para criação de datastore, jogos, ad tech e aplicativos web sem servidor.

> > 1.2.3. Cassandra

Muitas pessoas não sabem, mas o Cassandra foi desenvolvido no Facebook. Hoje em dia, o Cassandra – assim como o HBase – é mantido pela Apache Foundation. Isso até faz sentido, considerando a quantidade de dados que a rede social gera a cada milissegundo.

Mas, afinal, por que Cassandra é tão popular para trabalhar com Big Data? O fato é que Cassandra é muito otimizado para clusters, especialmente por funcionar sem mestres.

O fato de ter mecanismos distribuídos também otimiza bastante a operação com os clusters. Um outro ponto forte do Cassandra é o conceito de orientação por coluna, o que torna a latência bem menor em algumas pesquisas.

> > 1.2.4. Redis

O Redis é um modelo de armazenamento de dados, que é open source e foi lançado em 2009. Os dados são armazenados na forma de chave-valor e na memória do Redis, o que o torna rápido e flexível. Trata-se do Banco NoSQL mais famoso do tipo chave-valor. Assim como os dois primeiros, o Redis possui baixíssima latência. O Redis é também fácil de usar e muito rápido.

> > 1.2.5. HBase

O HBase é um banco de dados open source, orientado a colunas e distribuído. Atualmente, Spotify e Facebook são algumas das grandes corporações que utilizam esse modelo de armazenamento. O HBase foi formatado a partir do BigTable do Google e também é escrito em Java. É justamente por isso que tem fácil integração com o MapReduce.

Para quem não sabe o MapReduce é uma ferramenta do framework Apache Hadoop, uma das principais plataformas para tratamento de big data.

Por fazer parte do Projeto Apache, diretamente ligado à ciência de dados, o HBase é outro modelo de armazenamento bem famoso. Um dos seus pontos fortes é a pesquisa de dados que oferece uma resposta rápida. Transforma terabytes em milissegundos.

> > 1.3. NoSQL vs SQL

SQL	NoSQL
Armazenamento de Dados Estruturados por Tabela	Armazenamento de Dados estruturados e não-estruturados por colunas, grafos, chave-valor e documentos.
Esquema estático	Esquema dinâmico
Maturidade de suporte maior (geralmente pago)	Suporte por comunidade independente (open source)
Escalabilidade vertical	Escalabilidade horizontal
Pago	Gratuito
O desempenho não é alto em todas as consultas. Não suporta pesquisas e cruzamentos muito complexos.	Alto desempenho em consultas
Necessidade de predefinição de um esquema de tabela antes da adição de qualquer dado	Altamente flexível (fácil adição de colunas e campos de dados não estruturados)

Bancos de Dados NoSQL são extremamente úteis quando o assunto é grande volume de dados. Se estivermos falando de uma corporação pequena, que não aprofunda tanto assim a análise e o tratamento de dados, o banco relacional funciona muito bem.

E não, o banco de dados NoSQL não veio para substituir o relacional. Veio para ser uma alternativa, em meio a um mundo onde pouquíssimos dados ainda são usados para inteligência.

2. Introdução e instalação do MongoDB

O MongoDB é um dos principais bancos de dados NoSQL da atualidade. Estes nomes podem assustar muita gente logo de cara, mas o conceito é simples: ele, o MongoDB, é um software que instalamos, configuramos e utilizamos no computador, seja ele um daqueles que temos em casa (tipo desktop ou notebook) ou daqueles que rodam em grandes centros com milhares de computadores juntos (servidores).

Alguns leitores podem estranhar um pouco o nome MongoDB, mas é assim mesmo no começo. Leva um tempo até ficarmos acostumados com este estranho nome, mas isso passa logo quando trabalhamos com este software. O MongoDB é o banco de dados NoSQL mais popular e ele possui diversas características que o diferenciam de outros produtos. Uma delas é o fato de que ele é um software livre, ou seja, pode ser utilizado e modificado sem a necessidade de algum tipo de pagamento.

Neste capítulo vamos discutir um pouco sobre o que é, para que serve e por que devemos considerar o MongoDB como o nosso repositório para armazenamento de dados. Também são abordados assuntos um pouco mais técnicos como o formato de armazenamento JSON e seus elementos, e outros aspectos importantes do MongoDB.

