Consultas Complexas no MongoDB
Parte 2

QXD0099 - Desenvolvimento de Software para Persistência

Universidade Federal do Ceará - *Campus* Quixadá

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

Agenda

- Ordenação
- Busca Textual
- Índices Textuais em Múltiplos Campos no MongoDB
- Uso de Projeções
- Índices para Consultas Mais Rápidas
- Boas Práticas

Ordenação (sort)

- A função **.sort()** permite ordenar os documentos com base em um campo específico.
- Exemplo: Vamos supor que temos uma coleção chamada acoes com os seguintes dados:

```
[
    {"_id": 1, "ticker": "AAPL", "preco": 175.50, "setor": "Tecnologia"},
    {"_id": 2, "ticker": "TSLA", "preco": 650.30, "setor": "Automobilístico"},
    {"_id": 3, "ticker": "AMZN", "preco": 3200.80, "setor": "E-commerce"},
    {"_id": 4, "ticker": "GOOGL", "preco": 2800.60, "setor": "Tecnologia"},
    {"_id": 5, "ticker": "MSFT", "preco": 299.10, "setor": "Tecnologia"}
]
```

Ordenação (sort)

- Ordenando por preço de forma decrescente

```
from pymongo import MongoClient

client = MongoClient("mongodb://localhost:27017/")
db = client["investimentos"]
colecao = db["acoes"]

resultado = colecao.find().sort("preco", -1)  # Ordenação decrescente

for doc in resultado:
    print(doc)
```

Ordenação (sort)

- Resultado esperado

```
[
    {"_id": 3, "ticker": "AMZN", "preco": 3200.80, "setor": "E-commerce"},
    {"_id": 4, "ticker": "GOOGL", "preco": 2800.60, "setor": "Tecnologia"},
    {"_id": 2, "ticker": "TSLA", "preco": 650.30, "setor": "Automobilístico"},
    {"_id": 5, "ticker": "MSFT", "preco": 299.10, "setor": "Tecnologia"},
    {"_id": 1, "ticker": "AAPL", "preco": 175.50, "setor": "Tecnologia"}
]
```

- Dica: Criar um índice no campo de ordenação melhora a performance:

```
colecao.create_index([("preco", 1)])
```

Busca Textual

- O MongoDB permite pesquisar textos em campos indexados usando um índice textual.
- Criando um índice para busca textual

```
colecao.create_index([("ticker", "text")])
```

- Consultando ações que contenham **"AAPL"** no nome

```
resultado = colecao.find({"$text": {"$search": "AAPL"}})

for doc in resultado:
    print(doc)
```

Busca Textual

- Resultado esperado

```
[
    {"_id": 1, "ticker": "AAPL", "preco": 175.50, "setor": "Tecnologia"}
]
```

- Para buscas mais refinadas, você pode usar:
  - {"$text": {"$search": "\"mercado financeiro\""}}
    - Isso busca a frase exata "mercado financeiro"

Índices Textuais em Múltiplos Campos

- Os índices textuais no MongoDB permitem buscas eficientes em textos armazenados nos documentos. Você pode definir um índice textual em múltiplos campos para buscar informações em diferentes atributos ao mesmo tempo.

- **Como criar um índice textual em múltiplos campos**
  - Se quisermos pesquisar palavras em mais de um campo, precisamos criar um índice textual que abranja esses campos.

- **Exemplo de Estrutura de Dados**
  - Suponha que temos uma coleção notícias com os seguintes documentos:

Índices Textuais em Múltiplos Campos

```
[
    {"_id": 1, "titulo": "Apple lança novo iPhone", "conteudo": "O iPhone 16 traz novas funcionalidades",
"categoria": "tecnologia"},
    {"_id": 2, "titulo": "Tesla lança carro autônomo", "conteudo": "O novo Tesla Model Y pode dirigir
sozinho", "categoria": "automobilismo"},
    {"_id": 3, "titulo": "Microsoft apresenta nova IA", "conteudo": "A inteligência artificial da Microsoft
pode gerar textos", "categoria": "tecnologia"},
    {"_id": 4, "titulo": "Amazon expande negócios", "conteudo": "A Amazon está expandindo para novos
mercados", "categoria": "e-commerce"}
]
```

- Se quisermos permitir que os usuários pesquisem tanto no título quanto no conteúdo, precisamos criar um índice textual para esses campos.

