BANCO DE DADOS NoSQL COM MONGODB

CREDENCIADA JUNTO AO MEC PELA PORTARIA
N 3.455 DO DIA 19/11/2003

SUMÁRIO

NOSSA HISTÓRIA

A nossa história inicia com a realização do sonho de um grupo de empresários, em atender à crescente demanda de alunos para cursos de Graduação e Pós-Graduação. Com isso foi criado a nossa instituição, como entidade oferecendo serviços educacionais em nível superior.

A instituição tem por objetivo formar diplomados nas diferentes áreas de conhecimento, aptos para a inserção em setores profissionais e para a participação no desenvolvimento da sociedade brasileira, e colaborar na sua formação contínua. Além de promover a divulgação de conhecimentos culturais, científicos e técnicos que constituem patrimônio da humanidade e comunicar o saber através do ensino, de publicação ou outras normas de comunicação.

A nossa missão é oferecer qualidade em conhecimento e cultura de forma confiável e eficiente para que o aluno tenha oportunidade de construir uma base profissional e ética. Dessa forma, conquistando o espaço de uma das instituições modelo no país na oferta de cursos, primando sempre pela inovação tecnológica, excelência no atendimento e valor do serviço oferecido.

1. INTRODUÇÃO

MongoDB é um banco de dados de código aberto, gratuito, de alta performance, sem esquemas e orientado à documentos, lançado em fevereiro de 2009 pela empresa 10gen. Foi escrito na linguagem de programação C++ (o que o torna portável para diferentes sistemas operacionais) e seu desenvolvimento durou quase 2 anos, tendo iniciado em 2007. Classificado como um programa de banco de dados NoSQL, o MongoDB usa documentos semelhantes a JSON com esquemas. É desenvolvido pela MongoDB Inc. e publicado sob uma combinação da GNU Affero General Public License e Licença Apache. Suas características permitem com que as aplicações modelem informações de modo muito mais natural, pois os dados podem ser aninhados em hierarquias complexas e continuar a ser indexáveis e fáceis de buscar.

1.1 Banco de Dados Orientado a Documentos

A definição geral apresentada é que os Bancos de Dados orientados a Documentos utilizam o conceito de dados e documentos autocontidos e auto descritivos, e isso implica que o documento em si já define como ele deve ser apresentado e qual é o significado dos dados armazenados na sua estrutura.

Banco de Dados Orientados a Documentos tem como característica conter todas as informações importantes em um único documento, ser livre de esquemas, possuir identificadores únicos universais (UUID), possibilitar a consulta de documentos através de métodos avançados de agrupamento e filtragem (MapReduce) e também permitir redundância e inconsistência.

Esses bancos de dados também são chamados de Bancos NoSQL (Not Only SQL). Esse termo NoSQL é devido à ausência do SQL, mas esse tipo de Banco de Dados não se resume apenas a isso, por isso o termo não é o mais correto para esse novo tipo de Banco de Dados. No entanto, ele é aceito porque o termo já se popularizou na comunidade. Alguns chegaram a defender o termo NoREL (Not Relational), mas diferente do anterior este não foi muito aceito. De forma resumida esse tipo de Banco de Dados não traz consigo as ideias do modelo relacional e nem

a linguagem SQL. Uma diferença fundamental entre os dois modelos surge quando precisamos criar relacionamentos nos bancos de dados relacionais, diferente dos bancos orientados a documentos que não fornecem relacionamentos entre documentos, o que mantém seu design sem esquemas. Dessa forma, ao invés de armazenar dados relacionados em uma área de armazenamento separado, os bancos de dados de documentos integram esses dados ao próprio documento. Contudo, ainda assim é possível armazenar dados relacionados de forma separada, isso pode ser feito usando uma coleção separada, porém, é preciso tomar cuidado com isso visto que os ganhos de desempenhos podem ser perdidos.

Existem diversos Banco de Dados NoSQL hoje, os mais conhecidos são: MongoDB, DynamoDB, Azure Table Storage, Berkeley DB, Hadoop, Cassandra, Hypertable, Amazon SimpleDB, CouchDB, RavenDB, Neo4J, Infinite Graph, InforGrid, etc..

