Nosql resumo

Resumos para análise de dados

Alura

NoSQL

NoSQL

Diferente dos bancos de dados relacionais, os bancos **de dados NoSQL** têm a sua estrutura para armazenamento dos dados como também a sua linguagem de consulta. Devido a isso, surgiram vários tipos de bancos de dados e cada um tem a sua forma de trabalhar. Geralmente, eles são rotulados de acordo com a forma como os dados são armazenados, então vamos entender um pouquinho mais sobre estes tipos de banco de dados existentes a seguir:

**Grafos:** Em um banco orientado a grafos, não existem tabelas, documentos ou qualquer outra estrutura que seja comparável a uma tabela. Neste tipo de banco, tudo são nós (vértices) e relacionamentos (arestas). O mais popular deles é o **Neo4j**.

**Chave / valor:** Todos os registros fazem parte da mesma coleção de elementos. Fazendo uma comparação com os bancos de dados relacionais, é como se o banco de dados inteiro fosse uma única tabela que possui apenas duas colunas: uma chave, e uma outra com valor. Um exemplo desse tipo é o **Redis**.

**Colunar:** A estrutura principal dos bancos colunares é basicamente uma grande tabela. Cada registro pode, ter quantas e quais colunas precisar (schemaless). Os bancos de dados colunares são os que mais se assemelham aos bancos de dados relacionais por terem uma "tabela" na sua estrutura, mesmo que, na verdade, sejam muito diferentes, já que os dados estão orientados a colunas internamente. Um exemplo desse tipo é o **Cassandra**.

**Documento:** O banco de dados de estilo de documento usa o que equivale a um objeto como sua menor unidade lógica. E uma coleção pode conter vários documentos. Por não haver um esquema fixo, cada documento pode ter um número e um tipo de campo diferente, o que contribui para a flexibilidade. Um exemplo desse tipo é **MongoDB**.

O MongoDB será o foco dos nossos estudos, ele atualmente já é utilizado por diversas empresas para armazenar e trabalhar com os dados, um exemplo disso é a EA Sports FIFA que é a franquia de videogame esportivo mais vendida do mundo. Para atender milhões de jogadores, o estúdio de desenvolvimento Spearhead da EA selecionou o MongoDB para armazenar dados do usuário e estado do jogo.

Sumário

[MongoDB 3](#_Toc186720390)

[Documentos 5](#_Toc186720391)

[Realizando Consultas 6](#_Toc186720392)

[Atualizando e removendo dados 8](#_Toc186720393)

[BSON 9](#_Toc186720394)

[MongoDB vs Cassandra: qual escolher? 10](#_Toc186720395)

[Esquema 12](#_Toc186720396)

[Validação de esquemas 13](#_Toc186720397)

[Criando validação de esquemas 14](#_Toc186720398)

[Modificando regras de validação 15](#_Toc186720399)

[Documentos Inválidos 17](#_Toc186720400)

[Modelagem de dados 19](#_Toc186720401)

[Relacionamentos 22](#_Toc186720402)

[Conhecendo o mongoImport 24](#_Toc186720403)

[Estágios de agregação 27](#_Toc186720404)

[Conteúdo extra 30](#_Toc186720405)

# MongoDB

O MongoDB é considerado o banco de dados NoSQL mais utilizado atualmente, os engenheiros de dados caracterizam ele como o 5 banco de dados mais utilizado no mundo inteiro. Ele utiliza o formato de documentos que é salvo dentro de coleções.

Alguns benefícios são:

* Documento: Reestruturação de dados.
* Replicação: Cópias de banco de dados duplicadas em mais 2 servidores.
* Cluster: Garante que possa manusear diversos dados.

MongoDB é um projeto open source com distribuição gratuita para Linux, Mac e Windows. Ele está disponível em duas edições de servidor: Community e Enterprise.

A versão [**Community edition**](https://www.mongodb.com/) é a edição gratuita do MongoDB disponível para fins de avaliação e desenvolvimento. Com a versão **Community edition**, pode-se criar diversos bancos de dados e coleções, além de tudo, trabalhar com diversos documentos e realizar diversas análises dos dados.

Já a versão [**Enterprise edition**](https://www.mongodb.com/products/mongodb-enterprise-advanced?tck=docs_server) é a edição comercial do MongoDB. Ela inclui recursos adicionais não disponibilizados na versão gratuita, como recursos avançados de segurança, assistência especializada, ferramentas poderosas, entre outros.

Além das duas versões de servidores, o MongoDB também disponibiliza a sua versão hospedada na nuvem, o [**MongoDB Atlas**](https://www.mongodb.com/atlas/database?tck=docs_server). Você pode experimentar o **Atlas** gratuitamente com um cluster de 512 MB disponibilizado pela MongoDB.

Ao ter instalado o MongoDB como um serviço você precisa fazer a conexão no prompt, então abra o prompt e digite “mongosh” para conectar o serviço ao shell.

Para ver os bancos de dados existentes é preciso executar o comando “show databases”.

Para criar um novo banco é necessário executar o comando “use nome\_banco”.

Por padrão, o MongoDB já vem com três bancos de dados criados, o Admin, config e local. Além desses, nós temos a liberdade de criar novos bancos de dados. Porém, precisamos seguir algumas restrições:

* **Diferenciação de maiúsculas e minúsculas do nome do banco de dados**

Os nomes de banco de dados diferenciam maiúsculas de minúsculas no MongoDB. Por exemplo: se o banco de dados AluraDB já existir, o MongoDB retornará um erro se você tentar criar um banco de dados chamado AluraDB.

* **Restrições sobre nomes de banco de dados para Windows**

Para implantações do MongoDB em sistemas operacionais Windows, os nomes de banco de dados não podem conter nenhum dos seguintes caracteres:

/. "$\*<>:|?

* **Restrições sobre nomes de banco de dados para sistemas Unix e Linux**

Para implantações do MongoDB em sistemas operacionais Unix e Linux, os nomes de banco de dados não podem conter nenhum dos seguintes caracteres:

/. "$

Além disso, os nomes de banco de dados não podem conter o caractere nulo.

