Tipos de Dados no MongoDB

O MongoDB, como um banco de dados NoSQL orientado a documentos, oferece uma ampla variedade de tipos de dados para modelar informações de forma flexível. Essa flexibilidade é uma das principais razões para sua popularidade.

Tipos de Dados Básicos:

- String: Utilizado para armazenar texto, como nomes, endereços, e-mails, etc.
- Number: Inclui tanto números inteiros (32 e 64 bits) quanto números de ponto flutuante (64 bits).
- Boolean: Representa valores lógicos, verdadeiro ou falso.
- Date: Armazena datas e horas em formato BSON.
- ObjectId: Um identificador único gerado automaticamente para cada documento.

Tipos de Dados Complexos:

- Array: Uma lista ordenada de valores, podendo conter elementos de diferentes tipos.
- Object: Um conjunto não ordenado de pares chave-valor, onde a chave é uma string e o valor pode ser qualquer tipo de dado.
- Binary Data: Permite armazenar dados binários, como imagens, arquivos, etc.

Exemplo de uma Coleção com Diversos Tipos de Dados:

```
db.produtos.insertMany([
    {
        _id: '123',
        nome: "Hambúrguer",
        preco: 15.99,
        ingredientes: ["pão", "carne", "queijo", "alface",
"tomate"],
        vegetariano: false,
        data_cadastro: new Date(),
        calorias: {
            total: 500,
```

```
            porcoes: 2
        }
            },
    {
        _id: '456',
        nome: "Salada",
        preco: 12.50,
        ingredientes: ["alface", "tomate", "pepino", "cenoura"],
        vegetariano: true,
        data_cadastro: new Date()
    }
]);
```

Entendendo a coleção:

- _id: Um ObjectId único para cada documento.
- nome: Uma string representando o nome do produto.
- preco: Um número (double) indicando o preço.
- ingredientes: Um array de strings com os ingredientes.
- vegetariano: Um booleano indicando se o produto é vegetariano.
- data_cadastro: Uma data representando quando o produto foi cadastrado.
- calorias: Um objeto com informações sobre as calorias.

Observações:

Flexibilidade: O MongoDB permite que você tenha documentos com estruturas diferentes dentro da mesma coleção.

Esquemas: Embora não seja estritamente necessário, definir um esquema pode ajudar a garantir a consistência dos dados e facilitar a consulta.

Caso deseje criar um schema junto com a criação da collection, o comando é:

```
db.createCollection("produtos", {
  validator: {
     $jsonSchema: {
     "bsonType": "object",
     "required": ["_id", "nome", "preco", "ingredientes",
"vegetariano", "data_cadastro", "calorias"],
     "properties": {
     "_id": {
```

```
            "bsonType": "string",
            "description": "O identificador único do produto (deve
ser uma string)."
      },
      "nome": {
            "bsonType": "string",
            "minLength": 3,
            "description": "O nome do produto (mínimo de 3
caracteres)."
      },
      "preco": {
            "bsonType": "double",
            "minimum": 0,
            "description": "O preço do produto (deve ser um número
positivo)."
      },
      "ingredientes": {
            "bsonType": "array",
            "items": {
            "bsonType": "string"
            },
            "minItems": 1,
            "description": "Lista de ingredientes (deve conter pelo
menos um item)."
      },
      "vegetariano": {
            "bsonType": "bool",
            "description": "Indica se o produto é vegetariano."
      },
      "data_cadastro": {
            "bsonType": "date",
            "description": "Data de cadastro do produto."
      },
      "calorias": {
```

```
            "bsonType": "object",
            "required": ["total", "porcoes"],
            "properties": {
            "total": {
            "bsonType": "int",
            "minimum": 0,
            "description": "Total de calorias do produto."
            },
            "porcoes": {
            "bsonType": "int",
            "minimum": 1,
            "description": "Número de porções do produto (mínimo 1)."
            }
            },
            "description": "Informações nutricionais do produto."
      }
      }
      }
  }
});
```

Como visualizar o description?

