Consultas Complexas no MongoDB
Parte 1

QXD0099 - Desenvolvimento de Software para Persistência

Universidade Federal do Ceará - *Campus* Quixadá

Prof. Francisco Victor da Silva Pinheiro
victorpinheiro@ufc.br

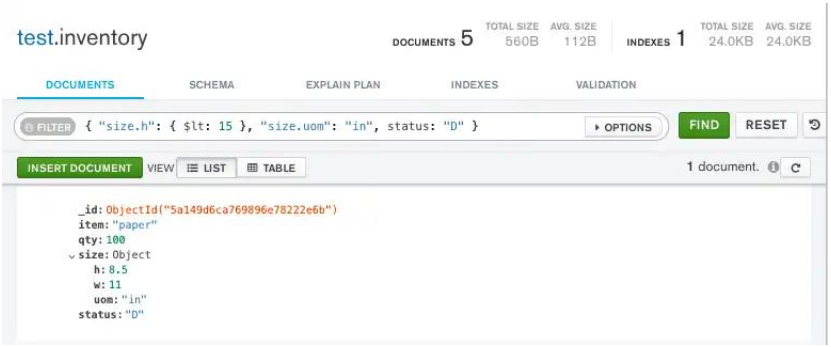
Agenda

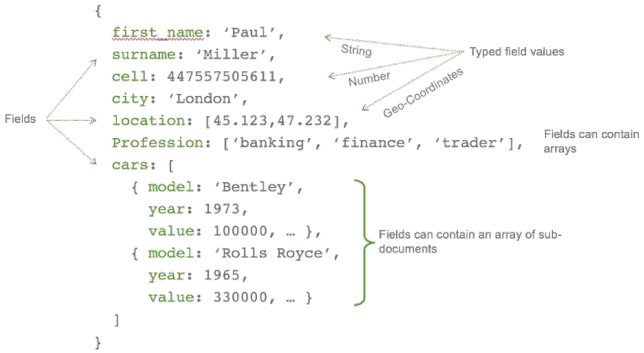
- Filtros Avançados
  - Operadores de Comparação
  - Operadores Lógicos
  - Operador $in e $nin
  - Filtrando Arrays
- Joins no MongoDB
  - Sintaxe do $lookup
  - $lookup com Arrays
  - $lookup com Condições Adicionais
- Paginação
  - Paginação com skip() e limit()
  - Paginação Baseada em Índices (_id ou outro índice)
- Agregações Avançadas Pipeline

Consultas Complexas no MongoDB

- O MongoDB, sendo um banco NoSQL orientado a documentos, não possui joins nativos como bancos de dados relacionais.

- Contudo, ele oferece ferramentas poderosas para realizar consultas complexas, como filtros avançados, lookup (joins simulados), paginação, e pipelines de agregação.





Filtros Avançados

- Os filtros avançados no MongoDB permitem realizar consultas complexas e refinadas em documentos dentro de uma coleção. Para isso, utilizamos operadores de comparação, operadores lógicos e operadores de array, dentre outros.
    - Os operadores de comparação permitem selecionar documentos com base em valores que atendam a condições específicas.
    - Os operadores lógicos permitem combinar múltiplas condições dentro de uma única consulta.
    - O operador $in permite buscar documentos onde um campo tenha um valor pertencente a uma lista especificada.
    - Se um campo contém um array, é possível filtrar documentos baseando-se nos valores contidos dentro desse array.

Operadores de Comparação

- **$gt:** Maior que (Greater Than)
  - { idade: { $gt: 18 } } → Retorna documentos onde idade seja maior que 18.

- **$gte:** Maior ou igual a (Greater Than or Equal)
  - { preco: { $gte: 100 } } → Retorna documentos onde preco seja maior ou igual a 100.

- **$lt:** Menor que (Less Than)
  - { salario: { $lt: 3000 } } → Retorna documentos onde salario seja menor que 3000.

Operadores de Comparação

- **$lte:** Menor ou igual a (Less Than or Equal)
  - { nota: { $lte: 7.5 } } → Retorna documentos onde nota seja menor ou igual a 7.5.

- **$eq:** Igual a (Equal)
  - { cidade: { $eq: "São Paulo" } } → Retorna documentos onde cidade seja exatamente "São Paulo".

- **$ne:** Diferente de (Not Equal)
  - { status: { $ne: "ativo" } } → Retorna documentos onde status não seja "ativo".