> > 2.1. Instalando o MongoDB no Windows

CLIQUE AQUI PARA BAIXAR

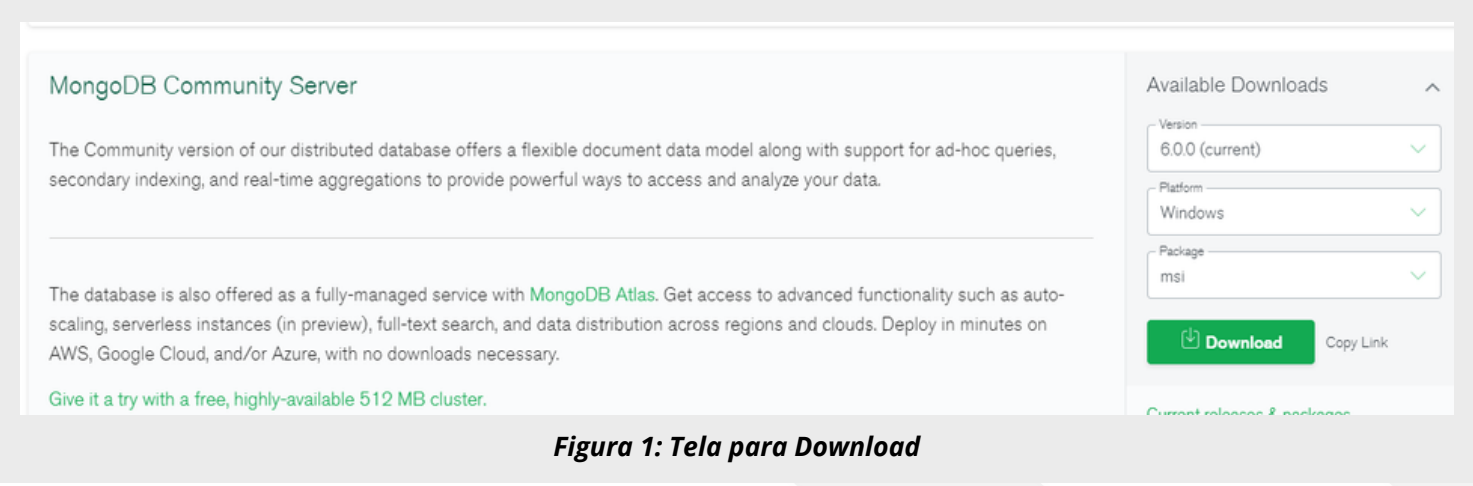

Figura 1: Tela para Download

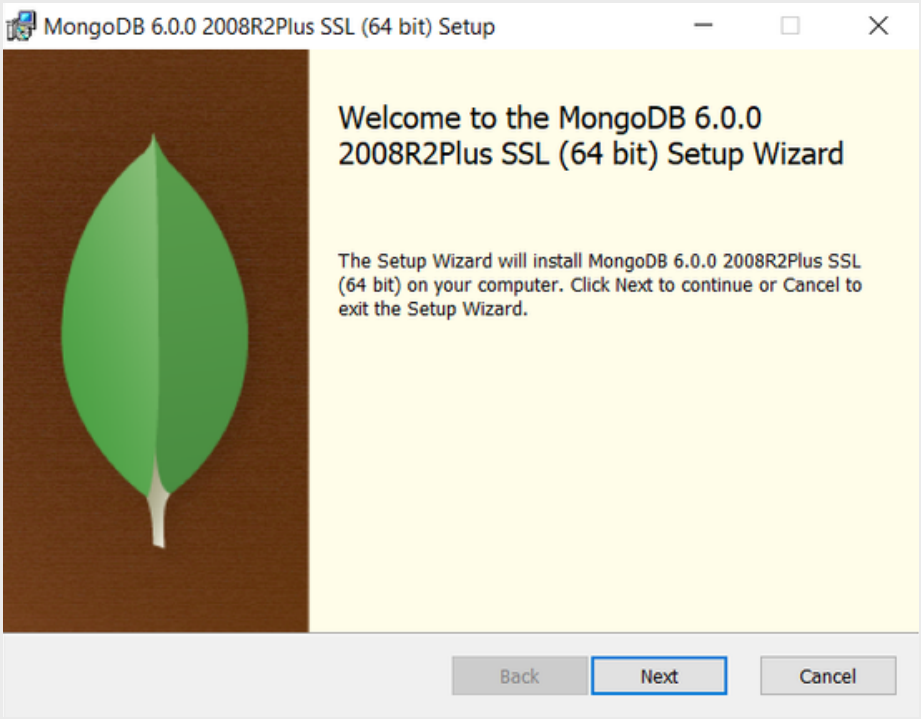

Figura 2: Iniciando a instalação

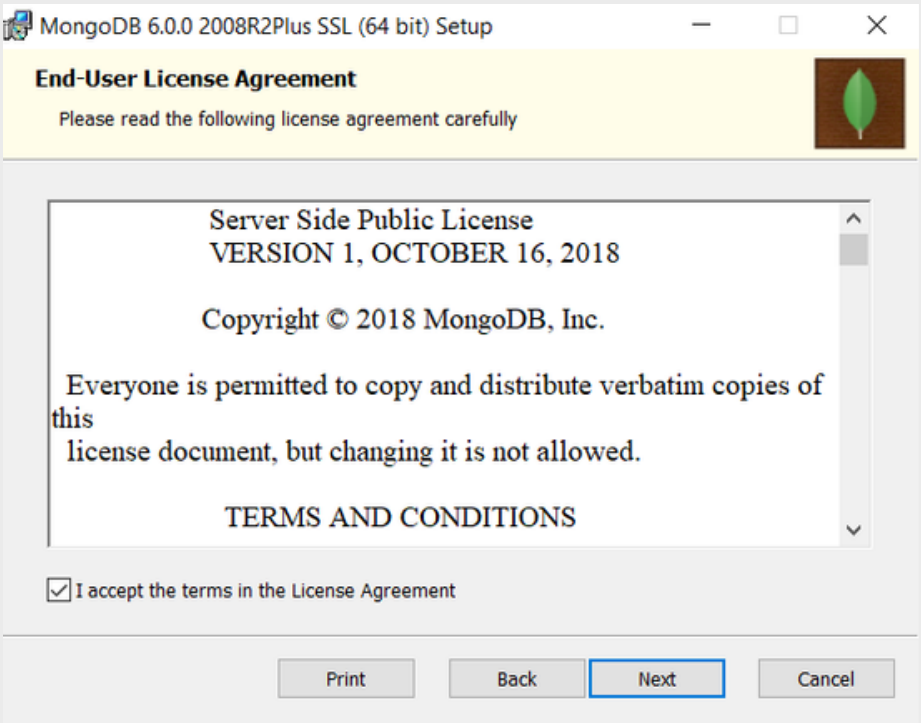

Figura 3: Aceitar termos de uso

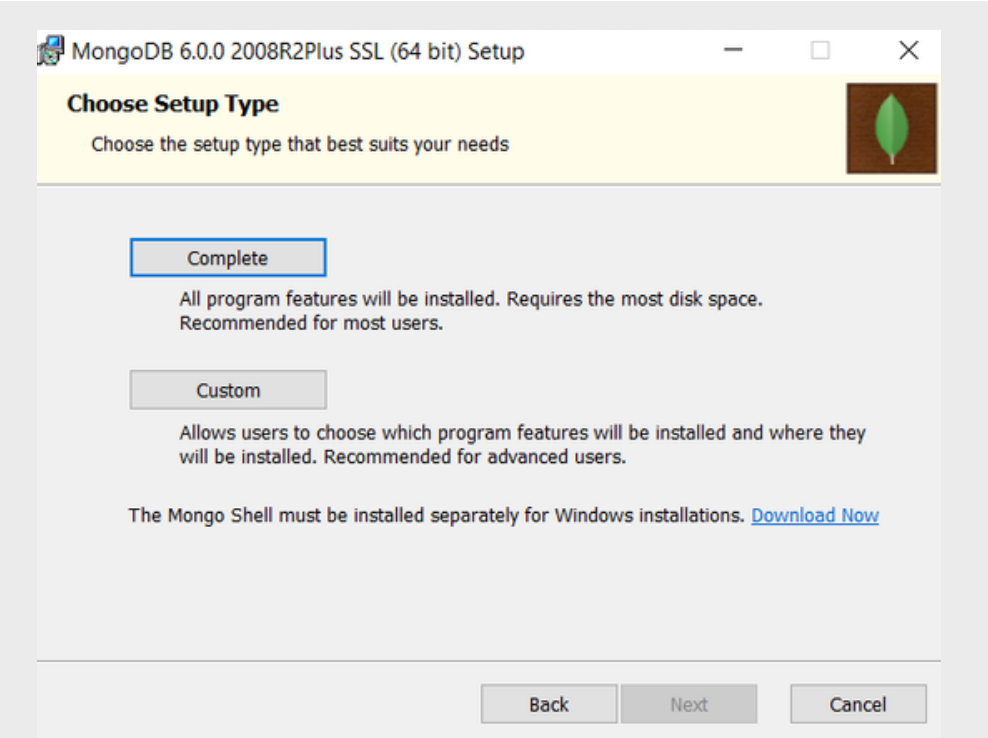

Figura 4: Clicar em "Complete"

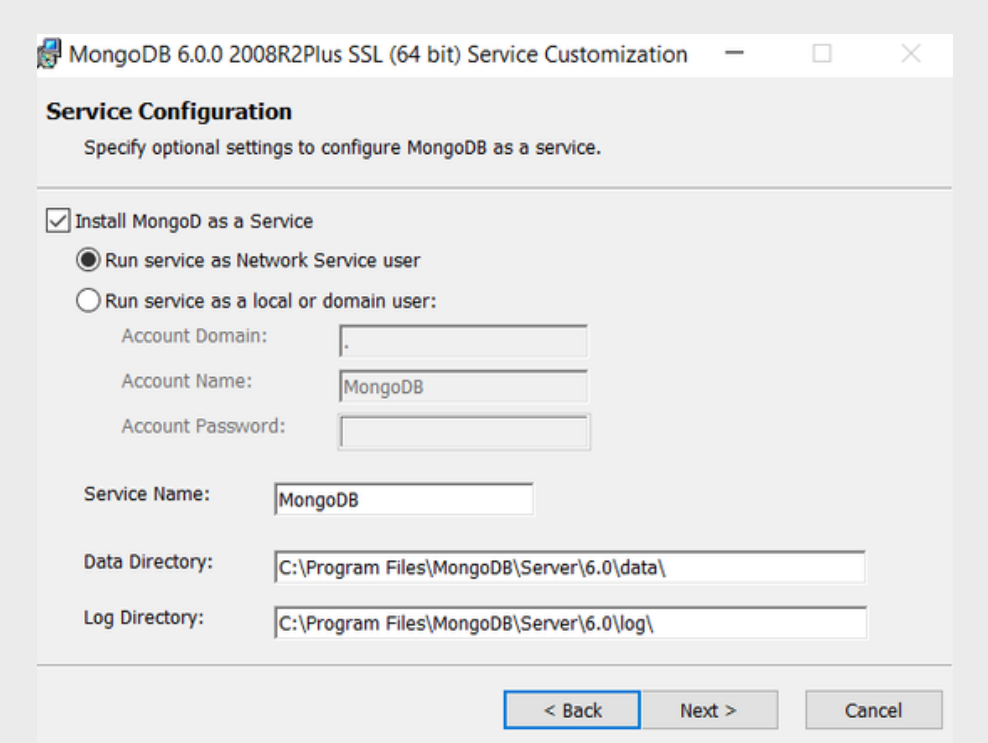

Figura 5: Instalar MongoDB com Serviço

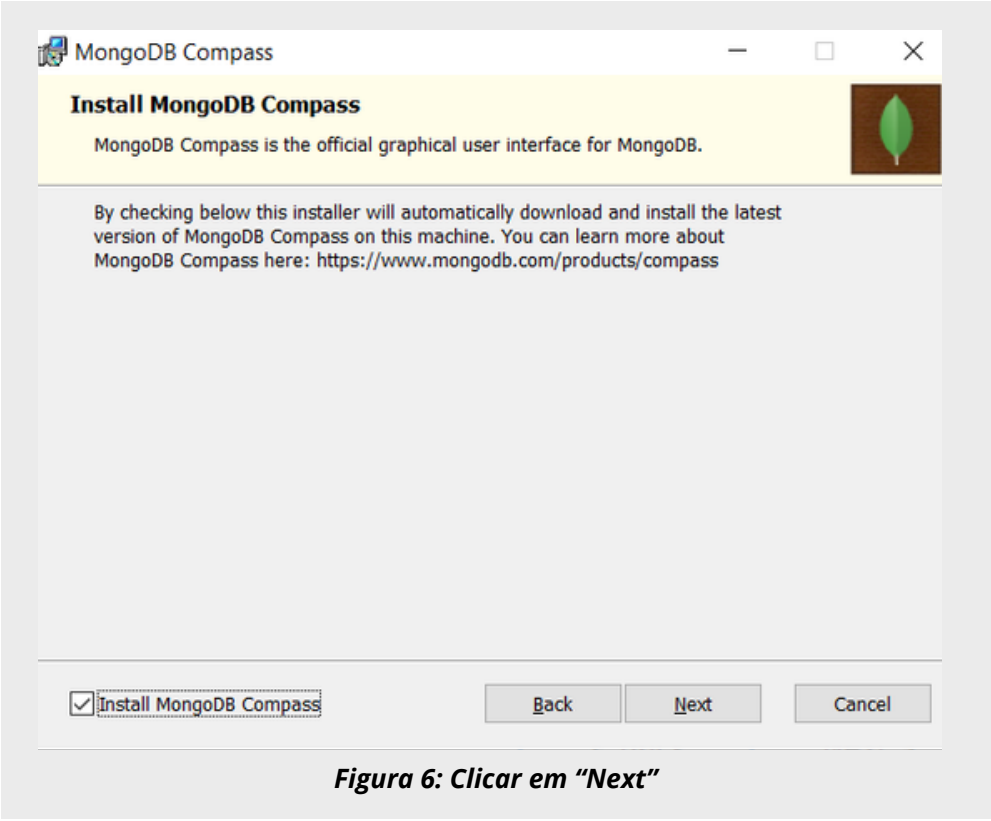

Figura 6: Clicar em "Next"

