
alpha
<ed/tech>
BANCOS DE DADOS
</>

Banco de Dados Não-Relacional

MONGODB

Bem-vindos à nossa aula sobre MongoDB, um banco de dados NoSQL que tem ganhado cada vez mais destaque no mundo da tecnologia. Durante esta jornada, exploraremos não apenas os conceitos básicos do MongoDB, mas também nos aprofundaremos em tópicos avançados, como o uso de variáveis de ambiente, tipos de índices, consultas otimizadas e boas práticas de desenvolvimento.

Nosso objetivo é fornecer a vocês, estudantes, uma compreensão abrangente do MongoDB e capacitá-los a aplicar esses conhecimentos em seus projetos futuros. Vamos mergulhar nos detalhes e explorar como o MongoDB pode ser uma ferramenta poderosa para armazenar e manipular dados em aplicações modernas.

A seguir, apresentaremos os tópicos que abordaremos nesta aula:

I. **Variáveis de Ambiente com python-dotenv**
   - A. O que são variáveis de ambiente?
   - B. Como configurar e utilizar variáveis de ambiente com python-dotenv no contexto do MongoDB.

II. **Tipos de Índices**
   - A. Índices simples
   - B. Índices compostos
   - C. Índices únicos
   - D. Índices parciais

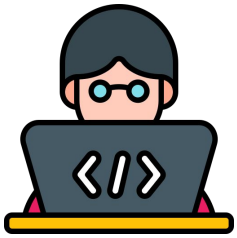
III. **Consultas com Índices**
   - A. Estratégias de otimização de consultas
   - B. Utilização de índices em consultas
   - C. Exemplos práticos de consultas com índices

IV. **Boas Práticas no Desenvolvimento**
   - A. Modelagem de dados eficiente
   - B. Gerenciamento de transações
   - C. Segurança e autenticação
   - D. Monitoramento e otimização de desempenho

Preparem-se para embarcar nesta jornada de descoberta e aprendizado sobre o MongoDB. Vamos explorar juntos como aproveitar ao máximo esse poderoso banco de dados em nossos projetos.

Explorando MongoDB

Variáveis de Ambiente



Variáveis de ambiente são elementos fundamentais no desenvolvimento de software, permitindo configurar e ajustar o comportamento de aplicativos e sistemas operacionais. Vamos explorar como utilizar essas variáveis em conjunto com a biblioteca python-dotenv, especialmente no contexto do MongoDB.

As variáveis de ambiente são valores dinâmicos que podem afetar o comportamento de processos em um sistema operacional. Elas são utilizadas para armazenar informações como diretórios de arquivos, configurações de sistema, chaves de acesso e outros dados importantes para o funcionamento de aplicativos.

Imagine que você tenha um programa Python que precisa acessar um banco de dados. Em vez de codificar diretamente as informações de conexão (como nome de usuário, senha e host do banco de dados) dentro do seu script, você pode definir essas informações como variáveis de ambiente. Isso permite maior flexibilidade e segurança, pois você pode alterar as configurações de conexão sem precisar modificar o código-fonte.

Uma analogia útil para entender variáveis de ambiente é pensar nelas como notas adesivas que você cola em diferentes partes de sua casa para lembrá-lo de tarefas específicas. Cada nota (variável de ambiente) contém uma instrução ou informação relevante para uma determinada área ou processo. Por exemplo: senhas, usuários, etc...

Configurar

Configuração
Para configurar variáveis de ambiente usando python-dotenv, primeiro precisamos criar um arquivo chamado .env. Este arquivo é um simples arquivo de texto onde colocaremos nossas variáveis de ambiente no formato NOME=VALOR. Por exemplo:

```
MONGODB_URI=mongodb://localhost:27017/minha_base_de_dados
```

Isso define uma variável de ambiente chamada MONGODB_URI, que contém o URI de conexão para o MongoDB.

Utilização
Depois de configurar nossas variáveis de ambiente no arquivo .env, podemos utilizar o python-dotenv para carregar essas variáveis em nosso código Python. Primeiro, instale o pacote python-dotenv através do pip:

```
pip install python-dotenv
```

Em seguida, dentro do nosso código Python, podemos carregar as variáveis de ambiente do arquivo .env da seguinte forma:

```
from dotenv import load_dotenv
import os

# Carregar variáveis de ambiente
load_dotenv()

# Utilizar as variáveis de ambiente
mongodb_uri = os.getenv('MONGODB_URI')
```

Agora, a variável mongodb_uri conterá o valor definido no arquivo .env, permitindo que nosso código Python se conecte ao MongoDB sem expor informações sensíveis, como credenciais de banco de dados, diretamente no código fonte.