Operadores Lógicos

- **or:** Pelo menos uma condição deve ser verdadeira
  - { $or: [{ idade: { $lt: 18 } }, { idade: { $gt: 60 } }] } → Retorna documentos onde idade seja menor que 18 ou maior que 60.

- **$and:** Todas as condições devem ser verdadeiras
  - { $and: [{ idade: { $gte: 18 } }, { idade: { $lte: 60 } }] } → Retorna documentos onde idade esteja entre 18 e 60.

- **$nor:** Nenhuma das condições deve ser verdadeira
  - { $nor: [{ status: "ativo" }, { status: "pendente" }] } → Retorna documentos onde status não seja "ativo" nem "pendente".

- **$not:**Nega a condição especificada
  - { status: { $not: { $eq: "ativo" } } } → Retorna documentos onde status não seja "ativo".

Operador $in e $nin

- **(Pesquisa dentro de um conjunto de valores)**

- **$in:** Verifica se um campo contém um dos valores especificados
  - { cidade: { $in: ["São Paulo", "Rio de Janeiro", "Belo Horizonte"] } } → Retorna documentos onde cidade seja "São Paulo", "Rio de Janeiro" ou "Belo Horizonte".

- **$nin:** Verifica se um campo não contém um dos valores especificados
  - { status: { $nin: ["pendente", "inativo"] } } → Retorna documentos onde status não seja "pendente" nem "inativo".

Filtrando Arrays

- **$all:** Retorna documentos onde o array contenha todos os valores especificados
  - { tags: { $all: ["eletrônicos", "promoção"] } } → Retorna documentos onde tags contenha ambos os valores "eletrônicos" e "promoção".

- **$size:** Retorna documentos onde o array tenha um número exato de elementos
  - { categorias: { $size: 3 } } → Retorna documentos onde o array categorias tenha exatamente 3 elementos.

- **$elemMatch:** Permite filtrar arrays com base em múltiplas condições
  - { avaliacoes: { $elemMatch: { nota: { $gte: 8 }, usuario: "João" } } } → Retorna documentos onde há pelo menos um objeto dentro de avaliacoes onde nota seja maior ou igual a 8 e usuario seja "João".

Exemplos

- **Exemplo 1:** Encontrar usuários que tenham idade entre 25 e 40 anos
  - db.usuarios.find({ idade: { $gte: 25, $lte: 40 } })

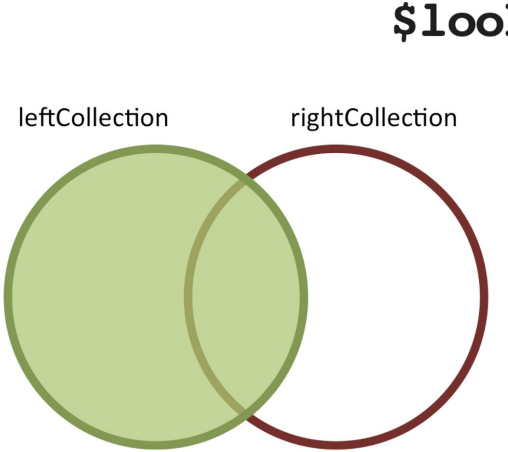
- **Exemplo 2:** Encontrar pedidos onde o status seja "pendente" ou "em processamento"
  - db.pedidos.find({ status: { $in: ["pendente", "em processamento"] } })

- **Exemplo 3:** Encontrar clientes que sejam de São Paulo e tenham compras acima de R$1000
  - db.clientes.find({ $and: [{ cidade: "São Paulo" }, { totalCompras: { $gt: 1000 } }]})

- **Exemplo 4:** Encontrar produtos que tenham no mínimo 2 categorias associadas
  - db.produtos.find({ categorias: { $size: 2 } })

Joins no MongoDB

- No MongoDB, o operador $lookup permite realizar um join entre coleções, similar ao JOIN no SQL. Ele é utilizado dentro do pipeline de agregação e permite combinar documentos de duas coleções com base em um campo comum.




```
db.leftCollection.aggregate(
[{
  $lookUp:
    {
      from: “rightCollection”,
      localField: “leftVal”,
      foreignField: “rightVal”,
      as: “embeddedData”
    }
}])
```

```
[
  {
    _id: 1,
    name: 'Ikigai',
    authorId: 101,
    author: [ { _id: 101, name: 'Hector Garcia', bookId: 1 } ]
  },
  {
    _id: 5,
    name: 'The 5 Love Languages',
    authorId: 105,
    author: [ { _id: 105, name: 'Gary Chapman', bookId: 5 } ]
  },
  {
    _id: 8,
    name: 'The Diversity Myth',
    authorId: 103,
    author: [ { _id: 103, name: 'Peter Thiel', bookId: 3 } ]
  },
  { _id: 7, name: 'Sapiens', authorId: 109, author: [] }
]
```