O campo **`description`** dentro do **JSON Schema** no MongoDB é usado apenas como uma **documentação interna** e não é retornado automaticamente ao consultar os dados da coleção. Ele serve para descrever a finalidade de cada campo e facilitar a manutenção do schema, especialmente para desenvolvedores que precisam entender a estrutura.

O `description` pode ser visualizado apenas quando se recupera o schema da collection com o comando:

```
db.getCollectionInfos({ name: "produtos" })
```

Aplicando validação em uma collection existente

Se a collection `produtos` já existir, podemos modificar sua configuração com o seguinte comando:

```
db.runCommand({ collMod: "produtos",
validator: { $jsonSchema: { … regras …}, validationLevel: "moderate"
});
```

Índices: Crie índices para melhorar o desempenho de consultas, especialmente em campos frequentemente utilizados em filtros.

Tipos Específicos: Além dos tipos básicos, o MongoDB suporta tipos mais específicos, como Geospatial, Timestamp, Regular Expression, etc., para cenários mais complexos.

Inserindo um produto inválido.

```
db.produtos.insertOne({codigo:2})
```

Observe que a única mensagem retornada pelo Mongodb foi: Document Failed Validation. Caso deseje visualizar mais detalhes do retorno, insira o comando de um estrutura try/catch:

```
try {
  db.produtos.insertOne ({codigo:  2});
} catch (err) {
    printjson(err);
}
```

insertOne() e insertMany() no MongoDB

insertOne()

Adiciona um único documento à coleção.

```
db.produtos.insertOne({ nome: "Notebook Gamer", preco: 5000, cor:
"Preto", caracteristicas: ["i9", "RTX 3080", "16GB RAM"], emEstoque:
true })
```

Este comando irá adicionar um novo produto à coleção "produtos" com os dados especificados.

insertMany()

Adiciona múltiplos documentos à coleção em uma única operação.

Exemplo:

```
db.produtos.insertMany([
{
      nome: "Smartphone X Pro",
      preco: 3500,
      cor: "Branco",
      caracteristicas: ["5G", "Câmera quádrupla", "Bateria de longa
duração"],
      emEstoque: true },
{
      nome: "Tablet S",
      preco: 1500,
      cor: "Cinza",
      caracteristicas: ["Tela grande", "Processador rápido", "Leve e
portátil"],
      emEstoque: false } ])
```

Inserindo Documentos com IDs Personalizados

Por padrão, o MongoDB gera um ObjectId único para cada novo documento. No entanto, você pode especificar seu próprio ID usando o campo "_id".

Exemplo:

```
db.produtos.insertOne(
{
      _id: "produto123",
      nome: "Smartwatch",
      preco: 800,
      cor: "Prata",
      caracteristicas: ["Monitoramento cardíaco", "Notificações"],
      emEstoque: true })
```

Observações:

- Tipos de Dados: Certifique-se de que os tipos de dados dos valores inseridos correspondam aos tipos de dados definidos no esquema da coleção.
- Arrays: Para inserir arrays, utilize colchetes [] e separe os elementos por vírgulas.
- Objetos: Para inserir objetos aninhados, utilize chaves {}.
- Data: Para inserir datas, utilize o construtor new Date().
- ObjectId: O ObjectId é um tipo de dado especial do MongoDB que gera um valor único para cada documento.

updateOne() e updateMany() no MongoDB

Os métodos `updateOne()` e `updateMany()` são ferramentas poderosas no MongoDB para modificar documentos em uma coleção. A principal diferença entre eles reside no número de documentos que são afetados por uma única operação.

updateOne()

- **Objetivo:** Atualiza apenas o **primeiro documento** que corresponder ao filtro especificado.
- **Sintaxe:**
- JavaScript

```
db.collection.updateOne(
    <filter>,
    <update>,
    <options>
)
```

```
db.usuarios.updateOne( { nome: "João" }, { $set: { idade: 30 } } )
```

Este comando irá encontrar o primeiro documento na coleção "usuarios" onde o campo "nome" é igual a "João" e atualizará o campo "idade" para 30.

updateMany()

- **Objetivo:** Atualiza **todos os documentos** que corresponderem ao filtro especificado.
- **Sintaxe:**
- JavaScript

```
db.collection.updateMany(

    <filter>,
```

```
    <update>,

    <options>

      )
```

```
db.produtos.updateMany( { categoria: "Eletrônicos" }, { $inc: {
preço: 10 } } )
```

Este comando irá encontrar todos os documentos na coleção "produtos" onde o campo "categoria" é igual a "Eletrônicos" e incrementará o campo "preço" em 10 unidades para cada um desses documentos.