Figura 1.1 – MongoDB

1.2 A MapReduce

O MapReduce é um modelo de programação que permite o processamento de dados massivos em um algoritmo paralelo e distribuído, geralmente em um cluster de computadores. Hoje, o Hadoop é utilizado em larga escala por grandes corporações, como Facebook e Twitter, em aplicações Big Data. Este tema será útil para aplicações que envolvam dados massivos para processamento paralelo (embora seja

interessante para processamento de quaisquer dados), geralmente utilizando um cluster de computadores.

A capacidade dos discos rígidos e outros elementos de armazenamento aumentaram bastante nos últimos anos, mas a velocidade de leitura e escrita dos mesmos não acompanhou o mesmo ritmo. Como um exemplo, a leitura de todo um disco rígido 20 anos atrás levava cerca de cinco minutos. Atualmente, leva mais de duas horas e meia. Trata-se de um longo período para ler todos os dados, e escrever é ainda mais lento. A solução mais óbvia para resolver esse problema é ler/escrever os dados em paralelo, utilizando vários discos. Deste modo, se existem 100 HDs, cada um com 1% do total dos dados, por exemplo, a leitura pode ser realizada 100 vezes mais rapidamente, em teoria.

No caso dessa leitura/escrita paralela dos dados, dois problemas são bastante comuns. O primeiro é bastante óbvio: se há 100 vezes mais discos rígidos, a chance de existir falha em um deles é 100 vezes maior, o que pode ocasionar perda de dados. Para evitar esse problema, geralmente utiliza-se a replicação, onde cópias de segurança dos dados são mantidas em diferentes discos. Outro problema é que muitas tarefas de análise de dados necessitam combinar dados “espalhados” em discos diferentes. Entretanto, esses problemas não geram dores de cabeça aos programadores, pois o MapReduce oferece um modelo de programação que os abstrai, uma vez que o processamento é realizado através de uma combinação entre chaves e valores (keys e values) que podem estar em diferentes discos.

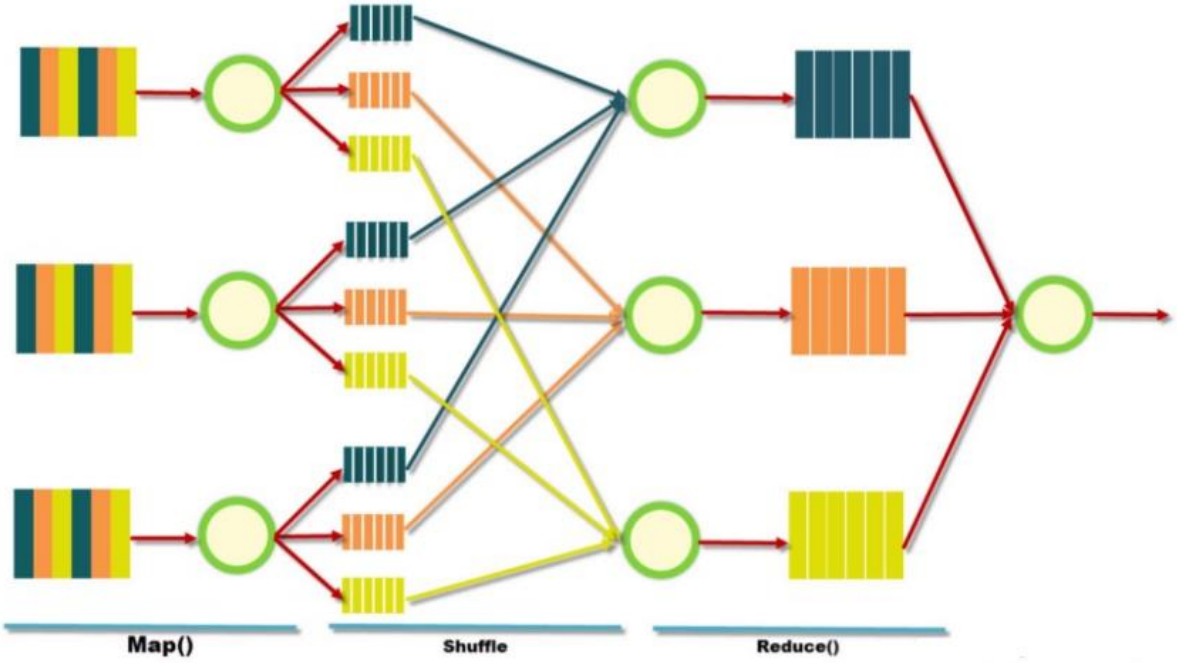

Figura 1.2 – MapReduce

1.3 Introdução à tecnologia NoSQL

Constantemente, até mesmo os profissionais da área preferem tornarem-se céticos usuários dos SGBD's (Sistemas de Gerenciamento de Bancos de Dados) puramente relacionais, para resolver problemas com estruturas muito dispares ao paradigma relacional, causando limitações e trabalho excessivamente desnecessário.