Banco de Dados Não-Relacional

Variáveis de ambiente

História

As variáveis de ambiente têm suas raízes nos sistemas operacionais Unix, onde eram usadas para definir configurações globais e parâmetros para os programas em execução.

Com o tempo, as variáveis de ambiente se tornaram uma parte fundamental dos sistemas operacionais modernos, sendo adotadas por sistemas como Windows, MacOS e distribuições Linux.

O uso de variáveis de ambiente se tornou crucial no desenvolvimento de software, especialmente em ambientes de computação em nuvem e contêineres, onde a portabilidade e a flexibilidade são essenciais.

Quando você instala um novo software em seu computador, muitas vezes precisa configurar variáveis de ambiente específicas para que o programa funcione corretamente. Por exemplo, ao instalar um servidor web, você pode precisar definir a variável de ambiente PATH para incluir o diretório onde o servidor está instalado, permitindo que o sistema encontre e execute os binários corretos.

Você pode comparar as variáveis de ambiente com as tradições e costumes de uma comunidade. Assim como diferentes comunidades têm suas próprias práticas e valores que afetam o comportamento das pessoas que vivem nelas, as variáveis de ambiente afetam o comportamento dos programas em um sistema operacional, influenciando como eles interagem entre si e com o ambiente ao seu redor.

Arquivo Exemplo: example.env

Este arquivo serve como um modelo para o desenvolvedor, mostrando quais variáveis de ambiente são necessárias e como devem ser configuradas. Ao criar o arquivo .env a partir do example.env, o desenvolvedor preenche os valores específicos para sua configuração local, garantindo consistência e segurança em todo o processo de desenvolvimento.

Um exemplo do arquivo .env (também chamado de example.env ou .env.example) poderia ser:

```
# Configuração do MongoDB
MONGODB_URI=mongodb://localhost:27017/minha_base_de_dados
```

Banco de Dados Não-Relacional

Tipos de Dados Suportados

O MongoDB é conhecido por sua flexibilidade em lidar com diferentes tipos de dados, e o pymongo nos permite manipular esses dados de forma eficiente em nossos projetos Python. Vamos mergulhar nos diferentes tipos de dados e como podemos trabalhar com eles usando pymongo.

Documentos BSON

Começaremos falando sobre os documentos BSON. BSON, que significa "Binary JSON", é o formato de armazenamento de dados usado pelo MongoDB. Ele é semelhante ao JSON, mas com suporte a tipos de dados adicionais, como objetos binários e referências. No pymongo, os documentos BSON são a unidade básica de armazenamento e manipulação de dados no MongoDB.

Imagine que estamos desenvolvendo um sistema de gerenciamento de biblioteca. Podemos representar cada livro como um documento BSON, onde podemos incluir informações como título, autor, ano de publicação e lista de categorias. Esses documentos BSON podem ser facilmente inseridos, atualizados e consultados usando pymongo.

```
from pymongo import MongoClient

# Conectando ao servidor MongoDB local
client = MongoClient('localhost', 27017)

# Acessando o banco de dados 'biblioteca' (ou criando-o se não existir)
db = client['biblioteca']

# Criando uma coleção 'livros' (ou acessando-a se já existir)
livros = db['livros']

# Inserindo um documento BSON representando um livro
livro = {
    'titulo': 'Dom Casmurro',
    'autor': 'Machado de Assis',
    'ano publicacao': 1899,
    'categorias': ['Ficção', 'Romance']
}
livros.insert_one(livro)

# Consultando todos os livros na coleção
for livro in livros.find():
    print(livro)
```

Banco de Dados Não-Relacional

Tipos de Dados Suportados

O MongoDB é conhecido por sua flexibilidade em lidar com diferentes tipos de dados, e o pymongo nos permite manipular esses dados de forma eficiente em nossos projetos Python. Vamos mergulhar nos diferentes tipos de dados e como podemos trabalhar com eles usando pymongo.

Tipos de dados

Os tipos de dados básicos são os tipos simples que todos estamos familiarizados, como strings, inteiros, floats e booleanos. No MongoDB, esses tipos de dados são suportados diretamente e podem ser armazenados em campos de documentos BSON.

```
# Armazenando um post de blog como um documento BSON
postagem = {
    'titulo': 'Introdução ao MongoDB com pymongo',
    'conteudo': 'Neste post, vamos aprender sobre o uso do pymongo
para interagir com o MongoDB.',
    'curtidas': 100,
    'publicado': True
}
```

Suponha que estamos desenvolvendo um aplicativo de blog. Podemos armazenar o conteúdo de cada post como uma string, o número de curtidas como um inteiro e o status de publicação como um booleano em documentos BSON separados para cada postagem.