* **Comprimento dos nomes de banco de dados**

Os nomes de banco de dados não podem estar vazios e devem ter menos de 64 caracteres.

Só usando o “use” criará a base de dados mas não aparecerá no MongoDB porque para ser um banco de dados precisa de coleções então criaremos uma no prompt digitando db.createCollection(“coleção”) , após ter feito isso, atualize a sua página do MongoDB e provavelmente estará lá a sua base de dados.

Assim como para criar um banco de dados, também existem restrições para se criar uma coleção aqui no MongoDB, que são:

* Os nomes das coleções devem começar com um sublinhado ou um caractere de letra.
* Não podem:
  + Conter o $.
  + Ser uma string vazia (por exemplo "",).
  + Conter o caractere nulo.
  + Começar com o system.prefixo. (Reservado para uso interno).

O MongoDB possui botões para excluir uma base de dados, e para excluir uma coleção pode digitar no prompt db.coleção.drop(), troque coleção pelo nome da sua coleção. Mas caso queira remover o base de dados pelo terminal, primeiro se conecte a database que quer excluir e depois execute db.dropDatabase() .

## Documentos

O MongoDB armazena os seus registros em documentos BSON, que é uma representação binária de documentos JSON, o benefício é que o BSON trabalha com mais tipos de dados. Da mesma forma que o JSON o BSON é composto por chave e valor, então o seu formato é parecido com o de um JSON. Um exemplo seria:

{

name: “Eduarda”,

age: 26,

status: “A”,

groups: [“news”, “sports”]

}

O MongoDB armazena registros de dados como documentos BSON, que é uma representação binária de documentos JSON.

O valor de um campo em um documento pode ser qualquer um dos tipos de dados BSON, incluindo outros documentos, matrizes e matrizes de documentos, então vamos conhecer alguns desses tipos de dados?

**NULL:** armazena valores nulos;

**Boolean:** pode armazenar valores true ou falso;

**Number:** número com sinal que pode ter uma notação com E exponencial;

**Inteiro:** pode armazenar o tipo de dados inteiro em duas formas, inteiro assinado de 32 bits e inteiro assinado de 64 bits;

**String:** uma sequência de um ou mais caracteres Unicode;

**Object:** um array não ordenado com itens do tipo chave/valor, também conhecidos como documentos aninhados;

**Array:** armazena uma lista ordenada de qualquer tipo, criada usando colchetes e com cada elemento separado por vírgulas;

**ObjectId:** identificador único de um registro do MongoDB;

**Date():** retorna a data atual em formato de string; e

**New Date()** e **ISODate():** retornam um objeto de data.

Esses são apenas alguns tipos de dados que podemos trabalhar no MongoDB, a título de complemento, você pode acessar também a documentação do MongoDB sobre [**BSON types**](https://www.mongodb.com/docs/manual/reference/bson-types/).

Assim, como para criar um banco de dados e coleções, também existem restrições ao se criar um documento:

* O nome do campo \_id é reservado para uso como chave primária. Seu valor deve ser único na coleção, é imutável e pode ser de qualquer tipo que não seja um array.
* Os nomes dos campos não podem conter o caractere **NULL**.

Os documentos BSON, também possuem restrições de tamanho:

* O tamanho máximo de um documento BSON é 16 megabytes.

Para acrescentar documentos pelo prompt temos 2 funções:

* InsertOne = adiciona um documento por vez.
* InsertMany = Adiciona vários documentos.

Para adicionar pelo prompt é preciso especificar a qual coleção quer adicionar, então a estrutura seria db.Coleção.InsertOne( { documento: formato BSON })

## Realizando Consultas

Temos 3 formas de visualizar os dados dentro do MongoDB, o modo de campo, o modo de BSON e o modo de tabela. Clicando em “FIND” você terá que escolher a forma como deseja pesquisar, a pesquisa que escolher já tem a estrutura adequada que será necessária para fazer a sua pesquisa, normalmente abrindo as chaves e passando em formato BSON, por exemplo:

{“ano de lançamento”: 2020}.

**FILTER**: utilizado para especificar qual será a condição que os documentos devem atender para serem retornados na busca.

**PROJECT**: utilizado para especificar quais campos serão ou não retornados na consulta.

* Ao Informar o nome do campo e informar 0, todos os campos, exceto os campos especificados no campo **project**, são retornados. Se o campo receber o valor de 1, ele será retornado na consulta. O campo \_id é retornado por padrão, a menos que este seja especificado no campo **project** e definido como 0.

**SORT**: especifica a ordem de classificação dos documentos retornados.

* Para especificar a ordem crescente de um campo, defina o campo como 1.
* Para especificar a ordem decrescente de um campo, defina o campo como -1.

**MAX TIME MS**: define o limite de tempo cumulativo em milissegundos para processar as operações da barra de consulta. Se o limite de tempo for atingido antes da conclusão da operação, o Compass interrompe a operação.

**COLLATION**: utilizado para especificar regras específicas do idioma para comparação de textos, como regras para letras maiúsculas ou minúsculas, acentos, entre outros.

**SKIP**: específica quantos documentos devem ser ignorados antes de retornar o conjunto de resultados.

**LIMIT**: especifica o número máximo de documentos a serem retornados.

Para pesquisar no prompt o método é db.coleção.find({condição de pesquisa}), aqui um exemplo:

db.users.**find**( ← collection

{ age: { $gt: 18 } }, ← query criteria

{ name: 1, address: 1 } ← projection

).**limit**(5) ← cursor modifier

Para ordenar a sua pesquisa poderá usar o modificador sort, um eemplo poderá ser:

db.users.find().sort({“age”: 1})

O número 1 significa que deverá ser ordenado pela idade de forma crescente, caso fosse decrescente poderia passar o -1, em seu lugar.

## Atualizando e removendo dados

Ao encontrar o documento que deseja editar, você pode editá-lo na própria página do MongoDB clicando em um ícone de lápis que aparecerá ao lado direito do documento.