Ferramentas NoSQL fornecem meios mais eficientes de armazenamento de grandes volumes de dados e/ou mecanismos de pesquisa de baixa latência, fatores importantes que precisam ser considerados durante a escolha de uma solução de armazenamento de dados (PORCELLI, 2011, p.21).

Não se trata apenas de uma linguagem, mas sim de um conjunto de ferramentas e estruturas. "NoSQL é um movimento que promove soluções de armazenamento de dados não relacionais." (PORCELLI, 2011).

Esse conjunto consiste em diversas tecnologias capazes de resolver certos problemas de forma mais específica, abordando, para tal, cada cenário de uma forma bem particular. Contudo, o objetivo do NoSQL não é substituir a linguagem SQL, como

muitos pensam. Sua proposta é (como o nome denomina: not only SQL – não apenas SQL) usar também modelos não-relacionais, para trazer a melhor solução para um determinado problema.

Segundo Porcelli (2011), desta forma, é possível trabalhar com tecnologias NoSQL e banco de dados relacionais dentro de uma mesma aplicação.

1.4 Por que NoSQL?

O modelo relacional começou a ser utilizado na década de 70 e se consagrou pela sua maior facilidade de compreensão do modelo, de manutenção, e de modelagem. O problema é que este modelo, na maioria dos casos, ainda é visto como a solução de todos os problemas.

Segundo Puente (2011), algumas soluções simplesmente não devem ser modeladas em tabelas. O motivo, apenas, é que esta não é a melhor representação dessa solução, não é o formato (ou modelo) mais adequado. E essa é a principal mensagem a se destacar sobre NoSQL.

Suponha um site de buscas como o Google, que cataloga e armazena informações sobre bilhões de páginas na web em servidores localizados em diversos lugares do planeta, e, quando alguém faz uma pesquisa, retorna essas informações em questão de milésimos de segundos, e em ordem de relevância para o usuário. Agora pode-se notar claramente o ponto chave da questão: essa eficiência seria viável num mundo feito de tabelas, num paradigma onde os dados se encontram separados e normalizados? Para retornar esses resultados seriam executados dezenas de join’s.

Visivelmente a melhor saída é uma desnormalização dos dados, e mais, a utilização de uma estrutura de persistência de dados que se adapte melhor ao problema proposto, que torne a sua resolução mais apropriada ao cenário. O que existe não é “a melhor tecnologia”, mas sim, a tecnologia que melhor se adequa a um determinado cenário.

A estrutura altamente rígida dos SGBD’s relacionais se deve, em grande parte, ao fato de que, na época que a mesma surgiu, os recursos eram muito limitados. As

conexões de internet discada eram lentas e caras, discos para armazenamento de dados, servidores, etc. Cada byte de memória temporária e de espaço em disco era precioso. Atualmente, não se perde muito por fugir das regras impostas por um modelo, afinal já se pode ter um servidor com espaço de memória razoável, sem pagar quantias exorbitantes por isso.

As estruturas NoSQL valem a pena quando grandes volumes de dados devem ser manipulados, para obter desempenho ao gravá-los nessa quantidade, e acessá-los com rapidez, ou quando os dados envolvidos não podem seguir um esquema claro, não possuem uma estrutura bem definida.