Tipos de dados Complexos

Além dos tipos básicos, o MongoDB suporta tipos de dados complexos, como listas e dicionários, que podem ser aninhados dentro de documentos BSON. Isso nos permite representar estruturas de dados mais complexas e hierárquicas em nossos dados.

```
# Criando um documento BSON para uma tarefa de gerenciamento
tarefa = {
    'descricao': 'Implementar autenticação de usuário',
    'etiquetas': ['Segurança', 'Autenticação'],
    'responsavel': {
        'nome': 'João',
        'email': 'joao@example.com',
        'cargo': 'Desenvolvedor'
    }
}
```

Imagine que estamos construindo um aplicativo de gerenciamento de tarefas. Podemos ter um documento BSON para cada tarefa, onde podemos incluir uma lista de etiquetas associadas a essa tarefa e um dicionário para representar as informações do responsável pela tarefa, como nome, e-mail e cargo.

Banco de Dados Não-Relacional

Tipos de Dados Suportados

Organização relacional

Embora o MongoDB seja um banco de dados NoSQL e não suporte relações como em bancos de dados relacionais tradicionais, ainda podemos organizar nossos dados de forma relacional, estabelecendo conexões entre documentos BSON por meio de referências ou aninhamento.

Suponha que estamos desenvolvendo um sistema de comércio eletrônico. Podemos ter documentos BSON separados para produtos e pedidos. Para estabelecer a relação entre eles, podemos incluir o ID do produto em cada pedido ou aninhar os detalhes do produto diretamente no documento de pedido.

```
# Documento BSON para um produto
produto = {
    ' id': 1,
    'nome': 'Camiseta',
    'preco': 29.99
}

# Documento BSON para um pedido
pedido = {
    'id pedido': 123,
    'itens': [
        {'produto_id': 1, 'quantidade': 2},
        {'produto_id': 2, 'quantidade': 1}
    ]
}
```

Em resumo, o pymongo nos permite trabalhar com uma ampla gama de tipos de dados suportados pelo MongoDB, desde documentos BSON até estruturas de dados complexas. Compreender esses tipos de dados e como trabalhar com eles é fundamental para desenvolver aplicativos eficientes e escaláveis usando MongoDB e pymongo. Espero que esta aula tenha fornecido uma compreensão clara dos diferentes tipos de dados e suas aplicações no contexto do MongoDB e pymongo.

Banco de Dados Não-Relacional

Tipos de Índices

Vamos explorar os tipos de índices em bancos de dados, uma parte fundamental para otimizar a busca e recuperação de dados.

Imagine que um livro é uma grande biblioteca de informações, e os índices são como os índices alfabéticos que você encontra no final de um livro, que ajudam a encontrar informações específicas de forma rápida e eficiente.

Para entendermos melhor o conceito de índices em bancos de dados, podemos fazer uma analogia com um livro. Um índice em um livro é como um guia rápido que nos ajuda a encontrar páginas específicas onde determinados assuntos são discutidos.

Da mesma forma, em um banco de dados, um índice é uma estrutura de dados que permite aos sistemas de gerenciamento de banco de dados (SGBDs) encontrar rapidamente os registros correspondentes a determinados critérios de busca.

Índices desempenham um papel crucial na otimização de consultas em bancos de dados, permitindo a recuperação eficiente de dados. Compreender os diferentes tipos de índices disponíveis e como usá-los adequadamente pode ajudar a melhorar significativamente o desempenho das consultas em seus sistemas.

Índices simples

Os índices simples são aqueles criados em apenas um campo de uma coleção de dados. Eles são eficazes para melhorar a velocidade de busca quando estamos procurando por valores específicos em um único campo.

Vamos ver um exemplo prático usando a biblioteca pymongo:

```
from pymongo import MongoClient

# Conectando ao servidor MongoDB local
client = MongoClient('localhost', 27017)

# Acessando o banco de dados 'biblioteca' (ou criando-o se não existir)
db = client['biblioteca']

# Criando um índice simples no campo 'autor' da coleção 'livros'
db.livros.create_index([('autor', 1)])
```

Neste exemplo, estamos criando um índice simples no campo 'autor' da coleção 'livros', o que facilitará a busca por livros de um autor específico.