Índices Textuais em Múltiplos Campos

- Podemos criar um índice textual que inclua tanto o campo título quanto conteudo:
- Esse índice permitirá buscas eficientes nos dois campos ao mesmo tempo.

```
from pymongo import MongoClient

client = MongoClient("mongodb://localhost:27017/")
db = client["meu_banco"]
colecao = db["noticias"]

# Criar um índice textual nos campos "titulo" e "conteudo"
colecao.create_index([("titulo", "text"), ("conteudo", "text")])
```

Índices Textuais em Múltiplos Campos

- Agora, podemos realizar pesquisas usando **$text** para buscar palavras-chave que aparecem em qualquer um dos campos indexados.

- **Exemplo 1:** Buscar notícias que contenham a palavra "Tesla"

```
resultado = colecao.find({"$text": {"$search": "Tesla"}})

for doc in resultado:
    print(doc)
```

- Resultado esperado: A busca encontrou o documento onde "Tesla" aparece no título.

```
[
    {"_id": 2, "titulo": "Tesla lança carro autônomo", "conteudo": "O novo Tesla Model Y pode dirigir sozinho", "categoria": "automobilismo"}
]
```

Índices Textuais em Múltiplos Campos

- **Exemplo 2:** Buscar notícias contendo "Tesla" mas excluindo "autônomo"

```
resultado = colecao.find({"$text": {"$search": "Tesla -autônomo"}})

for doc in resultado:
    print(doc)
```

- Retorna notícias que mencionam Tesla, mas que não contêm a palavra "autônomo".

Índices Textuais em Múltiplos Campos

- **Exemplo 3:** Ordenar os resultados por relevância

```
resultado = colecao.find(
    {"$text": {"$search": "Tesla"}},
    {"score": {"$meta": "textScore"}}  # Adiciona a relevância na busca
).sort([("score", {"$meta": "textScore"})])  # Ordena pelos resultados mais relevantes

for doc in resultado:
    print(doc)
```

- Isso faz com que os documentos mais relevantes apareçam primeiro.

Uso de Projeções

- Podemos limitar os campos retornados usando projection.
- **Exemplo:** Buscar apenas o **ticker** e **preco** das ações

```
resultado = colecao.find({}, {"_id": 0, "ticker": 1, "preco": 1})

for doc in resultado:
    print(doc)
```

- **Resultado esperado**

```
[
    {"ticker": "AAPL", "preco": 175.50},
    {"ticker": "TSLA", "preco": 650.30},
    {"ticker": "AMZN", "preco": 3200.80},
    {"ticker": "GOOGL", "preco": 2800.60},
    {"ticker": "MSFT", "preco": 299.10}
]
```

Uso de Projeções

- Se você quiser excluir um campo específico (por exemplo, dividendo), faça:

```
resultado = colecao.find({}, {"dividendo": 0})

for doc in resultado:
    print(doc)
```

- **Regras de Exclusão:**
  - Você pode definir 0 para excluir um campo.
  - Não pode misturar 0 e 1 na mesma projeção (exceto para _id).

Uso de Projeções

- Projeção em Subdocumentos
  - Se um documento tem um subdocumento como este:

```
{
    "_id": ObjectId("601d4f4cfc13ae34b8000001"),
    "nome": "Ação ABC",
    "setor": "Financeiro",
    "preco": 200.75,
    "empresa": {
        "nome": "ABC Corp",
        "pais": "Brasil"
    }
}
```

Uso de Projeções

- E queremos apenas o nome da ação e o nome da empresa:

```
resultado = colecao.find({}, {"_id": 0, "nome": 1, "empresa.nome": 1})

for doc in resultado:
    print(doc)
```