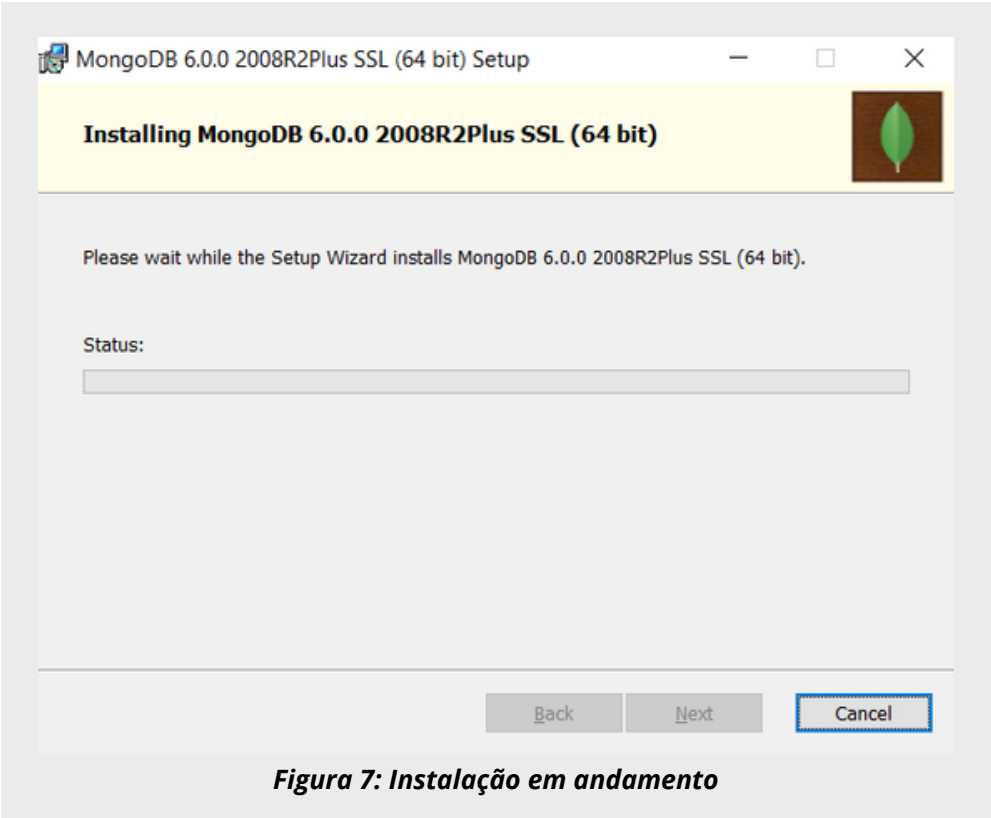

Figura 7: Instalação em andamento

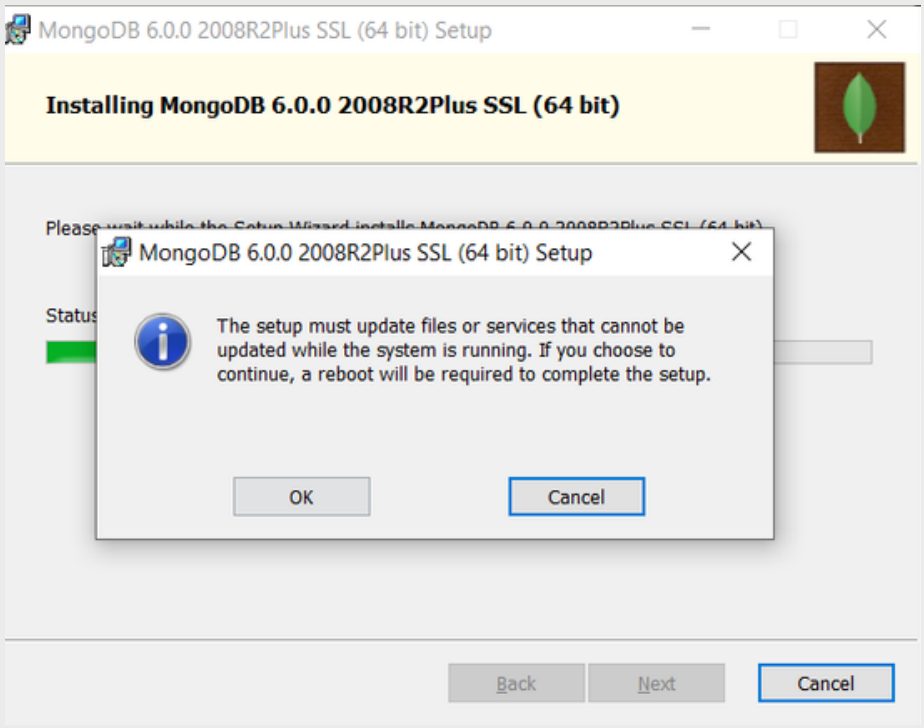

Figura 8: Realizar o Reboot da máquina. Clicar em "OK"

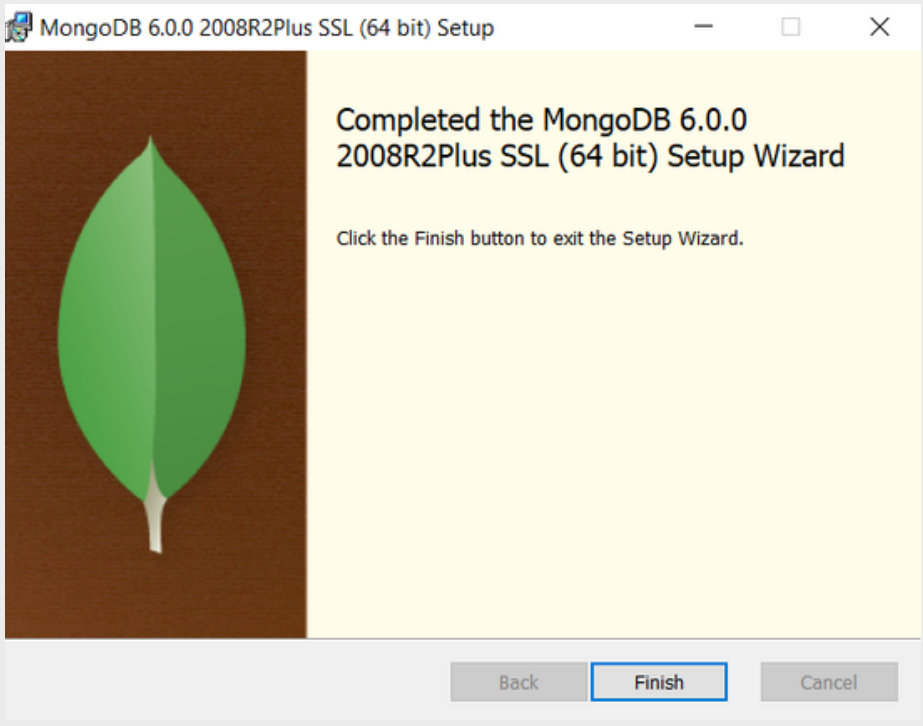

Figura 9: Instalação concluída

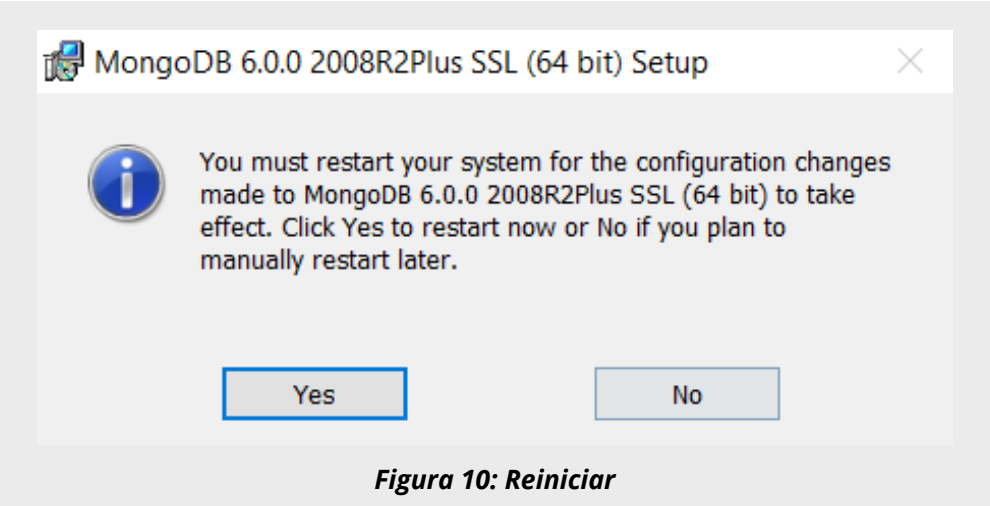

Figura 10: Reiniciar

3. Trabalhando com Usuários no MongoDB

SGDs modernos são capazes de armazenar e processar enormes quantidades de dados. Por causa disso, é relativamente incomum ter um único responsável por lidar com todas as atividades relacionadas ao gerenciamento de um banco de dados. Muitas vezes, diferentes usuários exigem diferentes níveis de acesso a determinadas partes de um banco de dados: alguns usuários podem precisar apenas ler dados específicos, enquanto outros devem ser capazes de inserir novos documentos ou modificar os existentes. Da mesma forma, um aplicativo pode exigir permissões exclusivas que só permitem que ele acesse as partes de um banco de dados que precisa para funcionar.