Outro exemplo é: Suponha que temos uma coleção de alunos e queremos buscar alunos por seus números de matrícula. Criar um índice simples no campo 'número de matrícula' nos permitirá encontrar alunos com base nesse critério de busca de forma mais eficiente.

Banco de Dados Não-Relacional

Tipos de Índices

Índices compostos

Índices compostos são criados em cima de múltiplos campos em uma coleção. Eles são úteis quando você precisa pesquisar em várias condições ao mesmo tempo.
Vamos ver um exemplo prático:

```
# Criando um índice composto nos campos 'autor' e 'ano publicacao'
db.livros.create_index([('autor', 1), ('ano_publicacao', 1)])
```

Aqui, estamos criando um índice composto nos campos 'autor' e 'ano_publicacao' da coleção 'livros', o que facilitará a busca por livros de um autor específico publicados em um determinado ano.

Ou imagine que temos uma coleção de produtos em um e-commerce e queremos buscar produtos por categoria e preço. Criar um índice composto nos campos 'categoria' e 'preço' nos permitirá realizar essa busca de forma mais eficiente.

Índices únicos

Os índices únicos garantem que não haja valores duplicados em um campo específico. Eles são úteis quando precisamos garantir a integridade dos dados e evitar duplicatas.
Vamos criar um índice único:

```
# Criando um índice único no campo 'isbn'
db.livros.create_index([('isbn', 1)], unique=True)
```

Aqui, estamos criando um índice único no campo 'isbn' da coleção 'livros', garantindo que nenhum livro tenha o mesmo número ISBN.

Suponha que estamos armazenando e-mails de usuários em uma coleção e queremos garantir que nenhum e-mail seja duplicado. Criar um índice único no campo 'e-mail' garantirá que cada e-mail seja único na coleção.

Índices parciais

Índices parciais são criados com base em uma condição específica, aplicada a todos os documentos em uma coleção. Eles são úteis quando você deseja indexar apenas uma parte dos documentos.
Vamos

```
# Criando um índice parcial nos documentos onde o
# campo 'status' é 'ativo'
db.livros.create_index(
      [('status', 1)],
      partialFilterExpression={'status': 'ativo'},
)
```

Neste exemplo, estamos criando um índice parcial nos documentos da coleção 'livros' onde o campo 'status' é 'ativo', o que pode ser útil se estivermos frequentemente consultando apenas os livros ativos.

Banco de Dados Não-Relacional

Otimização de Consultas

Estratégias de otimização

Quando lidamos com bancos de dados, é crucial entender como otimizar consultas para garantir um desempenho eficiente do sistema. Aqui estão algumas estratégias comuns de otimização de consultas:

Seleção de índices apropriados: Escolher os índices corretos para os campos frequentemente consultados pode melhorar significativamente o desempenho das consultas. É como usar o índice de um livro para encontrar rapidamente informações específicas.

Utilização de índices compostos: Índices compostos podem ser usados quando uma consulta envolve múltiplos campos. Isso pode reduzir o número de documentos que precisam ser examinados para encontrar os resultados desejados.

Evitar consultas excessivamente complexas: Consultas complexas, que envolvem muitas operações e filtros, podem ser mais lentas. Simplificar as consultas sempre que possível pode ajudar na otimização.

Análise de execução de consultas: Analisar o plano de execução das consultas pode ajudar a identificar gargalos de desempenho e otimizar consultas.

Suponha que temos uma coleção de livros em um banco de dados MongoDB e queremos otimizar consultas relacionadas a esses livros. Vamos começar conectando ao banco de dados e criando a coleção:

```
from pymongo import MongoClient

# Conectando ao servidor MongoDB local
client = MongoClient('localhost', 27017)

# Acessando o banco de dados 'biblioteca' (ou criando-o se não existir)
db = client['biblioteca']

# Criando uma coleção 'livros' (ou acessando-a se já existir)
livros = db['livros']
```

Utilização de índices em consultas

Os índices são como índices de um livro, que permitem que o banco de dados encontre rapidamente os registros desejados. Aqui estão algumas considerações ao utilizar índices em consultas:

Escolha do índice correto: Escolher o tipo correto de índice e os campos a serem indexados é crucial para otimizar as consultas. Índices simples, compostos, únicos e parciais são algumas opções disponíveis, e a escolha depende das necessidades específicas da aplicação.

Atualização regular dos índices: Os índices devem ser atualizados regularmente para refletir as alterações nos dados. Isso garante que os índices permaneçam eficazes e mantenham um desempenho consistente das consultas.