Sintaxe do $lookup

- O operador **$lookup** possui a seguinte estrutura:

```
{
    $lookup: {
      from: "colecao_destino",
      localField: "campo_local",
      foreignField: "campo_estrangeiro",
      as: "nome_do_array_resultante"
    }
  }
```

- **Parâmetros:**
  - **from:** Nome da coleção de destino (a coleção que será unida).
  - **localField:** Campo na coleção atual que será comparado.
  - **foreignField:** Campo na coleção de destino que será comparado.
  - **as:** Nome do array que armazenará os documentos correspondentes.

Exemplo de Uso

- Cenário: Banco de Dados de Compras
- **Coleção clientes**

```
[
    { "_id": 1, "nome": "João", "cidade": "São Paulo", "id_pedido": 101 },
    { "_id": 2, "nome": "Maria", "cidade": "Rio de Janeiro", "id_pedido": 102 },
    { "_id": 3, "nome": "Pedro", "cidade": "Belo Horizonte", "id_pedido": 103 }
  ]
```

- **Coleção pedidos**

```
[
    { "_id": 101, "produto": "Notebook", "valor": 4500 },
    { "_id": 102, "produto": "Smartphone", "valor": 2500 },
    { "_id": 103, "produto": "Tablet", "valor": 1500 }
  ]
```

Exemplo de Uso

- Agora, queremos buscar os clientes junto com os detalhes de seus pedidos.
- Consulta usando $lookup

```
db.clientes.aggregate([
    {
      $lookup: {
        from: "pedidos",
        localField: "id_pedido",
        foreignField: "_id",
        as: "detalhes_pedido"
      }
    }
  ])
```

Exemplo de Uso

- O resultado da consulta utilizando $lookup no MongoDB é uma junção (join) entre as coleções clientes e pedidos, de forma semelhante a um JOIN em bancos relacionais.

- Cada cliente agora tem um novo campo detalhes_pedido, que contém um array com os pedidos correspondentes.

- Se um cliente tivesse mais de um pedido correspondente (não é o caso deste exemplo), o array poderia ter mais de um item.

Exemplo de Uso

```
[
    {
      "_id": 1,
      "nome": "João",
      "cidade": "São Paulo",
      "id_pedido": 101,
      "detalhes_pedido": [
        { "_id": 101, "produto": "Notebook", "valor": 4500 }
      ]
    },
    {
      "_id": 2,
      "nome": "Maria",
      "cidade": "Rio de Janeiro",
      "id_pedido": 102,
      "detalhes_pedido": [
        { "_id": 102, "produto": "Smartphone", "valor": 2500 }
      ]
    }
  ]
```

$lookup com Arrays

- Se foreignField for um array, $lookup irá verificar se qualquer um dos elementos no array corresponde ao localField.

- **Exemplo**
  - Suponha que na coleção clientes, cada cliente possa ter múltiplos pedidos:

- Coleção clientes (modificada)

```
[
    { "_id": 1, "nome": "João", "cidade": "São Paulo", "ids_pedidos": [101, 102] },
    { "_id": 2, "nome": "Maria", "cidade": "Rio de Janeiro", "ids_pedidos": [103] }
  ]
```

$lookup com Arrays

- Agora, ajustamos a consulta para buscar vários pedidos por cliente:
- Aqui, **$lookup** faz uma correspondência múltipla:
  - Como ids_pedidos é um array, ele encontra todos os pedidos correspondentes e os adiciona no campo detalhes_pedidos.

```
db.clientes.aggregate([
    {
      $lookup: {
        from: "pedidos",
        localField: "ids_pedidos",
        foreignField: "_id",
        as: "detalhes_pedidos"
      }
    }
  ])
```

$lookup com Arrays

- Agora, cada cliente tem um array detalhes_pedidos com todos os pedidos associados:

```
[
    {
      "_id": 1,
      "nome": "João",
      "cidade": "São Paulo",
      "ids_pedidos": [101, 102],
      "detalhes_pedidos": [
        { "_id": 101, "produto": "Notebook", "valor": 4500 },
        { "_id": 102, "produto": "Smartphone", "valor": 2500 }
      ]
    },
    {
      "_id": 2,
      "nome": "Maria",
      "cidade": "Rio de Janeiro",
      "ids_pedidos": [103],
      "detalhes_pedidos": [
        { "_id": 103, "produto": "Tablet", "valor": 1500 }
      ]
    }
  ]
```

$lookup com Arrays

- No exemplo anterior, cada cliente tinha um único pedido (id_pedido como um valor único).