No MongoDB Compass, não temos a opção de executar comandos para realizar atualização de documentos de uma coleção. Porém, temos alguns botões que podemos utilizar para editar e fazer outras ações com os nossos dados, então vamos conhecer um pouquinho sobre esses recursos.

Para alterar informações de um documento, utilizamos o botão de editar documento, ao clicar neste botão, podemos alterar a descrição de um campo, seu valor e tipo dados.

Também temos a opção de copiar os dados de um documento já inserido na nossa coleção. Para isso, utilizamos o botão de copiar documentos, se colocarmos a informação, teremos todos os dados do documento copiado.

E para finalizar temos a opção de clonar um documento, ao clicar nesse botão, a janela de inserir dados ficará disponível com os dados do documento clonado, onde podemos alterar a informações dos campos e inseri-lo novamente.

Podemos usar no mongosh para atualizar igual ao método para inserir documentos os métodos:

* updateOne = atualiza um documento de cada vez.
* updateMany = atualiza vários documentos de uma vez.

O formato desses métodos seria assim:

db.series.updateOne({“Serie”: “Grimm”}, {$set: {“Classificação”: “16+” }})

Sendo o que está dentro de updateOne a condição do documento que deseja atualizar.

Além dos métodos **UpdateOne** e o **UpdateMany**, o MongoDB também disponibiliza o método **ReplaceOne**, que é utilizado para substituir um único documento na coleção com base no filtro.

O **ReplaceOne** possui a seguinte estrutura:

db.collection.replaceOne(

<filter>,

<replacement> )

**Filter**: especificamos a condição para a atualização.

**Replacement**: especificamos o documento que será substituído.

Assim como para atualizar o excluir também tem um ícone no documento, normalmente aparece como uma lixeira. E para excluir pelo mongosh é semelhante aos métodos de update, tendo os métodos:

* DeleteOne = exclui um documento por vez.
* DeleteMany = exclui vários documentos de uma vez.

Mas a sua estrutura é mais simples que os demais:

db.users.deleteMany( ← collection

{ status: "reject" } ← delete filter

)

Cuidado: ao passar sem filtro ele pode excluir todos os dados.

## BSON

O Json é o JavaScript Object Notation ele é um formato de arquivos usado para troca de dados.  
  
{

"nome": "João",

"idade": 30,

"email": "joao@example.com",

"enderecos": [

{

"tipo": "casa",

"rua": "Rua das Flores",

"numero": 123,

"cidade": "São Paulo",

"estado": "SP"

},

{

"tipo": "trabalho",

"rua": "Avenida Paulista",

"numero": 456,

"cidade": "São Paulo",

"estado": "SP"

}

],

"hobbies": ["futebol", "leitura", "cinema"]

}

Esse é um exemplo de documento JSON, porém o MongoDB não utiliza este padrão, mas sim o BSON, que é uma versão superset. A diferença é que o BSON possui tipos adicionais com relação ao JSON, como por exemplo um tipo DATE, e é mais fácil para que o MongoDB faça uma varredura (Processo para que ele encontre o que procura).

## MongoDB vs Cassandra: qual escolher?

Os bancos de dados NoSQL não tem um esquema fixo, como tabelas, colunas ou linhas. Assim, as empresas podem trabalhar e armazenar os seus dados não estruturados sem grandes problemas, já que existem vários tipos de bancos de dados NoSQL, que trabalham de formas diferentes. Desse modo, as empresas têm uma variedade de opções, podendo escolher o que melhor se adequa a sua necessidade.

Devido a sua estrutura, o MongoDB disponibiliza um grande recurso, que é a **Flexibilidade** em relação ao armazenamento dos dados, pois como não existe uma estrutura pré-definida, cada documento de uma coleção pode ter quantidades e tipos de campos diferentes. Garante também uma **Alta disponibilidade** através dos conjuntos de réplicas, que é um recurso de replicação dos dados, ou seja, o MongoDB permite que haja cópias dos bancos de dados duplicados em mais servidores, assim, garantindo **redundância e disponibilidade dos dados**. Com esse recurso de replicar os seus dados o MongoDB garante uma **rápida leitura** dos seus dados, já que os(as) clientes podem realizar consultas no banco de dados principal e nas réplicas.

Mas, estas cópias dos bancos de dados, não podem ser utilizadas para as gravações dos dados devido a sua arquitetura escravo - mestre, então acaba se tornando uma pequena desvantagem para quem precisa de um rápido processo de gravação, pois existe um **limite na capacidade de gravação**, já que só podemos realizar a gravação dos dados no servidor primário e esses dados serão replicados para os servidores secundários.

Um ponto que também precisamos levar em consideração é a linguagem de consulta utilizada. Para realizar essas consultas o MongoDB desenvolveu uma linguagem de consulta própria, pois, ao contrário do SQL, não existe um padrão a ser seguido no NoSQL, então cada banco de dados pode criar e utilizar a sua linguagem, sem seguir um padrão definido.

á o Cassandra é um banco de dados de código aberto que utiliza o modelo de colunas, e o seu armazenamento é feito em forma de tabela, mas diferente da modelagem de tabela utilizada nos bancos de dados relacionais, a modelagem de dados do Cassandra é orientada por consulta.

Segundo a [documentação](https://cassandra.apache.org/doc/latest/cassandra/data_modeling/intro.html), no Cassandra não existe o conceito de relacionamento, ou seja, não existem chaves estrangeiras ou integridade relacional. Ao contrário dos bancos de dados relacionais que normalizam os dados, o Cassandra realiza a desnormalização dos seus dados que podem ser duplicados entre as várias tabelas e a sua estrutura é dividida da seguinte forma:

* **Keyspace:** armazena os conjuntos de tabelas.
* **Tabela:** armazena um conjunto de linhas.
* **Partição:** parte obrigatória da chave primária que identifica o nó em um cluster onde a linha está armazenada.
* **Linha:** armazena uma coleção de colunas identificadas por uma chave primária.
* **Coluna:** valor único com um tipo que pertence a uma linha.