> > 3.1. Como MongoDB controla o acesso com Role-Based Access Control

O controle de acesso — também conhecido como *autorização* — é uma técnica de segurança que envolve determinar quem pode ter acesso a quais recursos. Para entender melhor o controle de acesso no MongoDB, pode ser útil primeiro distingui-lo de um conceito diferente, mas intimamente relacionado: autenticação. *A autenticação* é o processo de confirmar se um usuário ou cliente é realmente quem ele afirma ser. *A autorização*, por outro lado, envolve a definição de regras para um determinado usuário ou grupo de usuários definir quais ações eles podem executar e quais recursos eles podem acessar.

> > 3.1.1. Autenticando no MongoDB

Em muitos sistemas de gerenciamento de banco de dados, os usuários são identificados apenas com um nome de usuário e um par de senhas. Ao se conectar a um banco de dados com credenciais válidas, o usuário é autenticado e, a partir daí, é concedido o nível de acesso associado a esse usuário.

Em tal abordagem, o diretório do usuário é plano, o que significa que para todo servidor de banco de dados, cada nome de usuário deve ser único.

Em contraste, o MongoDB emprega uma estrutura de diretório de usuário mais complexa. No MongoDB, os usuários não são identificados apenas por seus nomes, mas também pelo *banco de dados* em que foram criados. Para cada usuário, o banco de dados em que foi criado é conhecido como "banco de dados de *autenticação*" deste. Isso significa que no MongoDB é possível ter vários usuários com o mesmo nome (**sammy**, por exemplo) desde que sejam criados em diferentes bancos de dados de autenticação. Para autenticar, você deve fornecer não apenas um nome de usuário e senha, mas também o nome do banco de dados de autenticação associado a este.

Pode-se supor que os usuários criados em um determinado banco de dados de autenticação tenham privilégios de acesso disponíveis apenas para esse banco de dados específico, mas este não é o caso. Cada usuário, não importa em qual banco de dados de autenticação tenha sido criado, pode ter privilégios atribuídos em diferentes bancos de dados.

> > 3.1.2. Autorizações no MongoDB (Role-Based Access Control)

No MongoDB, você controla quem tem acesso a quais recursos em um banco de dados, bem como até que ponto vai esse acesso, através de um mecanismo chamado Role-Based Access Control, muitas vezes encurtado como *RBAC*.

No Role-Based Access Control, os usuários não recebem permissões para executar ações em recursos diretamente, como inserir um novo documento no banco de dados ou consultar uma determinada coleção. Isso dificultaria a gestão e a manutenção das políticas de segurança consistentes com muitos usuários no sistema. [SUGESTÃO: dada a grande quantidade de usuários presentes no sistema.] Em vez disso, as regras que permitem ações sobre determinados recursos são atribuídas a *funções*.

Pode ser útil pensar em um papel como um dos trabalhos ou responsabilidades de um determinado usuário. Por exemplo, pode fazer sentido um gerente ter lido esse papel e nele ter escrito o acesso a cada documento na instância mongoDB de uma empresa, enquanto um analista de vendas pode ter um acesso "somente leitura" aos registros de vendas.

Os papéis são definidos com um conjunto de um ou mais *privilégios*. Cada privilégio consiste em uma ação (como a criação de novos documentos, a recuperação de dados de um documento ou a criação e exclusão de usuários) e o recurso no qual essa ação pode ser realizada (como um banco de dados chamado "relatórios" ou uma coleta chamada "ordens"). Assim como na vida real, onde uma empresa pode ter muitos analistas de vendas e funcionários tendo mais de uma responsabilidade, no MongoDB muitos usuários podem ser atribuídos a mesma função, e um único usuário pode ter muitas funções concedidas.

As funções são identificadas com a combinação do nome da função e do banco de dados, já que cada função — exceto as criadas no banco de dados de administradores — só pode incluir privilégios aplicados em seu próprio banco de dados.

Ao conceder funções de usuário definidas em um banco de dados diferente de seu banco de dados de autenticação, um usuário pode receber permissões para atuar em mais de um banco de dados. As funções podem ser concedidas quando você cria um usuário (ou a qualquer momento depois disso). A revogação da adesão à função também pode ser feita à vontade, tornando simples o procedimento de desvincular a gestão de usuários da gestão de direitos de acesso.

O MongoDB fornece um conjunto de funções incorporadas, descrevendo privilégios comumente usados em sistemas de banco de dados, como ler para conceder acesso somente leitura, lerWrite para conceder permissões de leitura e gravação, ou dbOwner para conceder privilégios administrativos completos sobre um determinado banco de dados. Para cenários mais específicos, as funções definidas pelo usuário também podem ser criadas com conjuntos personalizados de privilégios.

Role-Based Access Control permite atribuir aos usuários apenas o nível mínimo e preciso de permissões de acesso que eles precisam para trabalhar em suas respectivas tarefas. Esta é uma importante prática de segurança conhecida como o princípio do menor privilégio.

Para explicar como o Role-Based Access Control — RBAC funciona, segue um cenário de exemplo com uma empresa de vendas imaginária, chamada Xpto Vendas, que usa dois bancos de dados.

O primeiro banco de dados (chamado de vendas) armazenará dados sobre pedidos de clientes na loja da empresa com duas coleções separadas: "clientes" para dados pessoais de seus clientes, e "pedidos" para detalhes do pedido.

O segundo banco de dados (este chamado relatórios) armazenará relatórios agregados sobre vendas mensais. Este banco de dados conterá uma única coleção chamada relatórios.

A empresa possui apenas dois funcionários, ambos com um nível de acesso ao banco de dados que segue a abordagem de menor privilégio:

1. A representante de vendas precisa de acesso total a ambas as coleções no banco de dados de vendas, mas não precisa trabalhar com o banco de dados de relatórios.
2. Joe, um analista de vendas, precisa escrever acesso ao banco de dados de relatórios para construir relatórios, bem como acesso somente leitura ao banco de dados de vendas para recuperar os dados.

Para criar esses bancos de dados de amostra, abra o shell MongoDB no servidor onde você instalou o MongoDB com um comando como abaixo:

```
1.mongo
```

Você pode verificar se tem acesso a toda a instância do MongoDB emitindo o comando show dbs:

```
1.show dbs
```

Isso retornará uma lista de todos os bancos de dados disponíveis atualmente:

```
Output
admin   0.000GB
config  0.000GB
local   0.000GB
```

Depois de confirmar que você pode acessar esses bancos de dados, mude para o banco de dados de vendas:

```
1.use vendas
2.
```

```
Output
switched to db sales
```

No MongoDB, não há nenhuma ação explícita para criar um banco de dados. Um banco de dados só é criado quando armazena pelo menos um documento. Com isso em mente, você precisará inserir alguns documentos de amostra para preparar os bancos de dados.

Você está agora no banco de dados de vendas, mas ele não vai realmente existir até que você insira algo nele.

Crie uma coleção nomeada "clientes" dentro das vendas e insira simultaneamente um novo documento nele com a seguinte operação:

```
1.db.clientes.insert({name: XPTO})
```

Esse documento de exemplo contém apenas um campo de nome com o valor XPTO. Observe que os dados em si não são relevantes para mostrar como os direitos de acesso funcionam na prática, por isso esta etapa descreve como criar documentos de banco de dados que contêm apenas dados simulados de exemplo.

O MongoDB confirmará a inserção com:

```
Output
WriteResult({ "nInserted" : 1 })
```

Repita este processo, mas desta vez crie uma coleção chamada "pedidos". Para fazer isso, execute o seguinte comando. Desta vez, o único campo do documento é total e tem um valor de 100:

```
1.db.pedidos.insert({total: 100})
```

O MongoDB confirmará mais uma vez que o documento foi devidamente inserido.

Uma vez que você trabalhar com dois bancos de dados, você precisará preparar o banco de dados de relatórios também. Para isso, primeiro mude para o banco de dados de relatórios:

```
1.use relatorios
```

E insira outro documento, desta vez na coleção de relatórios:

```
1.db.relatorios.insert({pedidos: 1})
2.
```

Para confirmar que ambas as bases de dados foram devidamente preparadas, emita o comando show dbs mais uma vez.

```
1.show dbs
2.
```

Depois de executar este comando uma segunda vez, o resultado mostrará duas novas entradas para bancos de dados recém-criados. Esses bancos de dados só persistiram depois que você criou os primeiros documentos dentro de cada um deles:

```
Output
admin       0.000GB
config      0.000GB
local       0.000GB
relatorios  0.000GB
vendas      0.000GB
```

Os bancos de dados de amostras estão prontos. Agora, você pode criar um par de usuários que terão a menor quantidade de privilégios de acesso aos bancos de dados recém-criados necessários para este cenário de exemplo.