- Agora, os clientes podem ter múltiplos pedidos (ids_pedidos como um array), permitindo que $lookup retorne vários resultados.

$lookup com Condições Adicionais

- Se precisarmos filtrar os resultados dentro do $lookup, podemos usar $pipeline (disponível a partir do MongoDB 3.6):

```
db.clientes.aggregate([
    {
      $lookup: {
        from: "pedidos",
        let: { cliente_pedido_id: "$id_pedido" },
        pipeline: [
          { $match: { $expr: { $eq: ["$_id", "$$cliente_pedido_id"] } } },
          { $match: { valor: { $gt: 2000 } } }  // Filtra pedidos acima de R$2000
        ],
        as: "detalhes_pedido"
      }
    }
  ])
```

$lookup com Condições Adicionais

- O $lookup é uma ferramenta poderosa para realizar joins entre coleções no MongoDB. Com ele, podemos:
  - Relacionar coleções facilmente, como clientes e pedidos.
  - Lidar com arrays para múltiplos relacionamentos.
  - Usar $unwind para transformar arrays em documentos individuais.
  - Aplicar filtros adicionais para refinar os resultados.

Paginação

- A paginação no MongoDB é essencial para trabalhar com grandes conjuntos de dados, pois permite exibir resultados de maneira eficiente e evitar sobrecarga na consulta. Existem duas abordagens principais para paginação:

- Usando **skip()** e **limit()** (adequado para pequenos conjuntos de dados).

- **Usando cursores baseados em índice** (melhor para grandes volumes de dados).

Paginação com skip() e limit()

- Essa abordagem é simples e funciona bem quando o número de documentos é relativamente pequeno. O método skip(n) pula os primeiros n documentos e limit(m) define quantos documentos serão retornados.

- Exemplo:

```
# Configuração de página
pagina = 2
tamanho_pagina = 5

# Consulta com paginação
resultado = db.usuarios.find().skip((pagina - 1) * tamanho_pagina).limit(tamanho_pagina)

# Exibir resultados
for doc in resultado:
    print(doc)
```

Paginação com skip() e limit()

- pagina = 2: Define a página atual.
- tamanho_pagina = 5: Define quantos documentos devem ser retornados por página.
- (pagina - 1) * tamanho_pagina: Calcula quantos documentos devem ser ignorados antes de retornar os próximos.

Paginação com skip() e limit()

- **Problema com skip()**
  - Quando lidamos com grandes volumes de dados, skip() se torna ineficiente, pois o MongoDB ainda precisa percorrer todos os documentos anteriores para encontrar a página correta.

  - Quanto maior o número de documentos ignorados (skip()), mais tempo a consulta leva.

Paginação Baseada em Índices (_id ou outro índice)

- Para melhorar o desempenho da paginação, podemos utilizar cursores baseados em índices, como o campo _id. Em vez de usar skip(), filtramos os documentos com base no último ID da página anterior.

```
from bson.objectid import ObjectId

# Último ID da página anterior (obtido dinamicamente)
ultimo_id = "64d1f7c9b2a1b23e4556abcd"

# Consulta paginada usando índice
resultado = db.usuarios.find({"_id": {"$gt": ObjectId(ultimo_id)}}).limit(5)

# Exibir resultados
for doc in resultado:
    print(doc)
```

Paginação Baseada em Índices (_id ou outro índice)

- O campo _id no MongoDB é sequencialmente crescente, o que permite usá-lo para buscas eficientes.
- A consulta filtra apenas os documentos cujo _id seja maior que o último ID da página anterior.
- limit(5) controla o tamanho da página.
- Essa abordagem é muito mais eficiente que skip(), pois o MongoDB usa o índice _id diretamente, evitando percorrer documentos desnecessários.