Mais sobre o MongoDB em: [MongoDB validando documentos - Slides.pptx](https://cdn3.gnarususercontent.com.br/2710-mongodb/01/MongoDB+validando+documentos+-+Slides.pdf)

## Esquema

Uma das grandes dificuldades para quem está começando a trabalhar com banco de dados não relacional é justamente entender como modelar os dados. E algumas vezes se perguntam, por que devo modelar dados em um NoSQL? E temos 3 grandes motivos para isso:

1. Evitar Problemas
2. Entender as aplicações
3. Como utilizar o MongoDB

Já que ainda não tocamos no assunto vamos falar sobre quando escolher NoSQL ou banco de dados relacionais.

BD Relacionais:

* Relacionamento entre diversas entidades
* Usar chaves estrangeiras “Joins”

NoSQL:

* Grande quantidade de dados desestruturados
* Esquema desconhecido

Não existem regras para realizar a modelagem de dados em bancos de dados NoSQL, e cada banco de dados pode trabalhar de formas diferentes com seus dados.

Algumas das aplicações levantadas pelo time do MongoDB são: Catálogos de vendas e serviços internet das coisas, Mobile, entre outras. Cada aplicação pode utilizar um ou mais tipos de padrões de modelagem, como, por exemplo:

* Aproximação: Utilizado ao se realizar grandes quantidades de cálculos
* Atributos: Utilizado ao se trabalhar com grandes documentos que possuem campos semelhantes.
* Balde: Utilizado para gerenciar dados de streaming.
* Versão de documentos: Utilizado quando há a necessidade de armazenar versões anteriores de documentos.
* Versão do esquema: Utilizado quando há a necessidade de armazenar versões anteriores e atuais de documentos existam.
* Árvore: Utilizado quando os dados estão armazenados em uma estrutura hierárquica.

Porém, uma observação fundamental é que nós não precisamos seguir estes padrões de Modelagem, uma vez que podemos criar o nosso próprio modelo de acordo com as necessidades da nossa aplicação.

O esquema do MongoDB é flexível, ou seja, nem todos os dados precisam ter as mesmas informações ou os mesmo campos.

## Validação de esquemas

Ao criar as nossas coleções no MongoDB não precisamos definir a estrutura dos nossos documentos. Porém, ainda é necessário realizar a modelagem dos nossos dados para evitar possíveis problemas e conhecer como será a estrutura do banco de dados.

Um dos recursos disponibilizados pelo MongoDB para realizarmos a modelagem dos nossos dados é a validação de esquema.

E você pode se perguntar, o que é a validação de esquema?

A validação de esquema são regras que criamos para que os nossos dados sejam validados. Então, após definir o esquema de uma coleção, sempre que houver a inserção ou atualização de documentos, as regras serão aplicadas e os documentos validados. Caso o documento seja invalido e não esteja de acordo com o esquema criado, ele não será inserido na coleção.

Ao criar o esquema das nossas coleções, podemos definir regras de validação utilizando o operador de avaliação **$jsonSchema**.

Podemos especificar qual o tipo de dado o nosso campo irá receber utilizando a palavra chave bsonType , qual o valor máximo e mínimo utilizando maxLength e minLength, definir quais possíveis valores um campo pode aceitar utilizando Enum, entre outras palavras chave que podemos utilizar.

Mas, nós também podemos utilizar os **operadores de consulta** para validar os nossos dados. Nesse sentido, temos os operadores lógicos, $and, $or, $nor e $not, além de termos os operadores de comparação lt, $lte, $in, entre outros.

Porém, nem todos os operadores são permitidos, os **operadores de avaliação** $expr, $text e $where e os **operadores geoespaciais** $near e $nearSphere, não podem ser utilizados.

Você pode conhecer mais a respeito deste assunto acessando a [**documentação do MongoDB**](https://www.mongodb.com/docs/manual/core/schema-validation/specify-query-expression-rules/) sobre especificar validação com operadores de consulta.

O **NoSQLBooster** é uma ferramenta de desenvolvimento para trabalhar com bancos de dados MongoDB. Ele oferece um conjunto de recursos que tornam mais fácil interagir com o MongoDB, seja para consultas, gerenciamento ou desenvolvimento de aplicações.

## Criando validação de esquemas

Antes de criar as nossas coleções, é necessário criar o nosso banco de dados.

Temos 3 opções para criar banco de dados utilizando o **NoSQLBooster**:

**1 - Através do comando:** use nome\_banco\_de\_dados

Após executar este comando, automaticamente você estará conectado ao banco de dados, porém, você precisa inserir dados para que ele seja fisicamente criado.

**2 - Utilizando o assistente:**

Acessando o menu lateral **Open Connections**, clicando com o botão direito do mouse sobre a conexão onde será criado o banco de dados e selecionando a opção **Create Databases**.

Diferente de criar o banco de dados utilizando o comando, ao utilizar o assistente, o banco de dados é criado fisicamente de forma automática, pois é necessário armazenar as informações das contas das pessoas usuárias.

**3 - Utilizando o atalho:**

alt + N, o banco de dados é criado fisicamente de forma automática, pois assim como ao criar o banco de dados utilizando o assistente, é necessário armazenar as informações das contas das pessoas usuárias.

Uma forma de regra de validação que posso dar como exemplo foi esta:

db.**createCollection**("Produtos",

{

validator:

{

$jsonSchema:

{

bsonType: "object",

required:[“Identificador", "Descrição","Quantidade","Preço"],

properties:

{

Identificador:

{bsonType: "int"},

Descrição:

{bsonType: "string"},

Quantidade:

{bsonType: "int"},

Preço:

{bsonType: "double"}

}

}

}

})

## Modificando regras de validação

Para que possa verificar a sua regra de validação você pode utilizar o comando o db.getCollectionInfos({name: “nome\_coleção”}), isso dentro do NoSQLBooster.