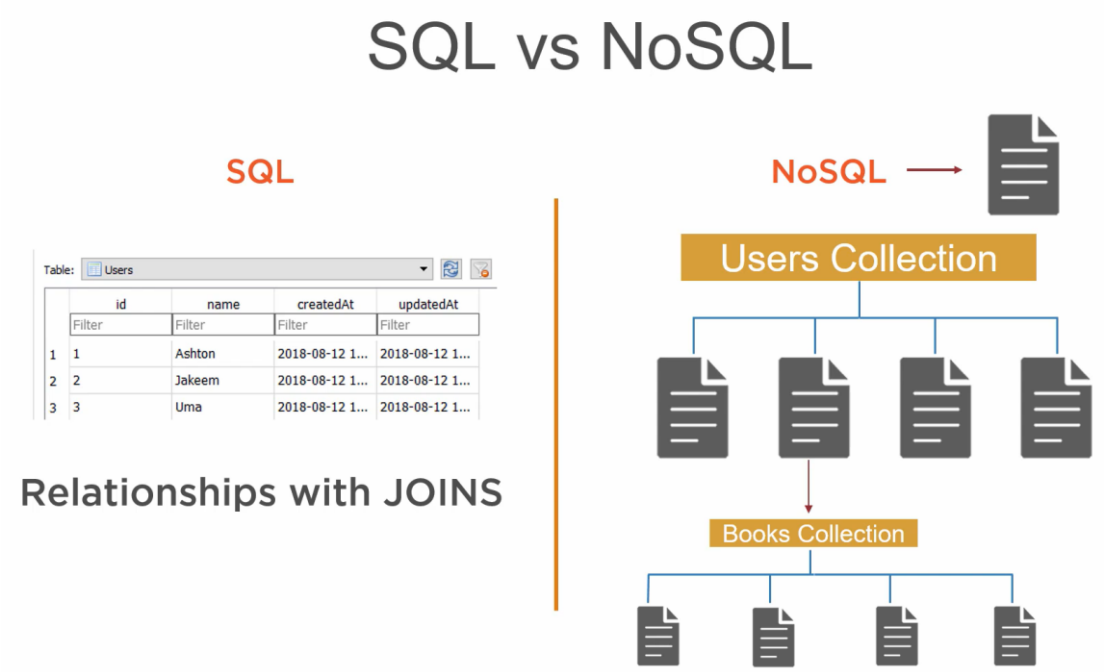

Figura 1.4 – SQL vs NoSQL

1.5 Principais modelos NoSQL

A tecnologia NoSQL foi propriamente denominada no ano de 2009, apesar de haver trabalhos falando sobre o uso de bancos de dados que não possuíam interface SQL desde 1998. Ao longo dessa evolução, existem hoje vários modelos de bancos NoSQL: Chave-Valor, Tabular, Documento e Grafos.

1.5.1 Modelo Chave-Valor

O armazenamento KV usa uma matriz associativa, também conhecida como mapa ou dicionário como base para seu modelo de dados. Nesse modelo, os dados são apresentados como uma coleção de currículos emparelhados.

É um modelo bem simples, que se baseia em tabelas de hash para garantir que cada registro seja armazenado com uma chave única. Esta chave está posicionada antes do próprio conteúdo do registro, agilizando sua busca, e possibilitando uma estrutura flexível para armazenar um dado. Muito utilizada para a gravação de logs.

1.5.2 Modelo Baseado em Documentos

Modelo muito parecido com o de Chave-Valor, só que mais complexo. Um documento é uma estrutura maleável, modelada pelo programador, que permite a ele ajustar e otimizar a forma como seus dados ficarão armazenados. Podemos comparar um documento a um formulário comum, com as vantagens de que é possível adicionar e remover campos e atributos deste, bem como modificar sua estrutura de forma dinâmica, em tempo de execução. Tabelas em linguagem SQL não possuem esta flexibilidade. Usado em larga escala por empresas que utilizam catálogos de livros e documentos escritos, que precisam armazenar cada item, e possivelmente manter um histórico de todas as alterações feitas em cada registro ao longo do tempo.

1.5.3 Modelo Tabular

Voltado para computação distribuída, eficiente ao armazenar grandes quantidades de dados separando-os em muitas máquinas. Para isso, utiliza as famosas Big Tables, tabelas gigantes onde as chaves apontam para várias colunas distintas. Preferível para motores de buscas, como o Google.