Comparação das Abordagens

Método	Vantagens	Desvantagens
`skip()` + `limit()`	Simples de implementar. Funciona bem para poucos documentos.	Ineficiente para grandes coleções (o desempenho degrada com valores altos de `skip()`).
Baseado em `_id`	Alta performance para grandes coleções. Uso otimizado de índices.	Requer armazenar o último ID da página anterior.

Agregações Avançadas Pipeline

- O pipeline de agregação processa documentos através de estágios sequenciais, onde cada estágio executa uma operação e passa o resultado para o próximo. Isso permite transformar e analisar os dados de maneira eficiente.

- Principais vantagens do pipeline de agregação:
  - Transformação de dados (filtragem, projeção, cálculo).
  - Agrupamento e agregação (médias, contagens, soma, etc.).
  - Ordenação e filtragem avançada.
  - Melhor desempenho comparado a consultas tradicionais com find().

Exemplo 1: Contar usuários por idade

- Agrupamos os usuários com base na idade e contamos quantos existem em cada grupo.

```
pipeline = [
    {"$group": {"_id": "$idade", "total": {"$sum": 1}}}
]

resultado = db.usuarios.aggregate(pipeline)

# Exibir resultados
for doc in resultado:
    print(doc)
```

Exemplo 1: Contar usuários por idade

- **$group:** Agrupa os usuários pelo campo idade.
- **"_id": "$idade":** Define a chave de agrupamento.
- **total: { "$sum": 1 }:** Conta quantos documentos existem para cada idade.

```
{ "_id": 25, "total": 10 }
{ "_id": 30, "total": 8 }
{ "_id": 35, "total": 5 }
```

- Cada linha representa uma idade e o número de usuários que possuem essa idade.

Exemplo 2: Filtrar, Agrupar e Ordenar

- Aqui, filtramos usuários com idade maior ou igual a 25, agrupamos por idade e ordenamos o resultado de forma decrescente.

```
pipeline = [
    {"$match": {"idade": {"$gte": 25}}},  # Filtrar usuários com idade >= 25
    {"$group": {"_id": "$idade", "total": {"$sum": 1}}},  # Agrupar por idade
    {"$sort": {"total": -1}}  # Ordenar em ordem decrescente
]

resultado = db.usuarios.aggregate(pipeline)

# Exibir resultados
for doc in resultado:
    print(doc)
```

Exemplo 2: Filtrar, Agrupar e Ordenar

- **$match:** Filtra usuários cuja idade é maior ou igual a 25.
- **$group:** Agrupa os usuários pela idade e conta o total em cada grupo.
- **$sort:** Ordena os resultados pelo total de usuários em ordem decrescente (-1).

```
{ "_id": 30, "total": 15 }
{ "_id": 25, "total": 10 }
{ "_id": 40, "total": 5 }
```

- Isso mostra que existem 15 usuários com 30 anos, 10 usuários com 25 anos, e 5 usuários com 40 anos.

Exemplo 3: Selecionar apenas alguns campos com $project

- Suponha que queremos exibir apenas o nome e a idade dos usuários, excluindo _id e outros dados.

```
pipeline = [
    {"$project": {"nome": 1, "idade": 1, "_id": 0}}
]

resultado = db.usuarios.aggregate(pipeline)

# Exibir resultados
for doc in resultado:
    print(doc)
```

Exemplo 3: Selecionar apenas alguns campos com $project

- **$project:** Permite modificar quais campos serão exibidos no resultado.
- **"nome":** 1, "idade": 1: Inclui apenas nome e idade.
- **"_id":** 0: Remove o campo _id.

```
{ "nome": "Ana", "idade": 25 }
{ "nome": "Carlos", "idade": 30 }
{ "nome": "Mariana", "idade": 35 }
```

Referências

- MongoDB – Site oficial
  - http://www.mongdb.com
- MongoDB Manual
  - http://docs.mongodb.org/manual/
  - http://docs.mongodb.org/manual/MongoDBmanual.pdf
- Slides: Building your first app: an introduction to MongoDB. Norman Graham. Consulting Engineer, 10gen.
- Slides: mongoDB.
  - Júlio Monteiro (julio@monteiro.eti.br).
- Slides Why MongoDB Is Awesome
  - John Nunemaker - Ordered List (john@orderedlist.com)

Obrigado!

Dúvidas?

Universidade Federal do Ceará - *Campus* Quixadá

Prof. Francisco Victor da Silva Pinheiro
victorpinheiro@ufc.br