Para criar modificações em regras existentes você deve recopia-las e modificar a seu gosto como o exemplo abaixo:

db.**runCommand**({collMod:"contas",

validator:{

$jsonSchema:{

bsonType:"object",

"additionalProperties": **false**,

required:["\_id","Numero\_Conta", "Tipo", "CPF", "Valor"],

properties:{

\_id:{

bsonType: "objectId",

description: "informe corretamente o endereço do cliente"

},

Numero\_Conta:{

bsonType: "string",

description: "informe corretamente o número da conta"

},

Tipo:{

bsonType: "string",

enum:["Conta corrente", "Conta poupança", "Conta salário"],

description: "informe corretamente o tipo da conta"

},

CPF:{

bsonType: "string",

minLength:14,

maxLength:14,

description: "informe corretamente o cpf do cliente na conta"

},

Valor:{

bsonType: "double",

description: "informe corretamente o valor da conta"

}

}

}

}

})

Existem dois níveis de validação de documentos são esses :

* Strict ->
* **Comportamento**: Quando a validação está configurada como **strict**, o MongoDB rejeita qualquer operação de inserção ou atualização que **não atenda completamente** às regras de validação definidas. Ou seja, se o documento não seguir todas as regras da **schema** de validação, ele será **completamente rejeitado**.
* **Uso**: Esse nível é adequado quando você quer garantir que **todos os documentos** de uma coleção sigam rigorosamente o modelo ou esquema de dados que você definiu.
* Moderate ->
* **Comportamento**: Com a validação **moderada**, o MongoDB permite que a operação de inserção ou atualização seja executada mesmo que o documento não atenda completamente às regras de validação, mas a operação será **parcialmente permitida**. Ou seja, os documentos podem ser parcialmente aceitos, mas o MongoDB ainda tenta aplicar as regras o máximo possível.
* **Uso**: Esse nível é útil quando você deseja um certo grau de flexibilidade nas operações de dados, permitindo que documentos parcialmente válidos sejam aceitos, desde que atendam a algumas regras essenciais.

## Documentos Inválidos

Para localizar e modificar documentos inválidos armazenados nas coleções, podemos utilizar o operador $jsonSchema e realizar a busca com base em um esquema.

O primeiro passo é definir um objeto de esquema e armazená-lo em uma variável.

Para criar uma variável, você pode utilizar o comando **let**:

let nome\_variavel = $jsonSchema

Então, vamos criar uma variável que armazenará o esquema da coleção de clientes:

let clientes =

{ $jsonSchema:

{ required: [ "nome", "cpf", "status\_civil", "data\_nascimento", "endereco" ],

properties:

{ nome: {

bsonType: "string",

maxLength: 150,

description: "O nome do cliente deve ser informado corretamente"

}, cpf: {

bsonType: "string",

minLength:14,

maxLength: 14,

description: "O CPF do cliente deve ser informado"

}, status\_civil: {

enum: [ "Solteiro(a)", "Casado(a)", "Separado(a)", "Divorciado(a)", "Viúvo(a)" ],

description: "Informe um dos valores presentes em ENUM"

}, data\_nascimento: {

bsonType: "string",

description: "A data de nascimento do cliente deve ser informado corretamente"

}, endereco: {

bsonType: "string",

description: "O endereço do cliente deve ser informado corretamente"

} } } }

**OBS.:** Para realizar a criação de uma variável, é necessário utilizar a linha de comando.

Após executar o comando e criar a variável, você pode utilizá-la como um filtro para buscar, modificar e remover documentos das coleções.

Buscando documentos que correspondem ao esquema de validação:

db.clientes.find(contas)

Buscando documentos que não correspondem ao esquema de validação.

db.clientes.find( { $nor: [ contas ] } )

## Modelagem de dados

Existem diversas formas de relacionamento mesmo em um banco de dados NoSQL, mas antes de entender mais sobre isso, vamos aprender um pouco mais sobre como é importante a modelagem de dados.

Aprender a **modelar os dados** pode ser uma das maiores dificuldades para quem vai começar a trabalhar com banco de dados, principalmente quando são do tipo NoSQL, já que não existem regras a serem seguidas para a padronização dos dados.

Para modelar um banco de dados relacional, temos os diagramas, modelos de dados, normalização de dados, formas normais, entre outros conceitos e padrões que podemos aplicar para modelar os nossos dados.

Ao se trabalhar em um projeto para modelar um banco de dados não relacional, nenhum desses conceitos ou padronizações se aplicam. Para cada projeto de banco de dados, podemos modelar os nossos dados de formas diferentes.

Por este motivo, é muito importante entendermos as necessidades da nossa aplicação e como ela utilizará o banco de dados, para que o projeto seja desenvolvido e atenda a todos os objetivos da aplicação.

Mas como o MongoDB trata as suas transações ACID?

O **MongoDB** oferece suporte a transações ACID (Atomicidade, Consistência, Isolamento e Durabilidade) desde a versão 4.0, lançada em 2018. Essa funcionalidade é particularmente importante para garantir que operações complexas envolvendo múltiplos documentos ou coleções sejam tratadas de forma confiável, o que antes não era possível com operações atômicas limitadas a um único documento.

**1. Atomicidade**

* **O que é?** Garante que todas as operações dentro de uma transação sejam concluídas com sucesso ou, caso contrário, que nenhuma operação seja aplicada.
* **Como funciona no MongoDB?** Quando você inicia uma transação em MongoDB, ele garante que todas as modificações feitas dentro dessa transação sejam aplicadas de forma atômica. Se a transação falhar, todas as mudanças feitas dentro da transação são revertidas (rollback).
* **Exemplo:** Se você estiver fazendo várias atualizações em diferentes documentos ou coleções e a transação falhar no meio, as mudanças realizadas até o ponto de falha são revertidas, e o banco de dados permanece em seu estado anterior.

**2. Consistência**

* **O que é?** Garante que o banco de dados esteja em um estado válido após a conclusão da transação. Em outras palavras, a transação leva o banco de dados de um estado consistente a outro estado consistente.
* **Como funciona no MongoDB?** Durante a transação, o MongoDB garante que as modificações sejam realizadas de acordo com as regras de integridade dos dados, incluindo qualquer validação de dados definida no banco. Se a transação violar uma restrição de consistência, ela falhará e será revertida.