> > 3.1.3. Criando um usuário

Nesta etapa, você criará o primeiro de dois usuários do MongoDB. Este primeiro usuário será para o representante de vendas da empresa. Essa conta precisará de acesso total ao banco de dados de vendas, mas não terá acesso algum ao banco de dados de relatórios.

Para isso, usaremos o papel de readWrite incorporado para conceder acesso tanto à leitura quanto à escrita aos recursos no banco de dados de vendas. Como representante de vendas, também usaremos o banco de dados de vendas como banco de dados de autenticação para o usuário recém-criado.

Primeiro, mude para o banco de dados de vendas:

```
1.use vendas
2.
```

```
Output
switched to db sales
```

Como o representante trabalha no departamento de vendas, sua conta de usuário MongoDB será criada com vendas como o banco de dados de autenticação.

Execute o seguinte método para criar o **usuário** representante:

```
db.createUser(

  {

    user: "representante",

    pwd: passwordPrompt(),

    roles: [

        { role: "readWrite", db: "vendas" }

      ]

    }

  )

```

Este método createUser inclui os seguintes objetos:

1. user representa o nome de usuário, que é **representante** neste exemplo;
2. Pwd representa a senha. Ao usar a passwordPrompt() você garantirá que o shell do MongoDB solicitará a senha ao executar o comando a ser inserido;
3. role é a lista de papéis concedidos. Este exemplo atribui **a representante** à função readWrite, concedendo-lhes acesso à leitura e gravação ao banco de dados de vendas. Nenhuma outra função é atribuída ao **representante** agora, o que significa que este usuário não terá nenhum direito adicional de acesso imediatamente após a criação.

```
. . .

    pwd: "password",

. . .

```

Se este método for bem-sucedido, ele retornará uma mensagem de confirmação do MongoDB semelhante à abaixo:

```
Output
Successfully added user: {
        "user" : "sammy",
        "roles" : [
                {
                        "role" : "readWrite",
                        "db" : "sales"
                }
        ]
}
```

Agora você pode verificar se o novo usuário pode fazer login no banco de dados e se seus direitos de acesso especificados são devidamente aplicados.

Você manterá a shell mongoDB atual com seu usuário administrativo conectado para mais tarde, e então abrirá **uma sessão separada do servidor**.

A partir da nova sessão de servidor, abra o shell MongoDB. Desta vez, especifique **a representante** como o usuário e as vendas como o banco de dados de autenticação:

```
mongo -u sammy -p --authenticationDatabase sales
```

Digite a senha definida ao criar o **usuário representante**. Depois de acessar o prompt shell, execute o comando show dbs para listar os bancos de dados disponíveis:

```
1.show dbs
```

Em contraste com sua conta administrativa, apenas um banco de dados será listado para **o representante**, pois você só lhes concederá acesso ao banco de dados de vendas.

```
Output
sales 0.000GB
```

Agora verifique se **representante** pode recuperar objetos de ambas as coleções no banco de dados de vendas. Mude para o banco de dados de vendas:

```
use vendas
```

Em seguida, tente recuperar todos os clientes:

```
db.clientes.find()
```

Este comando find retornará o documento que você criou nesta coleção anteriormente.

```
Output
{ "_id" : ObjectId("60d888946ae8ac2c9120ec40"), "name" : "Sammy" }
```

Você também pode confirmar que a segunda coleção, pedidos, está disponível como pretendido:

```
db.orders.find()
```

```
Output
{ "_id" : ObjectId("60d890730d31cc50dedea6ff"), "total" : 100 }
```

Para garantir que os direitos de acesso para o banco de dados de vendas tenham sido devidamente configurados, você pode verificar se o representante também pode inserir novos documentos. Tente inserir um novo cliente:

```
db.customers.insert({name: 'Ellie'})
```

Como você concedeu ao **representante** o papel de ReadWrite, ele está autorizado a escrever novos documentos para este banco de dados. O MongoDB confirmará que a inserção foi concluída com sucesso:

```
Output
WriteResult({ "nInserted" : 1 })
```

Por fim, verifique se o **representante** pode acessar o banco de dados de relatórios. Ele não será capaz de ler ou escrever quaisquer dados neste banco de dados, pois você não lhe concedeu acesso através das funções atribuídas.

Mude para o banco de dados de relatórios:

```
use reports
```

Este comando de uso não resultará em nenhum erro por si só. Tente acessar o documento inserido na Etapa 1, executando o seguinte:

```
db.reports.find()
```

Agora, o MongoDB lançará uma mensagem de erro em vez de retornar quaisquer objetos:

```
Output
Error: error: {
  "ok" : 0,
 "errmsg" : "not authorized on reports to execute command { find:
\"reports\", filter: {}, lsid: { id: UUID(\"cca9e905-89f8-4903-ae12-
46f23b43b967\") }, $db: \"reports\" }",
  "code" : 13,
  "codeName" : "Unauthorized"
}
```

A mensagem de erro não autorizada diz que **o representante** não tem direitos de acesso suficientes para interagir com o banco de dados de relatórios.

Até agora, você criou o primeiro usuário de banco de dados com privilégios limitados e verificou que os direitos de acesso são devidamente aplicados. Em seguida, você criará um segundo usuário com privilégios diferentes.

Tendo criado o usuário representante de vendas, você ainda precisa de uma conta para Joe, o analista de vendas da empresa. Lembre-se do cenário de exemplo de que a função de trabalho de Joe requer um conjunto diferente de privilégios para as bases de dados.

O processo de criação desta nova conta de usuário é semelhante ao processo que você seguiu para criar o **usuário representante.** Retorne à sessão do servidor onde seu usuário administrativo está logado no shell MongoDB. A partir daí, mude para o banco de dados de relatórios:

```
1.use relatorios
2.
```

Como Joe trabalha no departamento de relatórios, sua conta de usuário MongoDB será criada com relatórios como o banco de dados de autenticação.

Crie um novo usuário **joe** com o seguinte comando:

```
db.createUser(

  {

    user: "joe",

    pwd: passwordPrompt(),

    roles: [

      { role: "readWrite", db: "relatorios" },

      { role: "read", db: "vendas" }

    ]

  }

)

```

Observe as diferenças entre o método usado para criar **joe** e o usado para criar **o representante** na etapa anterior. Desta vez, você atribui duas funções separadas:

1. readWrite aplicado ao banco de dados de relatórios significa que **Joe** será capaz de ler e escrever dados de relatórios de vendas para este banco de dados;
2. ler ao banco de dados de vendas garante que **joe** pode acessar os dados de vendas, mas não será capaz de escrever quaisquer documentos nesse banco de dados.

Esse comando retornará uma mensagem de confirmação semelhante à seguinte:

```
Output
Successfully added user: {
        "user" : "joe",
        "roles" : [
                {
                        "role" : "readWrite",
                        "db" : "relatorios"
                },
                {
                        "role" : "read",
                        "db" : "vendas"
                }
        ]
}
```

Em seguida, verifique se as permissões do novo usuário estão sendo devidamente aplicadas.

Mais uma vez, abra **outra sessão** de servidor, pois você fará uso tanto do usuário administrativo do MongoDB quanto do usuário **representante** em um passo posterior.

Abra o shell MongoDB, desta vez especificando **joe** como o usuário e relatorios como o banco de dados de autenticação:

```
mongo -u joe -p --authenticationDatabase relatorios
```

Quando solicitada, digite a senha definida ao criar o **usuário joe**. Uma vez que você tenha acesso ao shell prompt, execute o comando show dbs para listar os bancos de dados disponíveis para **Joe**:

```
show dbs
```

Como **Joe** pode usar os bancos de dados de vendas e relatórios, esses dois bancos de dados serão listados na saída:

```
Output
relatorios  0.000GB
vendas      0.000GB
```

Agora você pode verificar se **Joe** pode recuperar objetos do banco de dados de vendas.

Mude para vendas:

```
use vendas
```

Execute o seguinte comando para tentar recuperar todas os pedidos:

```
db.pedidos.find()
```

Supondo que você configure as permissões corretamente, este comando retornará o documento que você criou nesta coleção anteriormente.

```
Output
{ "_id" : ObjectId("60d890730d31cc50dedea6ff"), "total" : 100 }
```

Em seguida, tente inserir um novo documento na coleta de pedidos:

```
db.orders.insert({total: 50})
```

Como você atribuiu a Joe apenas uma função de leitura para este banco de dados, este comando de inserção falhará com uma mensagem de erro:

```
Output
WriteCommandError({
  "ok" : 0,
   "errmsg" : "not authorized on sales to execute command { insert: \"pedidos\", ordered: true, lsid: { id: UUID(\"ebbe853b-e269-463f-a1d4-2c5a5accb966\") }, $db: \"vendas\" }",
  "code" : 13,
  "codeName" : "Unauthorized"
})
```

A mensagem não autorizada diz a razão por trás da falha — os direitos de acesso que **Joe** tem não são suficientes para inserir um novo documento.

Em seguida, confirme se **Joe** pode ler e escrever dados no banco de dados de relatórios.