Exemplos Adicionais com Operadores de Atualização

- **$set:** Define um novo valor para um campo existente.
- **$inc:** Incrementa um valor numérico.
- **$unset:** Remove um campo.
- **$push:** Adiciona um elemento a um array.
- **$pull:** Remove um elemento de um array.

Exemplo com $push:

```
db.usuarios.updateOne( { _id: ObjectId("6473b29c4b5e7436642b2f37")
}, { $push: { hobbies: "Jogar xadrez" } } )
```

Este comando adicionará "Jogar xadrez" à lista de hobbies do usuário com o ID especificado.

Exemplo com $pull:

```
db.produtos.updateMany( { categoria: "Alimentos" }, { $pull: {
ingredientes: "Glúten" } } )
```

Este comando removerá "Glúten" da lista de ingredientes de todos os produtos da categoria "Alimentos".

Em resumo:

- `updateOne()` é ideal para atualizações específicas em um único documento.
- `updateMany()` é ideal para atualizações em massa de múltiplos documentos.

Exemplos Adicionais com Operadores de Atualização em uma Coleção de Produtos

Vamos explorar mais a fundo os operadores de atualização `$set`, `$inc`, `$unset`, `$push` e `$pull`, utilizando uma coleção de produtos como exemplo.

```
{ "_id": ObjectId("6473b29c4b5e7436642b2f37"),
"nome": "Smartphone X",
"preco": 2000,
"cor": "Preto",
"caracteristicas": ["5G", "Câmera dupla", "Bateria de longa
duração"],
"emEstoque": true }
```

$set: Definindo um Novo Valor

- **Exemplo:** Alterar a cor de um produto

```
db.produtos.updateOne( { nome: "Smartphone X" }, { $set: { cor:
"Azul" } } )
```

Após a execução, o campo "cor" do produto "Smartphone X" será alterado para "Azul".

$inc: Incrementando um Valor Numérico

- **Exemplo:** Aumentar o preço de todos os produtos em 10%

db.produtos.updateMany({}, // Atualiza todos os documentos { $inc: { preco: preco * 0.1 } })

Este comando irá aumentar o preço de todos os produtos em 10%. Note o uso do operador `$inc` com uma expressão matemática para calcular o novo valor.

$unset: Removendo um Campo

- **Exemplo:** Remover o campo "cor" de todos os produtos

```
db.produtos.updateMany( {}, { $unset: { cor: "" } } )
```

Após a execução, o campo "cor" será removido de todos os documentos da coleção.

$push: Adicionando um Elemento a um Array

- **Exemplo:** Adicionar uma nova característica a um produto

```
db.produtos.updateOne( { nome: "Smartphone X" }, { $push: {
caracteristicas: "Carregamento rápido" } } )
```

Este comando adicionará a característica "Carregamento rápido" ao array "caracteristicas" do produto "Smartphone X".

$pull: Removendo um Elemento de um Array

- **Exemplo:** Remover uma característica de todos os produtos

```
db.produtos.updateMany( {}, { $pull: { caracteristicas: "5G" } } )
```

Este comando removerá a característica "5G" do array "caracteristicas" de todos os produtos.

Combinando Operadores

Você pode combinar vários operadores em uma única operação de atualização:

```
db.produtos.updateOne(
{ nome: "Smartphone X" },
{
     $set: { preço: 1800 },
     $push: { caracteristicas: "Resistente à água" },
     $pull: { caracteristicas: "Câmera dupla" }
}
)
```

Este comando irá:

- Alterar o preço do produto "Smartphone X" para 1800.
- Adicionar a característica "Resistente à água" ao produto.
- Remover a característica "Câmera dupla" do produto.