**3. Isolamento**

* **O que é?** Garante que as transações sejam isoladas umas das outras, de forma que uma transação não possa interferir nos dados de outra transação simultânea.
* **Como funciona no MongoDB?** O MongoDB oferece o nível de isolamento **"Read Committed"** por padrão, o que significa que dentro de uma transação, você só pode ler os dados que foram confirmados (ou seja, não inclui leituras de dados modificados em transações não confirmadas).

O isolamento é implementado de forma que as leituras e gravações feitas por uma transação são visíveis apenas para ela até o momento em que ela for concluída com sucesso.

**4. Durabilidade**

* **O que é?** Garante que, uma vez que a transação é confirmada, suas alterações são permanentes, mesmo que ocorra uma falha no sistema.
* **Como funciona no MongoDB?** O MongoDB usa um mecanismo de **Write Ahead Log (WAL)** para garantir que todas as transações sejam gravadas de forma segura. Quando a transação é confirmada (commit), as modificações são persistidas no disco de forma durável e resistente a falhas de hardware ou software.

**Tipos de Transações no MongoDB**

* **Transações Simples:** Realizadas em uma única operação de banco de dados, como inserir, atualizar ou excluir documentos. Embora suportem ACID dentro de um único documento, quando você começa a trabalhar com múltiplos documentos ou coleções, a transação deve ser explícita.
* **Transações Multi-Documentos:** Permitidas desde a versão 4.0, essas transações podem envolver múltiplos documentos, coleções e até bancos de dados diferentes. Para esse tipo de transação, é necessário usar a interface de transação explícita para garantir que os princípios ACID sejam aplicados.

**Como iniciar e usar transações no MongoDB?**

Para usar transações, você deve iniciar uma sessão de banco de dados e iniciar a transação dentro dessa sessão. Aqui está um exemplo básico de como isso pode ser feito em **MongoDB com Node.js**:

const session = client.startSession();

session.startTransaction();

try {

const collection1 = client.db("meuDB").collection("colecao1");

const collection2 = client.db("meuDB").collection("colecao2");

collection1.insertOne({ nome: "item1" }, { session });

collection2.insertOne({ nome: "item2" }, { session });

await session.commitTransaction();

console.log("Transação confirmada!");

} catch (error) {

await session.abortTransaction();

console.log("Transação falhou e foi revertida", error);

} finally {

session.endSession();

}

No código acima:

* Uma **sessão** é iniciada.
* A transação começa com startTransaction().
* Operações são realizadas em coleções diferentes.
* Se tudo ocorrer bem, a transação é confirmada com commitTransaction(). Se algum erro ocorrer, a transação é revertida com abortTransaction().

**Considerações e Limitações**

* **Performance**: Embora as transações tragam robustez ao sistema, elas podem impactar a performance, especialmente em ambientes de alta carga, uma vez que o MongoDB tem que manter o controle rigoroso das operações e garantir o isolamento e a consistência.
* **Escalabilidade**: O uso de transações distribuídas (envolvendo múltiplos shards) pode ser mais complexo e exigir configurações adequadas.

A execução de alguns comandos, como a criação de variáveis, não pode ser executada diretamente no NoSQLBooster. Por este motivo, será necessário a utilização de uma nova ferramenta: o Visual Studio Code.

## Relacionamentos

Banco de dados relacionais são um conjunto de tabelas que se relacionam entre si. Eles utilizam a linguagem SQL para a manipulação dos seus dados e são gerenciados pelos SGBDs (Sistema Gerenciador de Banco de Dados).

Nos bancos de dados relacionais, os relacionamentos entre as tabelas são definidos com base nas **cardinalidades**. Esses relacionamentos são do tipo:

* UM PARA MUITOS;
* MUITOS PARA MUITOS;
* UM PARA UM;
* MUITOS PARA UM.

O MongoDB é um banco de dados NoSQL que também utiliza o conceito de relacionamento, porém de uma forma diferente. Os relacionamentos no MongoDB são entre os documentos e podemos classificá-los em três tipos:

* um-para-um com documentos incorporados;
* um-para-muitos com documentos incorporados; e
* um-para-muitos com referências de documentos.

Dessa forma, os relacionamentos também devem ser levados em consideração ao se modelar um projeto de banco de dados no MongoDB.

um-para-um com documentos incorporados = Um exemplo em aplicação bancária seria uma informação de endereço para uma informação de conta, ou seja, só temos um único valor, em cada documento incorporado.

O MongoDB possui um esquema flexível que pode facilitar a estruturação de um novo banco de dados.Também facilita possíveis mudanças na estrutura das coleções já existentes, uma vez que não é necessário realizar um comando de modificação para alterar a estrutura.

Ao passar um novo campo em um comando de insert, os novos documentos passam a ter uma nova estrutura. Porém, muitas vezes surge a necessidade de se ter controle das versões existentes de um esquema de uma coleção dentro do banco de dados. Alguns desses motivos, são:

* A aplicação não pode ser parada para a atualização de todos os documentos do banco de dados para uma nova versão do esquema.
* A atualização pode demorar um período muito longo para ser finalizada, como dias ou semanas.
* Não existir a real necessidade de atualizar os documentos já existentes para a nova versão do esquema.

Dessa forma, ao se utilizar o padrão de controle de versão de esquema, é possível manter um controle das versões diferentes de um mesmo esquema em coleções dentro de um banco de dados no MongoDB.

No MongoDB as operações de gravação são atômicas a nível de um único documento, ou seja, caso uma operação de atualização de um documento falhe, ela não interrompe outras operações.

Ao se modelar um novo projeto de banco de dados, as informações que têm dependências a nível de alteração, mais precisamente que devem ser alteradas em conjunto, são mantidas em um mesmo documento. Isso pode garantir que os campos possam ser atualizados atômicamente.

Ao utilizar esta organização para as informações, estamos utilizando o **padrão de operações atômicas**.