Banco de Dados Não-Relacional

Otimização de Consultas

Exemplos práticos de consultas com índices

Vamos explorar alguns exemplos práticos de consultas usando índices em um banco de dados MongoDB:

Exemplo 1: Consulta simples usando índice único

```
# Consulta por um livro usando o número ISBN
livro = livros.find_one({'isbn': '978-3-16-148410-0'})
print(livro)
```

Exemplo 2: Consulta composta usando índice composto

```
# Consulta por livros de um autor específico publicados em um determinado ano
livros_autor_ano = livros.find({autor': 'Machado de Assis', 'ano_publicacao': 1899})
for livro in livros_autor_ano:
    print(livro)
```

Esses exemplos ilustram como usar consultas com índices em um banco de dados MongoDB usando pymongo. É essencial entender como otimizar consultas e escolher os índices corretos para garantir um desempenho eficiente do sistema.

Banco de Dados Não-Relacional

Boas Práticas

Vamos falar sobre boas práticas no desenvolvimento de aplicações utilizando o MongoDB como banco de dados, e o PyMongo como biblioteca para interagir com o MongoDB em Python. Vamos abordar os seguintes subtopicos: Modelagem de dados eficiente, Gerenciamento de transações, Segurança e autenticação, e Monitoramento e otimização de desempenho.

Modelagem de dados eficiente

Na modelagem de dados eficiente, é importante projetar o esquema do seu banco de dados de forma a otimizar consultas e garantir uma boa performance. Aqui estão algumas práticas:

- **Análise de requisitos:** Antes de começar a modelagem, é essencial entender os requisitos da sua aplicação e como os dados serão acessados.
- **Modelagem denormalizada:** No MongoDB, muitas vezes é preferível denormalizar os dados, ou seja, incorporar dados relacionados em um único documento. Isso pode reduzir o número de consultas necessárias e melhorar o desempenho.
- **Uso de índices:** Índices são essenciais para melhorar a velocidade das consultas. Identifique os campos que serão frequentemente consultados e crie índices para eles.

Gerenciamento de transações

Transações garantem a consistência dos dados em operações que envolvem várias escritas ou leituras no banco de dados. Aqui estão algumas práticas:

Transações atômicas: No MongoDB, as transações são atômicas por padrão, o que significa que todas as operações em uma transação são aplicadas com sucesso ou nenhuma delas é aplicada.

Uso de sessões: Utilize sessões para agrupar operações em uma transação. Isso garante que todas as operações em uma transação sejam executadas na mesma sessão.

Exemplo em Python usando PyMongo:

```
# Iniciando uma sessão
with client.start session() as session:
    with session.start transaction():
        # Executar operações dentro da transação
```

Banco de Dados Não-Relacional

Boas Práticas

Segurança e autenticação

A segurança e autenticação são críticas para proteger os seus dados. Aqui estão algumas práticas:

- **Autenticação:** Utilize autenticação para controlar quem pode acessar o banco de dados. Crie usuários com permissões mínimas necessárias para cada aplicação.
- **Criptografia:** Utilize SSL/TLS para criptografar a comunicação entre o cliente e o servidor MongoDB.

Exemplo em Python usando PyMongo:

```
# Conectando ao MongoDB com autenticação
client = pymongo.MongoClient(
"""mongodb://username:password@localhost:27017/?authMechanism=SCRAM-SHA
-256""")
```

Monitoramento e otimização de desempenho

Monitorar o desempenho do seu sistema é essencial para identificar gargalos e otimizá-lo. Aqui estão algumas práticas:

- **Monitoramento de consultas:** Utilize ferramentas de monitoramento para identificar consultas lentas e otimizá-las, se necessário.
- **Monitoramento de uso de recursos:** Acompanhe o uso de CPU, memória e disco do servidor MongoDB para identificar possíveis problemas de desempenho.

Exemplo em Python usando PyMongo:

```
# Exibir todas as consultas em lento
db.setProfilingLevel(1)
db.system.profile.find().sort( { ts : -1 } )
```

Nesta aula, discutimos diversas boas práticas para o desenvolvimento de aplicações utilizando MongoDB e PyMongo. É essencial seguir essas práticas para garantir um sistema eficiente, seguro e de alto desempenho. Lembre-se de sempre analisar os requisitos da sua aplicação e adaptar as práticas conforme necessário. Se tiverem alguma dúvida, não hesitem em perguntar!

Encerramos por aqui nosso módulo de banco de dados, mas fiquem ligados que tem mais assuntos interessantes por vir!!

Obrigado pessoal!