Mude para relatórios:

```
use relatorios
```

Em seguida, tente acessar os dados dentro dele, usando o comando find:

```
db.relatorios.find()
```

Como **Joe** pode ler no banco de dados, o MongoDB responderá com a lista de documentos disponíveis atualmente nesta coleção:

```
Output
{ "_id" : ObjectId("60d8897d6ae8ac2c9120ec41"), "pedidos" : 1 }
```

Em seguida, tente inserir um novo relatório, executando o seguinte comando:

```
db.relatorios.insert({pedidos: 2})
```

Esse comando também terá sucesso com a mensagem de saída semelhante a esta:

```
Output
WriteResult({ "nInserted" : 1 })
```

Com isso, você criou o segundo usuário de banco de dados com privilégios limitados, mas desta vez você lhes concedeu funções para dois bancos de dados separados. Você também verificou que seus direitos de acesso são devidamente aplicados pelo servidor de banco de dados. Posteriormente você concederá e, logo em seguida, revogará permissões adicionais a um dos usuários existentes.

> > 3.1.4. Atribuindo e revogando permissões aos usuários

Nas etapas 2 e 3, você criou novos usuários e os atribuiu a funções durante o processo de criação. Na prática, os administradores de banco de dados muitas vezes precisam revogar ou conceder novos privilégios aos usuários que já foram criados no sistema. Nesta etapa, você concederá ao usuário **representante** uma nova função para permitir que ele acesse o banco de dados de relatórios e revogue essa permissão pouco depois.

A partir **do shell administrativo**, mude para o banco de dados de vendas onde o usuário **representante** foi criado:

```
1.use vendas
2.
```

Para verificar se o usuário está lá, execute o comando dos usuários do show:

```
1.show users
2.
```

Esse comando retornará uma lista de todos os usuários nesse banco de dados, bem como suas respectivas funções:

```
Output
{
        "_id" : "vendas.representante",
        "userId" : UUID("cbc8ac18-37d8-4531-a52b-e7574044abcd"),
        "user" : "representante",
        "db" : "vendas",
        "roles" : [
                {
                        "role" : "readWrite",
                        "db" : "vendas"
                }
        ],
        "mechanisms" : [
                "SCRAM-SHA-1",
                "SCRAM-SHA-256"
        ]
}
```

Para atribuir uma nova função a este usuário, você pode usar o método grantRolesToUser. Este método aceita o nome do usuário e a lista de funções a serem concedidas na mesma sintaxe usada ao criar um novo usuário.

O objetivo nesta etapa **é conceder** permissões somente para relatórios, atribuindo-lhes a função de leitura para esse banco de dados:

```
1.db.grantRolesToUser("representante", [{role: "read", db:
  "relatorios"}])
2.
```

O comando não produz saída a menos que haja um erro, então, a falta de qualquer mensagem é um comportamento esperado.

Você pode verificar se o comando fez efeito executando o comando dos usuários do show novamente:

```
1. show users
2.
```

Esse comando retornará a saída semelhante à seguinte:

```
Output
{
        "_id" : "vendas.representante",
        "userId" : UUID("cbc8ac18-37d8-4531-a52b-e7574044abcd"),
        "user" : "representante",
        "db" : "vendas",
        "roles" : [
                {
                        "role" : "read",
                        "db" : "relatorios"
                },
                {
                        "role" : "readWrite",
                        "db" : "vendas"
                }
        ],
        "mechanisms" : [
                "SCRAM-SHA-1",
                "SCRAM-SHA-256" ]}
```

Observe a role recém-adicionada na seção de roles.

Agora você pode verificar se **o representante** é realmente capaz de acessar o banco de dados de relatórios após a mudança. Mude para a janela do terminal com **representante** logado e tente acessar os relatórios mais uma vez.

Mude para o banco de dados de relatórios:

```
use relatorios
```

E, em seguida, execute o comando find na coleção de relatórios:

```
db.relatorios.find()
```

Da última vez, o comando falhou com uma mensagem de erro. Desta vez, porém, ele retornará a lista de documentos no banco de dados de relatórios:

```
Output
{ "_id" : ObjectId("60d8897d6ae8ac2c9120ec41"), "pedidos" : 1 }
{ "_id" : ObjectId("60d899cafe3d26bf80e947fd"), "pedidos" : 2 }
```

Depois de algum tempo, você pode querer revogar a capacidade do usuário representante de acessar os relatórios. Para ilustrar isso, volte para o console administrativo e execute o seguinte comando, que revogará os privilégios de leitura do usuário representante no banco de dados de relatórios:

```
1. db.revokeRolesFromUser("representante", [{role: "read", db:
   "relatorios"}])
2.
```

O método revokeRolesFromUser leva o mesmo conjunto de argumentos que o grantRolesToUser, mas remove as funções especificadas.

Mais uma vez, você pode verificar se a função não está mais disponível com os usuários do show:

```
Output
{
        "_id" : "vendas.representante",
        "userId" : UUID("cbc8ac18-37d8-4531-a52b-e7574044abcd"),
        "user" : "representante",
        "db" : "vendas",
        "roles" : [
                {
                        "role" : "readWrite",
                        "db" : "vendas"
                }
        ],
        "mechanisms" : [
                "SCRAM-SHA-1",
                "SCRAM-SHA-256"
        ]
}
```

Para verificar novamente se o representante não pode mais ler no banco de dados de relatórios, tente reexecutar o comando anterior no shell com o representante logado:

```
db.relatorios.find()
```

Desta vez, o comando falhará novamente com um erro não autorizado:

```
Output
Error: error: {
  "ok" : 0,
  "errmsg" : "not authorized on reports to execute command { find:
\"relatorios\", filter: {}, lsid: { id: UUID(\"2c86fba2-7615-40ae-9c3b-
2dfdac2ed288\") }, $db: \"relatorios\" }",
  "code" : 13,
  "codeName" : "Unauthorized"
}
```

Isso mostra que sua revogação da função de leitura do usuário representante foi bem sucedida.

4. Operações de CRUD com o MongoDB

O MongoDB é um banco de dados persistente orientado a documentos usado para armazenar e processar dados na forma de documentos. Assim como outros sistemas de gerenciamento de banco de dados, o MongoDB permite gerenciar e interagir com dados por meio de quatro tipos fundamentais de operações de dados:

1. Operações de Create, que envolvem a inserção de dados para o banco de dados;
2. Operações Read, que consultam um banco de dados para recuperar dados dele;
3. Operações de Update, que alteram dados que já existem em um banco de dados;
4. Operações de Delete, que removem permanentemente dados de um banco de dados.

Conecte-se à instalação do MongoDB abrindo a concha MongoDB. Certifique-se de se conectar como um usuário do MongoDB com privilégios para escrever e ler dados.

```
1.mongo
2.
3.use database_name
4.
```

Agora que você se conectou ao servidor MongoDB usando um shell MongoDB, você pode passar a criar novos documentos.

> > 4.1. Criando Documentos

Para ter dados em que você pode praticar leitura, atualização e exclusão nas etapas posteriores deste guia, esta etapa se concentra em como criar documentos de dados no MongoDB.

Imagine que você está usando o MongoDB para construir e gerenciar um diretório de monumentos históricos famosos de todo o mundo. Este diretório armazenará informações como o nome de cada monumento, país, cidade e localização geográfica.

Os documentos deste diretório seguirão um formato semelhante ao seguinte exemplo, que representa **as Pirâmides de Gizé**:

The Pyramids of Giza

```
{
    "name": "The Pyramids of Giza",
    "city": "Giza",
    "country": "Egypt",
    "gps": {
        "lat": 29.976480,
        "lng": 31.131302
    }
}
```

Este documento, como todos os documentos do MongoDB, está escrito em BSON. BSON é uma forma binária de JSON, um formato de dados legível por humanos. Todos os dados nos documentos BSON ou JSON são representados como pares de campo e valor que tomam a forma de campo: valor.

Esse documento consiste em quatro campos. Primeiro é o nome do monumento, seguido pela cidade e pelo país. Todos os três campos contêm strings. O último campo, chamado GPS, é um documento aninhado que detalha a localização gps do monumento. Esse local é composto por um par de coordenadas de latitude e longitude, representadas pelos campos de lat e gnl, respectivamente, cada uma das quais possui valores de ponto flutuante.

Insira este documento em uma nova coleção chamada monumentos, usando o método insertOne. Como seu nome indica, insertOne é usado para criar documentos individuais, em vez de criar vários documentos ao mesmo tempo.

No shell MongoDB, execute a seguinte operação:

```
db.monuments.insertOne(

  {

    "name": "The Pyramids of Giza",

    "city": "Giza",

    "country": "Egypt",

        "gps": {

          "lat": 29.976480,

          "lng": 31.131302

        }

     }

  )

```

Observe que você não criou explicitamente a coleção de monumentos antes de executar este método insertOne. O MongoDB permite que você execute comandos em coleções inexistentes livremente, e as coleções que faltam só são criadas quando o primeiro objeto é inserido. Ao executar este exemplo insertOne() não só ele vai inserir o documento na coleção, mas também criará a coleção automaticamente.