O MongoDB disponibiliza três formas de estruturar os nossos dados utilizando a estrutura de dados incorporados, na qual os dados que se relacionam são armazenados em um único documento chamado de **estrutura de referências**. Nela, os dados que se relacionam são armazenados em coleções separadas e a terceira (e última) é a estrutura de árvore que armazena os dados hierárquicos.

Porém, nem sempre utilizar apenas uma das estruturas pode resolver o problema da aplicação.

Muitas vezes ao desenvolver o projeto, podemos identificar que os nossos dados precisam estar em coleções separadas. Mas, para obter as informações necessárias, precisaríamos realizar diversas consultas no banco de dados.

É possível, então, utilizar as duas estruturas de dados em um único banco de dados? A resposta é sim. Para resolver o problema, os dados mais acessados de uma coleção seriam duplicados para a coleção “Principal”, utilizando a estrutura de dados incorporados e a de referência. Porém, dessa forma nós estaríamos retornando também dados desnecessários, podendo afetar o desempenho do banco de dados.

Então, em vez de duplicar todas as informações de uma coleção para outra, copiamos apenas os campos que são acessados com mais frequência. Assim, não é necessário incorporar todas as informações ou incluir uma referência para uma junção das informações. Esta estrutura é conhecida como o **padrão de referência estendida**.

## Conhecendo o mongoImport

Podemos criar coleções no MongoDB de três maneiras diferentes! Primeiro, usando o comando db.createCollection() diretamente no shell do MongoDB. Depois, criando-as manualmente pela interface gráfica, clicando com o botão direito no banco de dados e selecionando "Create Collection". E por fim, a maneira mais prática que aprendemos: usando o mongoimport para importar dados de arquivos JSON, CSV ou TSV, criando as coleções e populando-as automaticamente!

Sobre a instalação do mongoimport, que faz parte das MongoDB Command Line Database Tools.

O mongoimport é uma ferramenta utilizada para realizar a importação de arquivos que possuem o formato JSON , CSV ou TSV. Estes arquivos podem ser criados pelo Mongoexport, que é a ferramenta de exportação do MongoDB ou por outra ferramenta de exportação.

**Instalando o Mongoimport**

1) Acesse o navegar e busque por [mongoimport](https://www.mongodb.com/try/download/bi-connector" \t "_blank).

2) Ao acessar o link, localize a opção **MongoDB Command Line Database Tools Download** ou **Download de ferramentas de banco de dados de linha de comando do MongoDB**, caso tenha feito a tradução da página.

3) Selecione a **versão**, a **plataforma** e o **tipo do arquivo**(pacote) e clique em **download**.

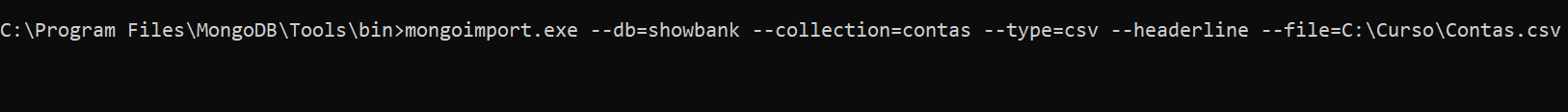
4) Com o download realizado, salve o arquivo na pasta que desejar. Em seu ambiente, descompacte o arquivo.

Pronto! Agora todas as ferramentas do mongodb, como o mongoimport e o mongoexport, estão disponíveis no seu ambiente. Aproveite este mergulho em MongoDB e desfrute conosco das inúmeras possibilidades deste software.

**Realizando a importação de arquivos**

1) Acesse a linha de comando e navegue até a pasta onde o arquivo do mongoimport foi salvo no seu ambiente.

2) Na linha de comando localize o executável do Mongoimport e execute o seguinte comando:



Para renomear uma coleção no MongoDB, podemos utilizar o método **renameCollection**, passando como parâmetro o novo nome da coleção.

Temos alguns outros grupos de operadores que podem ser utilizados junto ao método Find, como os:

* Operadores de consulta de avaliação;
* Operadores de consulta geoespacial;
* Operadores de consulta bit a bit.

Então, vamos conhecer melhor cada um deles e alguns exemplos de cada grupo:

**Operadores de consulta de avaliação:**

Esses operadores retornam dados em uma consulta baseada em avaliações de campos individuais ou de toda a coleção.

* $expr – Permite o uso de expressões de agregação dentro da linguagem de consulta.
* $jsonSchema – Valida os documentos em relação ao esquema JSON fornecido.
* $mod – Executa uma operação de módulo no valor de um campo.

**Operadores de consulta geoespacial:**

Os operadores geoespaciais permitem que seja realizado consultas baseadas em dados geográficos.

* $geoWithin: seleciona documentos dentro de um determinado polígono;
* $near: seleciona documentos baseados em pontos geoespaciais do mais próximo ao mais distante.

**Operadores de consulta bit a bit:**

Os operadores de consulta bit a bit são utilizados para realizar operações com base nas condições de posição de um bit.

* $bitsAllClear: seleciona documentos onde todos os bits especificados são 0.
* $bitsAllSet: seleciona documentos onde todos os bits especificados são 1.
* $bitsAnyClear: seleciona documentos onde pelo menos um dos bits especificados é 0.
* $bitsAnySet: seleciona documentos onde pelo menos um dos bits especificados é 1.

Atenção: Lembre-se que esses exemplos são apenas um ponto de partida e existem muitos outros operadores disponíveis para consulta na documentação. Para isso, acesse o [**MongoDB**](https://www.mongodb.com/docs/manual/reference/operator/query/) e conheça um pouco mais sobre esses operadores.

## Estágios de agregação

MapReduce é um modelo de programação e um método de processamento de dados distribuído. Usado para realizar consultas em grandes conjuntos de dados, como os encontrados em bancos de dados NoSQL no MongoDB. O MapReduce divide a consulta em tarefas menores que são executadas em paralelo em vários nós de processamento, o que torna a consulta mais rápida e escalável.