- Saída esperada:

```
{"nome": "Ação ABC", "empresa": {"nome": "ABC Corp"}}
```

Resumo das Técnicas de Projeção

Técnica	Exemplo
Retornar campos específicos	{"_id": 0, "nome": 1, "preco": 1}
Omitir campos	{"dividendo": 0}
Trabalhar com subdocumentos	{"empresa.nome": 1}
Limitar arrays ($slice)	{"historico_precos": {"$slice": 2}}
Filtrar arrays ($elemMatch)	{"transacoes": {"$elemMatch": {"tipo": "compra"}}}
Aplicar projeção com filtro	{"preco": {"$gt": 100}}, {"_id": 0, "nome": 1, "preco": 1}

Índices para Consultas Mais Rápidas

- Os índices aceleram buscas e ordenações. Sem índices, o MongoDB faz uma varredura completa nos documentos, impactando o desempenho.

- **Criar índice simples:**
  - Isso melhora consultas que filtram ou ordenam pelo preço (preco).

```
colecao.create_index([("preco", 1)])
```

- **Exemplo:** Essa consulta será muito mais rápida porque o MongoDB usará o índice para encontrar documentos com preco maior ou igual a 500.

```
colecao.find({"preco": {"$gte": 500}})
```

Índices para Consultas Mais Rápidas

- **Criar índice composto para consultas envolvendo múltiplos campos:**
- Esse índice será útil para buscas que filtram pelo nome e ordenam pelo preco de forma decrescente (-1).

```
colecao.create_index([("nome", 1), ("preco", -1)])
```

- O MongoDB pode utilizar um índice composto para consultas como:

```
colecao.find({"nome": "Ação XPTO"}).sort("preco", -1)
```

- Como **nome** e **preco** fazem parte do índice composto, a busca será otimizada.

Índices para Consultas Mais Rápidas

- Verificar índices criados:

```
print(list(colecao.list_indexes()))
```

- Isso retorna os índices ativos na coleção, como:

- **Podemos ver:**
  - O índice padrão no _id (sempre existe por padrão).
  - O índice simples no preco.
  - O índice composto em nome e preco.

```
[
    {"v": 2, "key": {"_id": 1}, "name": "_id_"},
    {"v": 2, "key": {"preco": 1}, "name": "preco_1"},
    {"v": 2, "key": {"nome": 1, "preco": -1}, "name":
"nome_1_preco_-1"}
]
```

Índices para Consultas Mais Rápidas

- Utilize explain() para verificar o impacto dos índices nas consultas:
- O comando explain("executionStats") mostra como uma consulta está sendo executada e se está usando índices.

```
resultado = colecao.find({"nome": "Ação XPTO"}).explain("executionStats")
print(resultado)
```

Índices para Consultas Mais Rápidas

- Resultado

```
{
    "executionStats": {
        "executionSuccess": true,
        "nReturned": 1,
        "totalKeysExamined": 1,
        "totalDocsExamined": 1,
        "executionTimeMillis": 1
    },
    "queryPlanner": {
        "winningPlan": {
            "stage": "IXSCAN",
            "keyPattern": {"nome": 1, "preco": -1}
        }
    }
}
```

- `totalKeysExamined`: Número de chaves analisadas no índice (baixo = eficiente).
- `totalDocsExamined`: Número de documentos percorridos (se for alto, pode indicar necessidade de otimização).
- `"stage": "IXSCAN"` indica que **o índice foi usado** na busca.

Boas Práticas

- **Evite consultas que percorrem toda a coleção:** Sempre que possível, utilize filtros eficientes e índices para evitar COLLSCAN (varredura completa).
- **Use projeções para limitar os campos retornados:** Isso reduz o tráfego de rede e melhora o desempenho.
- **Escolha índices com base nas consultas mais frequentes:** Criar muitos índices pode degradar a performance de escrita.
- **Evite $where e expressões JavaScript:** São lentas e bloqueiam otimizações de índice.
- **Prefira agregações para análises complexas:** O aggregate() permite sumarizações avançadas sem trazer todos os documentos para a aplicação.

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