1.5.4 Modelo Baseado em Grafos

Diferente dos bancos de dados relacionais o Banco de dados orientado a grafos possui outras formas de persistência de dados o NoSQL. A ideia é criar um modelo menos genérico que o modelo relacional, proporcionando uma modelagem mais simples, buscando obter maior performance, tanto por sua implementação livre de operações custosas como JOINs, quanto pelo uso de algoritmos de grafos. Sendo muito mais simples de desenhar ele não precisa de um design complexo de tabelas para começar a incluir os dados.

Em geral, as estruturas baseadas em gráficos consistem em vértices e arestas (dados e conexões) que podem ser chamadas de relacionamentos de dados. Os gráficos se comportam da mesma forma que os humanos pensam, os dados são organizados em relacionamentos específicos entre as unidades de dados que têm suas próprias características. Esse tipo de banco de dados é útil para visualizar, analisar e ajudá-lo a encontrar conexões entre diferentes dados.

Baseado em relacionamentos entre nós, os quais possuem possibilidade de mudança de formato individual. Torna-se fácil fazer conexões entre eles, assemelhando-se ao conceito de banco com registros encadeados. Sua estrutura é ideal para a modelagem de redes sociais.

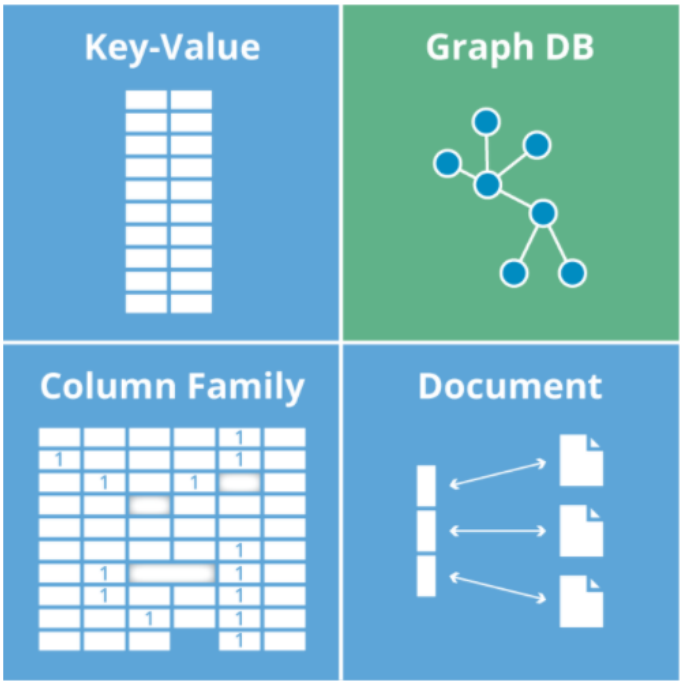

Figura 1.5 – Modelos NoSQL

1.6 Características do MongoDB

O MongoDB é um banco de dados orientado a documentos, diferente dos Bancos de dados tradicionais que seguem o modelo relacional.

Dessa forma, já temos uma primeira diferença entre os dois modelos, onde o Banco orientado a documentos lida com documentos e não com registros como no modelo relacional onde tudo é representado usando uma abordagem bidimensional (tabelas representadas através de duas dimensões: linhas e colunas).

Este Banco de Dados tem como característica ser código-fonte aberto licenciado pela GNU AGPL (Affero General Public License) versão 3.0, possuir alta performance, não possuir esquemas, ser escrito em C++, multiplataforma e ser formado por um conjunto de aplicativos JSON. Apesar do projeto MongoDB ter iniciado em 2007 o Banco de Dados somente foi concluído em 2009 lançando assim a primeira versão do MongoDB. Diversas linguagens e plataforma já possuem drivers para o MongoDB, entre elas destacam-se: C, C#, C++, Haskell, Java, JavaScript, Perl, PHP, Python, Ruby e Scala. Além disso, o MongoDB possui binários para diversas plataformas como Windows, Mac OS X, Linux e Solaris.

Entre as empresas que já utilizam o MongoDB destacam-se: Globo.com, SourceForge, FourSquare, MailBox (serviço de e-mail do Dropbox), LinkedIn, SAP, MTV, Pearson Education, e muitos outros. Uma lista com todos os serviços em ambiente de produção que estão utilizando o MongoDB pode ser encontrado aqui.

2. INSTALANDO O MONGODB

Para instalar o MongoDB devemos primeiramente baixa-lo, escolhendo uma versão de sistema operacional. Agora já podemos descompactar o zip para alguma pasta.