Observações:

- **$set:** É o operador mais utilizado para atribuir um novo valor a um campo.
- **$inc:** É útil para realizar incrementos ou decrementos em campos numéricos.
- **$unset:** É utilizado para remover campos de documentos.
- **$push** e **$pull:** São especialmente úteis para manipular arrays.

deleteOne() e deleteMany() no MongoDB

Os métodos `deleteOne()` e `deleteMany()` são utilizados para remover documentos de uma coleção no MongoDB. A principal diferença entre eles reside no número de documentos que serão removidos:
`deleteOne()` remove apenas o primeiro documento que corresponda ao filtro, enquanto `deleteMany()` remove todos os documentos que correspondam.

Sintaxe Básica:

```
db.collection.deleteOne(filter, options)
```

```
db.collection.deleteMany(filter, options)
```

Exemplos:

```
db.produtos.deleteOne({ nome: "Smartphone X" })
```

```
db.produtos.deleteMany({ preco: { $gt: 2000 } })
```

Método find() no MongoDB

O método `find()` é uma das ferramentas mais poderosas do MongoDB para realizar consultas em suas coleções. Ele permite selecionar documentos que atendem a critérios específicos, oferecendo uma grande flexibilidade para filtrar e extrair dados.

Sintaxe Básica:

```
db.nome_da_colecao.find(query, projection)
```

- ***query***: Um objeto que define os critérios de seleção. É aqui que você especifica os operadores lógicos e relacionais para filtrar os documentos.
- ***projection***: Um objeto opcional que define quais campos você deseja incluir ou excluir da saída.

Operadores Lógicos e Relacionais:

- **Operadores de Igualdade:**
  - `=`: Igualdade exata (ex: `nome: "Smartphone X"`)
- **Operadores de Desigualdade:**
  - `$ne`: Diferente (ex: `cor: {$ne: "Preto"}`)
- **Operadores de Maior e Menor:**
  - `>`: Maior que (ex: `preco: {$gt: 1500}`)
  - `<`: Menor que (ex: `preco: {$lt: 1500}`)
  - `>=`: Maior ou igual a (ex: `preco: {$gte: 1500}`)
  - `<=`: Menor ou igual a (ex: `preco: {$lte: 1500}`)
- **Operadores Lógicos:**
  - `$and`: E lógico (ex: `{ preco: {$gt: 1500}, emEstoque: true }`)
  - `$or`: Ou lógico (ex: `{ cor: "Preto", caracteristicas: "5G" }`)
  - `$not`: Negação (ex: `{ emEstoque: {$not: true} }`)
- **Operadores de Array:**
  - `$in`: Verifica se um valor existe em um array (ex: `{ caracteristicas: {$in: ["5G", "Câmera tripla"]} }`)
  - `$nin`: Verifica se um valor não existe em um array (ex: `{ caracteristicas: {$nin: ["5G"]} }`)
- **Operadores de Expressões Regulares:**
  - `/regex/`: Busca por padrões em strings (ex: `{ nome: /Smartphone/ }`)

Exemplos Utilizando a Coleção "produtos":

1. Encontrar todos os smartphones com preço acima de 1500:

```
db.produtos.find({ preco: { $gt: 1500 } });
```

2. Encontrar smartphones pretos ou com câmera dupla:

```
db.produtos.find({ $or: [{ cor: "Preto" }, { caracteristicas: "Câmera dupla" }] });
```

3. Encontrar smartphones em estoque que não possuem a característica 5G:

```
db.produtos.find({ emEstoque: true, caracteristicas: { $nin: ["5G"] } });
```

4. Encontrar smartphones com nome iniciando com "Smartphone": (^)

```
db.produtos.find({ nome: /^Smartphone/ });
```

5. Encontrar smartphones com nome terminando com "Samsung": ($)

```
db.produtos.find({ nome: /Samsung$/ });
```

6. Projeção: Mostrar apenas o nome e o preço dos produtos:

```
db.produtos.find({}, { nome: 1, preco: 1, _id: 0 }); // _id: 0 para não incluir o _id na saída
```

7. Smartphones Samsung com preço abaixo de 2500

```
db.produtos.find({
    $and: [
        { nome: /Samsung$/ }, // Nome termina com "Samsung"
        { preco: { $lt: 2500 } }
    ]
});
```