O processo do MapReduce consiste em duas fases:

* **Map:** Nesta fase a consulta é dividida em tarefas menores, cada uma delas sendo aplicada a um subconjunto dos dados.
* **Reduce:** Nesta fase os resultados parciais são combinados para produzir o resultado final da consulta.

No MongoDB, o MapReduce é usado para realizar consultas em coleções existentes. Como já estudamos anteriormente, cada coleção pode ter documentos com estruturas diferentes, ou seja, não há um esquema rígido que precise ser seguido ao inserir dados em uma coleção. Como resultado, as consultas em coleções podem ser complexas e o MapReduce é uma maneira eficaz de lidar com essas consultas.

Nessa perspectiva, o MapReduce no MongoDB é executado em três etapas. Observe a seguir as características de cada uma:

* **Map:** A primeira etapa é a fase de map, que divide a consulta em tarefas menores. Cada tarefa é executada em um nó do cluster e produz um conjunto de resultados parciais.
* **Reduce:** Na segunda etapa, a fase de reduce, os resultados parciais são combinados em um único resultado final. Isso é feito por meio de uma função de redução que agrega os resultados parciais.
* **Finalize:** Na terceira etapa, a fase de finalização, uma função opcional é aplicada ao resultado final para fazer uma limpeza ou processamento adicional.

Embora o MapReduce esteja obsoleto a partir da versão 5.0 do MongoDB, temos a possibilidade de usar no seu lugar a pipeline de agregação. Sendo esta um ou mais estágios que processam documentos, fornecendo melhor desempenho e uma maior usabilidade.

o count() pode ser utilizado de 3 formas:

1. como método find() como cursor de modificação;
2. como método;
3. com o método de agregação.

Em uma nova aba, vamos relembrar as 2 primeiras formas de utilizá-lo.

Faremos uma consulta na tabela de clientes e usaremos o cursor de modificação count() para buscar e contar a quantidade de documentos:

db.clientes.find().count()

db.clientes.count()

A diferença entre o uso do count() como modificador e como método, é que na primeira opção ele busca e conta, enquanto na segunda apenas conta e retorna a mensagem com o número exato de documentos existentes.

A terceira forma de usar o count() é junto do método aggregate(), da seguinte forma:

**db**.clientes.aggregate({})

Em seguida, passamos o operador $count: com um apelido para o campo criado durante a execução. Optemos por "contagem de clientes":

db.clientes.aggregate({$count:"contagem de clientes})

**Estágio Group**

Este estágio agrupa os documentos de acordo com uma chave de grupo. Então é necessário passar um id para que ele possa executar.

Faremos isso para nossa tabela de contas, e não mais de clientes:

db.contas.**aggregate**({$group:{\_id:"$tipo"}})

Com o operador $unionWith é possível realizar consultas complexas que recuperam dados de várias coleções em uma única consulta, além de retornar os resultados combinados. Isso é útil quando se deseja realizar uma análise mais ampla dos dados, pois nem sempre os dados são armazenados fisicamente em uma única coleção dentro do banco de dados.

Utilizamos a seguinte sintaxe para utilizar o operador $unionWith em uma consulta:

db.coleção.**aggregate**( [

{ $project: { \_id: 0 } },

{ $unionWith: {<coleção>} }

])

O método aggregate é uma poderosa ferramenta utilizada no MongoDB, pois através dela podemos realizar consultas complexas em dados armazenados em suas coleções. Esse método permite que sejam aplicados diversos operadores, incluindo operadores de comparação, lógicos e matriz, para filtrar, agrupar e analisar dados de uma coleção. Então, vamos conhecer um pouco mais sobre esses operadores.

* **Operadores de Comparação:**

Os operadores de comparação, como $gt (maior que), $lt (menor que) e $eq(igual a), $ne (diferente de), são usados para comparar valores em documentos de uma coleção e retornar resultados que atendem a determinadas condições. Esses operadores são usados com frequência em conjunto com o operador $match, que é responsável por filtrar documentos que atendem às condições especificadas.

* **Operadores Lógicos:**

Os operadores lógicos, como $and, $or e $not, são usados para combinar condições em consultas. O operador $and, por exemplo, permite que múltiplas condições sejam combinadas em uma única consulta. Já o operador $or permite que duas ou mais condições sejam combinadas em uma consulta, e o operador $not nega uma condição.

* **Operadores de Matriz:**

Os operadores de matriz, como $zip e $size, são usados para trabalhar com campos de matriz em documentos de uma coleção. O operador $zip, por exemplo, é usado para mesclar dois arrays juntos. Já o operador $size é usado para retornar o número de elementos existentes em uma matriz.

* **Expressões Trigonométricas:**

Os operadores de expressões trigonométricas, como $sin, $cos e $tan, são usados para executar cálculos trigonométricos em dados armazenados em uma coleção MongoDB. Esses operadores podem ser usados para calcular valores com base em informações baseadas em ângulos.

Vale ressaltar que esses são apenas alguns exemplos de grupo de operadores, que podem ser usados em conjunto com o método aggregate para realizar consultas complexas em dados de uma coleção no MongoDB.

## Conteúdo extra

Você pode utilizar SQL dentro do NoSQLBoooter com o MongoDB com o comando mdb.runSQLQuery(`código SQL`)

Para criar um novo usuário no MongoDB podemos usar o comando db.createUser(“nome”) e o tipo de autenticação padrão do MongoDB é o SCRAM e podemos autenticar um usuário usando o comando db.auth(<nome\_do\_usuario>,<senha\_do\_usuario>) em uma conexão do Mongo Shell.

No mundo da administração de banco de dados MongoDB, a realização de backups é uma prática fundamental para garantir a segurança e a integridade dos dados. Duas ferramentas amplamente utilizadas para esse propósito são o mongodump e o mongorestore.

Sabendo disso, como podemos executar o backup de uma coleção específica no MongoDB usando o utilitário "mongodump"?

mongodump --collection <nome\_da\_colecao> --db <nome\_do\_banco>