Após isso devemos ir até a pasta onde descompactamos o MongoDB e ir na pasta “bin”. Por fim executamos o “mongod.exe”. Agora já podemos executar o cliente de Shell do MongoDB. O aplicativo de Shell do MongoDB está incluído junto com a distribuição sendo localizado na pasta “bin”. No Windows, está na forma do aplicativo mongo.exe.

Através deste Shell podemos criar bancos de dados, documentos e coleções. Se em qualquer momento for preciso obter ajuda, basta dar o comando "help" na linha de comando do Shell do Mongo.

Por padrão, o Shell do Mongo se conecta ao banco de dados "test". Para mudar para outro banco de dados, usamos o comando "use nomeDoBanco". Se o banco de dados não existir o MongoDB o criará assim que forem incluídos dados nele. O Shell deve apresentar a mensagem: switched to db meuMongoDB.

MongoDB (de Humongous) é o banco de dados NoSQL mais utilizado no mundo! Seu código é aberto, é possível usá-lo em hardware commoditie e em múltiplas plataformas, é possível escalar horizontalmente (aumentar o desempenho do banco de dados adicionando mais nós em um cluster) e é um banco de dados orientado a documentos.

O MongoDB não armazena dados no formato da tabela, seus dados são semiestruturados.

Ele recebe documentos JSON (Java Script Object Notation), converte-os para o formato BSON (Binário JASON) e os armazena neste formato.

Aqui eu preciso dizer que o MongoDB não é perfeito para todas as situações! Ele é perfeito para armazenar dados semiestruturados!

O processo de instalação é muito simples… No mesmo esquema do “Next -> Next -> Next”!!! Very easy!

Veja a sequência de imagens abaixo:

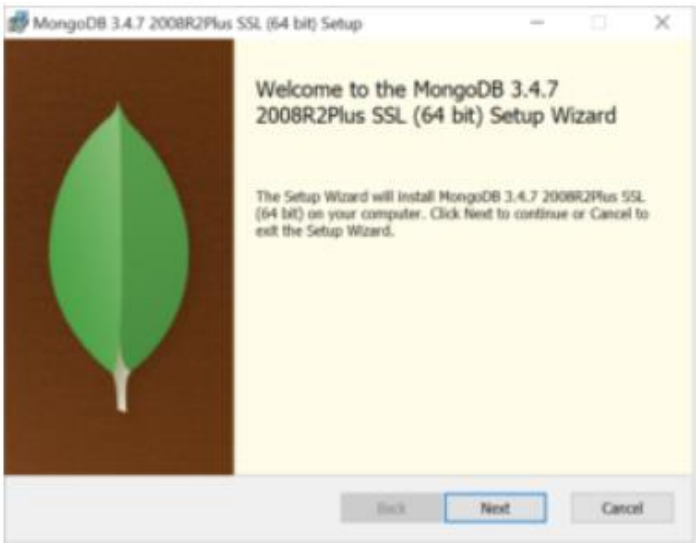

Figura 2 – Instalação passo 1

Figura 2 – Instalação passo 2

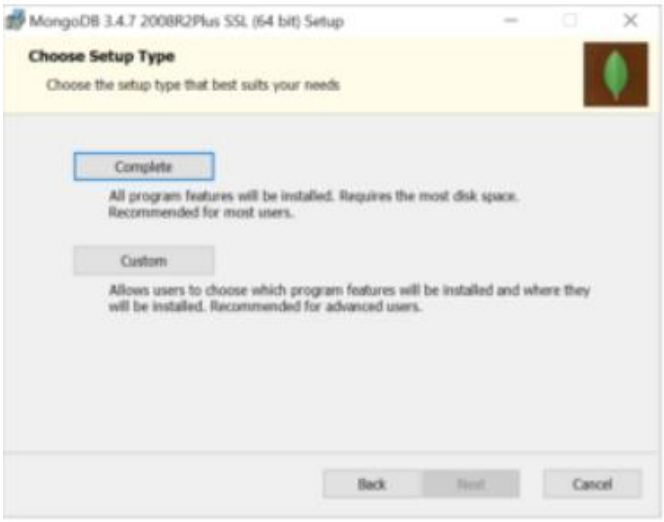

Figura 2 – Instalação passo 3

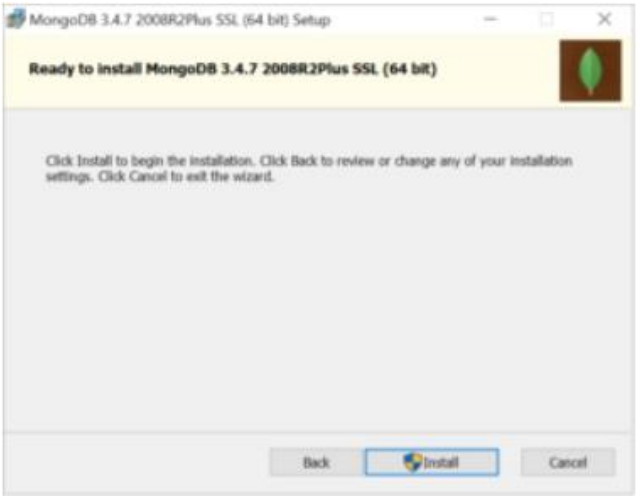

Figura 2 – Instalação passo 4

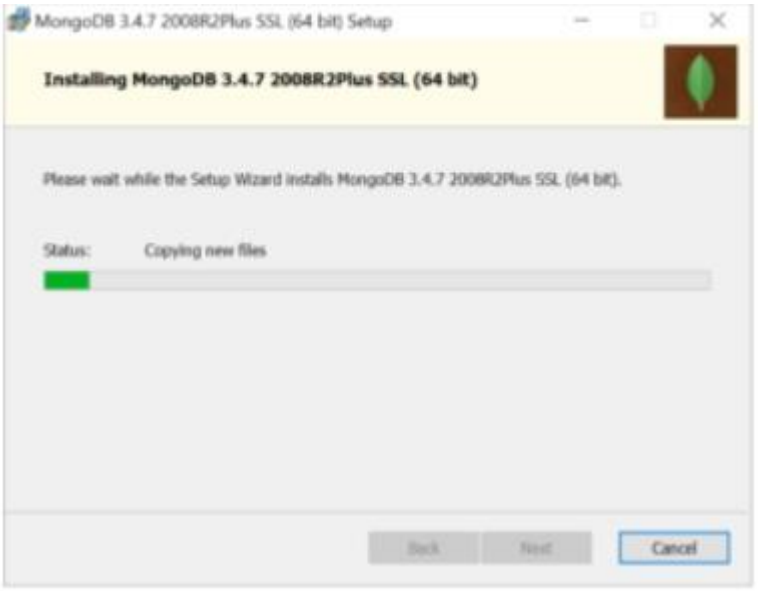

Figura 2 – Instalação passo 5

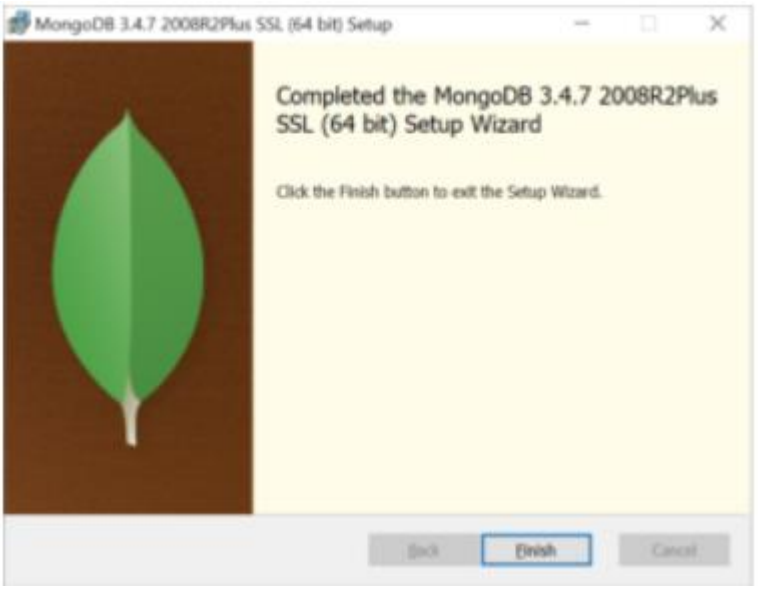

Figura 2 – Instalação passo 6

2.1 Diretório de Dados

Antes de iniciar pela primeira vez, o serviço MongoDB é necessário criar manualmente o diretório de dados. O diretório de dados deve estar no volume raiz da instalação.