O MongoDB executará o método insertOne e inserirá o documento solicitado representando as Pirâmides de Gizé. A saída da operação informará que ela foi executada com sucesso, e também fornecerá o ObjectId que ele gerou automaticamente para o novo documento:

```
Output
{
  "acknowledged" : true,
  "insertedId" : ObjectId("6105752352e6d1ebb7072647")
}
```

No MongoDB, cada documento dentro de uma coleção deve ter um campo _id único que atua como uma chave primária. Você pode incluir o campo _id e fornecer-lhe um valor de sua própria escolha, desde que você garanta que o campo _id de cada documento será único. No entanto, se um novo documento omitir o campo _id, o MongoDB gerará automaticamente um identificador de objeto (na forma de um objeto ObjectId) como o valor para o campo _id.

Você pode verificar se o documento foi inserido verificando a contagem de objetos na coleção dos monumentos:

```
1.db.monuments.count()
2.
```

Uma vez que você só inseriu um documento nesta coleção, o método de contagem retornará 1:

```
Output
1
```

Inserir documentos um a um como este rapidamente se tornaria tedioso se você quisesse criar vários documentos. O MongoDB fornece o método insertMany, que você pode usar para inserir vários documentos em uma única operação.

Execute o seguinte comando de exemplo, que usa o método insertMany para inserir seis monumentos famosos adicionais na coleção de monumentos:

```
db.monuments.insertMany([

  {"name": "The Valley of the Kings", "city": "Luxor", "country":
  "Egypt", "gps": { "lat": 25.746424, "lng": 32.605309 }},

  {"name": "Arc de Triomphe", "city": "Paris", "country": "France",
  "gps": { "lat": 48.873756, "lng": 2.294946 }},

  {"name": "The Eiffel Tower", "city": "Paris", "country": "France",
  "gps": { "lat": 48.858093, "lng": 2.294694 }},

  {"name": "Acropolis", "city": "Athens", "country": "Greece", "gps":
  { "lat": 37.970833, "lng": 23.726110 }},

  {"name": "The Great Wall of China", "city": "Huairou", "country":
  "China", "gps": { "lat": 40.431908, "lng": 116.570374 }},

  {"name": "The Statue of Liberty", "city": "New York", "country":
  "USA", "gps": { "lat": 40.689247, "lng": -74.044502 }}

])

```

Observe os colchetes ([e]) em torno dos seis documentos. Esses colchetes significam uma *série* de documentos. Dentro de suportes quadrados, vários objetos podem aparecer um após o outro, delimitados por vírgulas. Nos casos em que o método MongoDB requer mais de um objeto, você pode fornecer uma lista de objetos na forma de uma matriz como esta.

O MongoDB responderá com vários identificadores de objetos, um para cada um dos objetos recém-inseridos:

```
Output
{
  "acknowledged" : true,
  "insertedIds" : [
    ObjectId("6105770952e6d1ebb7072648"),
    ObjectId("6105770952e6d1ebb7072649"),
    ObjectId("6105770952e6d1ebb707264a"),
    ObjectId("6105770952e6d1ebb707264b"),
    ObjectId("6105770952e6d1ebb707264c"),
    ObjectId("6105770952e6d1ebb707264d")
  ]
}
```

Você pode verificar se os documentos foram inseridos verificando a contagem de objetos na coleção de monumentos:

```
1.db.monuments.count()
2.
```

Depois de adicionar esses seis novos documentos, a saída esperada deste comando é 7:

```
Output
7
```

Com isso, você usou dois métodos de inserção separados para criar uma série de documentos representando vários monumentos famosos. Em seguida, você lerá os dados que acabou de inserir com o método find() do MongoDB.

> > 4.2. Consultando Documentos

Agora que sua coleção tem alguns documentos armazenados dentro dela, você pode consultar seu banco de dados para recuperar esses documentos e ler seus dados. Esta etapa primeiro descreve como consultar todos os documentos em uma determinada coleção e, em seguida, descreve como usar filtros para reduzir a lista de documentos recuperados.

Depois de completar a etapa anterior, agora você tem sete documentos descrevendo monumentos famosos inseridos no acervo de monumentos. Você pode recuperar todos os sete documentos com uma única operação, usando o método find():

```
1.db.monuments.find()
2.
```

Esse método, quando usado sem argumentos, não aplica nenhuma filtragem e pede ao MongoDB que devolva todos os objetos disponíveis na coleção especificada: monumentos. O MongoDB retornará a seguinte saída:

```
Output

{ "_id" : ObjectId("6105752352e6d1ebb7072647"), "name" : "The Pyramids of
Giza", "city" : "Giza", "country" : "Egypt", "gps" : { "lat" : 29.97648,
"lng" : 31.131302 } }

{ "_id" : ObjectId("6105770952e6d1ebb7072648"), "name" : "The Valley of
the Kings", "city" : "Luxor", "country" : "Egypt", "gps" : { "lat" :
25.746424, "lng" : 32.605309 } }

{ "_id" : ObjectId("6105770952e6d1ebb7072649"), "name" : "Arc de
Triomphe", "city" : "Paris", "country" : "France", "gps" : { "lat" :
48.873756, "lng" : 2.294946 } }

{ "_id" : ObjectId("6105770952e6d1ebb707264a"), "name" : "The Eiffel
Tower", "city" : "Paris", "country" : "France", "gps" : { "lat" :
48.858093, "lng" : 2.294694 } }

{ "_id" : ObjectId("6105770952e6d1ebb707264b"), "name" : "Acropolis",
"city" : "Athens", "country" : "Greece", "gps" : { "lat" : 37.970833,
"lng" : 23.72611 } }

{ "_id" : ObjectId("6105770952e6d1ebb707264c"), "name" : "The Great Wall
of China", "city" : "Huairou", "country" : "China", "gps" : { "lat" :
40.431908, "lng" : 116.570374 } }

{ "_id" : ObjectId("6105770952e6d1ebb707264d"), "name" : "The Statue of
Liberty", "city" : "New York", "country" : "USA", "gps" : { "lat" :
40.689247, "lng" : -74.044502 } }
```

O shell do MongoDB imprime todos os sete documentos um por um e na íntegra. Observe que cada um desses objetos tem uma propriedade _id, que você não definiu. Como mencionado anteriormente, os campos _id servem como a chave principal de seus respectivos documentos, e foram criados automaticamente quando você executou o método insertMany na etapa anterior.

A saída padrão da concha MongoDB é compacta, com os campos e valores de cada documento impressos em uma única linha. Isso pode se tornar difícil de ler com objetos contendo vários campos ou documentos aninhados, em particular.

Para tornar a saída do método find() mais legível, você pode usar o recurso de impressão pretty, como este:

```
1.db.monuments.find().pretty()
2.
```

Desta vez, o shell MongoDB imprimirá os documentos em várias linhas, cada uma com recuo:

```
Output
{
  "_id" : ObjectId("6105752352e6d1ebb7072647"),
  "name" : "The Pyramids of Giza",
  "city" : "Giza",
  "country" : "Egypt",
  "gps" : {
    "lat" : 29.97648,
    "lng" : 31.131302
  }
}
{
  "_id" : ObjectId("6105770952e6d1ebb7072648"),
  "name" : "The Valley of the Kings",
  "city" : "Luxor",
  "country" : "Egypt",
  "gps" : {
    "lat" : 25.746424,
    "lng" : 32.605309
  }
}
. . .
```

Observe que nos dois exemplos anteriores, o método find() foi executado sem argumentos. Em ambos os casos, ele devolveu todos os objetos da coleção. Você pode aplicar filtros a uma consulta para reduzir os resultados.

Lembre-se dos exemplos anteriores de que o MongoDB atribuiu automaticamente **ao Vale dos Reis** um identificador de objetos com o valor de ObjectId ("6105770952e6d1ebb72648"). O identificador de objeto não é apenas a sequência hexaponica dentro do ObjectId("), mas todo o objeto ObjectId — um tipo especial de dados usado no MongoDB para armazenar identificadores de objetos.

O método find() a seguir retorna um único objeto, aceitando um documento de filtro de consulta como argumento. Os documentos do filtro de consulta seguem a mesma estrutura que os documentos inseridos em uma coleção, consistindo em campos e valores, mas eles são usados para filtrar resultados de consulta.

O documento de filtro de consulta usado nesse exemplo inclui o campo _id, com o identificador **de objeto do Vale dos Reis** como o valor. Para executar essa consulta em seu próprio banco de dados, certifique-se de substituir o identificador de objeto destacado com o de um dos documentos armazenados em sua própria coleção de monumentos:

```
1. db.monuments.find({"_id":
   ObjectId("6105770952e6d1ebb7072648")}).pretty()
2.
```

O documento do filtro de consulta neste exemplo usa a condição de igualdade, o que significa que a consulta devolverá quaisquer documentos que tenham um par de campo e um valor correspondente ao especificado no documento. Essencialmente, este exemplo diz ao método find() para retornar apenas os documentos cujo valor _id é igual ao ObjectId ("6105770952e6d1ebb72648").