Por exemplo, se você instalou o MongoDB no C:\, o diretório de dados estará dentro de C:\. Por padrão, você precisa criar um diretório de dados (data) e, dentro dele, crie o diretório db.

Em nosso exemplo, o diretório de dados é C:\data\db. É importante ter permissão total neste diretório.

Se você quiser usar outro diretório para ser diretório de dados é possível! Mas quando você precisará iniciar o mongod pela linha de comando e usar o parâmetro –dbpath.

Exemplo:> C:\MongoDB\bin\mongod.exe

–dbpath C:\MongoDB\NovoDiretorioDados

Se você não quiser usar o diretório de dados padrão, não deseja usar o parâmetro –dbpath, há uma terceira opção, use um arquivo de configuração e, dentro dele, especifique o diretório de dados.

Este arquivo possui formato YAML e pode conter todas as opções usadas na linha de comando. Se quiser conhecer todas as opções, consulte o site oficial do MongoDB (https://docs.mongodb.com/manual/reference/configuration-options/).

Para usar o arquivo de configuração na linha de comando, existem duas opções:

> C:\MongoDB\bin\mongod.exe -f C:\MongoDB\mongod.conf

> C:\MongoDB\bin\mongod.exe –config C:\MongoDB\mongod.conf

2.2 Tópicos de instalação

- Para instalar o MongoDB faça um download no site oficial;
- A versão da comunidade é gratuita;
- A instalação é semelhante a “Next -> Next -> Next”;
- É necessário criar um diretório de dados;
- O caminho padrão para o diretório de dados é Volume\data\db;

- É possível usar outros diretórios para o diretório de dados, mas na execução de mongod é necessário usar o parâmetro – dbpath com o novo path do diretório de dados.
- É possível usar um arquivo de configuração e, neste, especifique o diretório de dados. Para usar esta opção no comando mongod use o parâmetro -f ou –config.

2.3 Testes e comandos

Diversos players de cloud computing fornecem versões de Mongo hospedadas e prontas para uso como Umbler e mLab, 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 e baixe gratuitamente a versão mais recente para o seu sistema operacional, que é a versão 4.4 na data em que foi desenvolvido este material.

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 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).

```
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:

```
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-o. 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:

```
> 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):

```
> 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:

```
> 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:

```
> 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, algo que iremos fazer agora com a função insert:

```
> 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!

2.4 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:

```
> 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 traz 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”?

2.5 FIND Avançado

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

```
> db.customers.find().pretty()
```

É o mesmo comando find() que usamos anteriormente, mas com a função pretty() no final para indentar 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 guardá-los na 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:

```
> 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:

```
> 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?

```
> 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:

```
> 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:

```
> 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] }}
- $all: MongoDB permite campos com arrays. Ex: { tags: [“NodeJS”, “MongoDB”] }. Com esse operador, você compara se seu campo multivalorado possui todos os valores de umarray específico. Ex: {tags: {$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:

```
> 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:

```
> 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.

2.6 UPDATE Avançado

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:

```
> 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, independentemente 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):

```
> 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:

```
> 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

2.7 DELETE Avançado

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:

```
> db.customers.deleteOne({nome: "Luiz"})
```

Vai excluir o primeiro cliente cujo nome seja igual a “Luiz”.

Obviamente existem coisas muito mais avançadas do que esse material sobre o MongoDB. Lhe encorajo a dar uma olhada no site oficial do banco de

dados(https://www.mongodb.com/2) 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.

Referências Bibliográficas

BRADSHAW, Shannon. **MongoDB: The Definitive Guide: Powerful and Scalable Data Storage**. O'Reilly Media; 3ª edição, 2019

Disponível em: <https://www.mongodb.com/2>. Acesso em: 01 jan. 2021

Disponível em: <http://db4beginners.com/categoria/database/nosql/mongodb/>. Acesso em: 04 jan. 2021

Disponível em: < http://www.cienciaedados.com/nosql-database/>. Acesso em: 04 jan. 2021