Após a execução desse método, o MongoDB retornará um único objeto correspondente ao identificador de objeto solicitado:

```
Output
{
  "_id" : ObjectId("6105770952e6d1ebb7072648"),
  "name" : "The Valley of the Kings",
  "city" : "Luxor",
  "country" : "Egypt",
  "gps" : {
    "lat" : 25.746424,
    "lng" : 32.605309
  }
}
```

Você pode usar a condição de qualidade em qualquer outro campo do documento também. Para ilustrar, tente procurar monumentos na França:

```
1.db.monuments.find({"country": "France"}).pretty()
2.
```

Esse método devolverá dois monumentos:

```
Output
{
  "_id" : ObjectId("6105770952e6d1ebb7072649"),
  "name" : "Arc de Triomphe",
  "city" : "Paris",
  "country" : "France",
  "gps" : {
    "lat" : 48.873756,
    "lng" : 2.294946
  }
}
{
  "_id" : ObjectId("6105770952e6d1ebb707264a"),
  "name" : "The Eiffel Tower",
  "city" : "Paris",
  "country" : "France",
  "gps" : {
    "lat" : 48.858093,
    "lng" : 2.294694
  }
}
```

Os documentos do filtro de consulta são bastante poderosos e flexíveis, e permitem que você aplique filtros complexos em documentos de coleções.

> > 4.3. Alterando Documentos

É comum que documentos dentro de um banco de dados orientado a documentos como o MongoDB mudem ao longo do tempo. Às vezes, suas estruturas devem evoluir junto com as mudanças nos requisitos de um aplicativo, ou os dados em si podem mudar. Esta etapa se concentra em como atualizar os documentos existentes alterando os valores de campo em documentos individuais, bem como adicionando um novo campo a cada documento em uma coleção.

Semelhante aos métodos insertOne() e insertMany(), o MongoDB fornece métodos que permitem atualizar um único documento ou vários documentos ao mesmo tempo. Uma diferença importante com esses métodos de atualização é que, ao criar novos documentos, você só precisa passar os dados do documento como argumentos de método. Para atualizar um documento existente na coleção, você também deve passar por um argumento que especifica qual documento você deseja atualizar.

Para permitir que os usuários façam isso, o MongoDB usa o mesmo mecanismo de documento de filtro de consulta em métodos de atualização que o que você usou na etapa anterior para encontrar e recuperar documentos. Qualquer documento de filtro de consulta que possa ser usado para recuperar documentos também pode ser usado para especificar documentos para atualizar.

Tente mudar o nome de **Arc de Triomphe** para o nome completo de **Arc de Triomphe de l'Étoile**. Para isso, use o método updateOne() que atualiza um único documento:

```
db.monuments.updateOne(

  { "name": "Arc de Triomphe" },

  {

    $set: { "name": "Arc de Triomphe de l'Étoile" }

  }

)

```

O primeiro argumento do método updateOne é o documento do filtro de consulta com uma única condição de igualdade, conforme abordado na etapa anterior. Nesse exemplo { "nome": "Arc de Triomphe" } encontra-se documentos com chave de nome segurando o valor de Arc de Triomphe. Qualquer documento de filtro de consulta válido pode ser usado aqui.

O segundo argumento é o documento de atualização, especificando quais alterações devem ser aplicadas. O documento consiste em operadores de atualização como chaves e parâmetros para cada um dos valores da operadora. Neste exemplo, o operador de atualização usado é $set. Ele é responsável por definir campos de documentos para novos valores e requer um objeto JSON com novos valores de campo. Aqui, definir: { "nome": "Arc de Triomphe de l'Étoile" } diz mongoDB para definir o valor do nome de campo para Arc de Triomphe de l'Étoile.

O método retornará um resultado dizendo que um objeto foi encontrado pelo documento do filtro de consulta, e também que um objeto foi atualizado com sucesso.

```
Output
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
```

ⓘ ***Nota:*** *Se o filtro de consulta de documentos não for preciso o suficiente para selecionar um único documento, o updateOne() atualizará apenas o primeiro documento retornado de vários resultados.*

Para verificar se a atualização funcionou, tente recuperar todos os monumentos relacionados à França:

```
1.db.monuments.find({"country": "France"}).pretty()
2.
```

Desta vez, o método retorna **Arc de Triomphe**, mas com seu nome completo, que foi alterado pela operação de atualização:

```
Output
{
  "_id" : ObjectId("6105770952e6d1ebb7072649"),
  "name" : "Arc de Triomphe de l'Étoile",
  "city" : "Paris",
  "country" : "France",
  "gps" : {
    "lat" : 48.873756,
    "lng" : 2.294946
  }
}
. . .
```

Para modificar mais de um documento, você pode usar o método updateMany().

Como exemplo, não há informações sobre quem criou a entrada e você gostaria de creditar ao autor que adicionou cada monumento ao banco de dados. Para fazer isso, você adicionará um novo campo de editor a cada documento da coleção de monumentos.

O exemplo a seguir inclui um documento de filtro de consulta vazio. Ao incluir um documento de consulta vazio, esta operação corresponderá a todos os documentos da coleção, e o método updateMany() afetará cada um deles . O documento de atualização adiciona um novo campo de editor a cada documento e lhe atribui um valor de Sammy:

```
db.monuments.updateMany(

  { },

  {

    $set: { "editor": "Sammy" }

  }

   )

```

Esse método retornará a seguinte saída:

```
Output
{ "acknowledged" : true, "matchedCount" : 7, "modifiedCount" : 7 }
```

Essa saída informa que sete documentos foram combinados e sete também foram modificados.

Confirme se as alterações foram aplicadas:

```
db.monuments.find().pretty()

```

```
Output
{
  "_id" : ObjectId("6105752352e6d1ebb7072647"),
  "name" : "The Pyramids of Giza",
  "city" : "Giza",
  "country" : "Egypt",
  "gps" : {
    "lat" : 29.97648,
    "lng" : 31.131302
  },
  "editor" : "Sammy"
}
{
  "_id" : ObjectId("6105770952e6d1ebb7072648"),
  "name" : "The Valley of the Kings",
  "city" : "Luxor",
  "country" : "Egypt",
  "gps" : {
    "lat" : 25.746424,
    "lng" : 32.605309
  },
  "editor" : "Sammy"
}
. . .
```

Todos os documentos devolvidos agora têm um novo campo chamado editor definido para Sammy. Ao fornecer um nome de campo não existente para o operador de atualização $set, a operação de atualização criará campos ausentes em todos os documentos combinados e definirá adequadamente o novo valor.

Embora você provavelmente use $set na maioria das vezes, muitos outros operadores de atualização estão disponíveis no MongoDB, permitindo que você faça alterações complexas nos dados e na estrutura de seus documentos. Você pode saber mais sobre essas operadoras de atualização na documentação oficial do MongoDB sobre o assunto.

> > 4.4. Excluindo Documentos

Há momentos em que os dados no banco de dados se tornam obsoletos e precisam ser excluídos. Assim como nas operações de atualização e inserção do Mongo, existem os métodos deleteOne(), que remove apenas o *primeiro* documento compatível com o documento do filtro de consulta, e deleteMany(), que exclui vários objetos ao mesmo tempo.

Para praticar usando esses métodos, comece tentando remover o monumento **Arc de Triomphe de l'Étoile** que você modificou anteriormente:

```
1.db.monuments.deleteOne(
2.
3.    { "name": "Arc de Triomphe de l'Étoile" }
4.
5.)
6.
```

Observe que este método inclui um documento de filtro de consulta, como os exemplos de atualização e recuperação anteriores. Como antes, você pode usar qualquer consulta válida para especificar quais documentos serão excluídos.

O MongoDB retornará o seguinte resultado:

```
Output
{ "acknowledged" : true, "deletedCount" : 1 }
```

Aqui, o resultado informa quantos documentos foram excluídos no processo.

Verifique se o documento foi de fato removido da coleção consultando monumentos na França:

```
1.db.monuments.find({"country": "France"}).pretty()
2.
```

Dessa vez, o método retorna apenas um único monumento, **a Torre Eiffel**, desde que você removeu o **Arco do Triunfo de l'Étoile**:

```
Output
{
  "_id" : ObjectId("6105770952e6d1ebb707264a"),
  "name" : "The Eiffel Tower",
  "city" : "Paris",
  "country" : "France",
  "gps" : {
    "lat" : 48.858093,
    "lng" : 2.294694
  },
  "editor" : "Sammy"
}
```

Para ilustrar a remoção de vários documentos ao mesmo tempo, remova todos os documentos do monumento para os quais Sammy era o editor. Isso esvaziará a coleção, como você já designou Sammy como o editor de cada monumento:

```
1.db.monuments.deleteMany(
2.
3.  { "editor": "Sammy" }
4.
5.)
6.
```

Desta vez, o MongoDB permite que você saiba que esse método removeu seis documentos:

```
Output
{ "acknowledged" : true, "deletedCount" : 6 }
```

Você pode verificar se a coleção de monumentos está agora vazia, contando o número de documentos dentro dele:

```
1.db.monuments.count()
2.
```

```
Output
0
```

Uma vez que você acabou de remover todos os documentos da coleção, esse comando retorna a saída esperada de 0.

Referências bibliográficas

FLOWER, Martin. **NoSQL – Um Guia Conciso para o Mundo Emergente da Persistência Poliglota**. 1ª ed. São Paulo: Novatec, 2013.

HOWS, David; MEMBREY, Peter; PLUGGE, Eelco. **Introdução ao MongoDB**. 2ª ed. São Paulo: Novatec, 2017.

Digital College

ENSINO DE HABILIDADES DIGITAIS

digitalcollege.com.br