Bancos não relacionais: SGBD's NoSQL Orientados a Documentos - MongoDB

QXD0099 - Desenvolvimento de Software para Persistência

Universidade Federal do Ceará - *Campus* Quixadá

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

Agenda

- Recapitulando o MongoDB
- Estrutura de Dados
- Relacionamentos no MongoDB
- Bibliotecas Python para MongoDB
  - PyMongo
  - MongoEngine
  - Motor
  - Beanie
  - ODMantic
  - Ming
  - Recomendações
- Características do PyMongo
- Conexão com Linguagens de Programação

Recapitulando o MongoDB

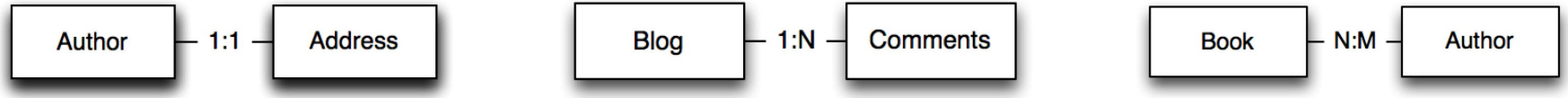
- MongoDB é um banco de dados NoSQL, orientado a documentos.
- Os dados são armazenados em documentos JSON/BSON.
- Não requer esquemas rígidos, sendo ideal para aplicações com estruturas de dados dinâmicas.

Estrutura de Dados

- **Banco de Dados:** Conjunto de coleções.
- **Coleções:** Conjunto de documentos (equivalente a tabelas no SQL).
- **Documento:** Estrutura JSON que contém pares chave-valor.

Relacionamentos no MongoDB

- Embora seja NoSQL, é possível criar relações entre documentos:
  - **1:1:** Um documento aponta para outro diretamente.
  - **1:N:** Um documento contém referências para vários outros.
  - **N:N:** Um documento contém várias referências de outro e vice versa.



1:1 (Um para Um)

- **Quando usar:** Dados estreitamente relacionados que raramente são acessados separadamente.

- **Implementação:** Incorporar (embed) o documento relacionado dentro do principal.

- **Exemplo:**
- Um usuário possui um perfil detalhado:

```
{
    "_id": "user123",
    "name": "John Doe",
    "profile": {
        "age": 30,
        "bio": "Software engineer"
    }
}
```

1:N (Um para Muitos)

- **Quando usar:** Um documento tem vários documentos relacionados.

- **Implementação:**
  - **Embed:** Relacionados armazenados diretamente como array no documento principal.
  - **Referência:** IDs dos relacionados armazenados e os dados mantidos em outra coleção.

- **Exemplo (Referência):**
- Um autor com vários livros:

```
// Autor
{
    "_id": "author123",
    "name": "Jane Smith",
    "books": ["book1", "book2"]
}

// Livro
{
    "_id": "book1",
    "title": "MongoDB Guide",
    "author_id": "author123"
}
```

N:N (Muitos para Muitos)

- **Quando usar:** Relacionamentos complexos entre entidades onde ambas podem ter múltiplos relacionamentos entre si.

- **Implementação:** Criar uma coleção intermediária para armazenar as associações.

- **Exemplo:**
- Estudantes inscritos em cursos

```
// Coleção intermediária
{
    "student_id": "student123",
    "course_id": "course456",
    "enrollment_date": "2025-01-01"
}
```

Bibliotecas Python para MongoDB

PyMongo, MongoEngine, Motor, Beanie, ODMantic e Ming

PyMongo

- PyMongo é a biblioteca oficial do MongoDB para Python, fornecendo uma interface de baixo nível para interagir diretamente com o banco de dados. Ele permite realizar operações CRUD, consultas complexas e manipulação de coleções de maneira direta, sem abstrações adicionais.

- **Características:**
  - Acesso direto à API do MongoDB.
  - Alta flexibilidade para trabalhar com esquemas dinâmicos.
  - Suporte completo para comandos MongoDB (CRUD, agregações, índices, etc.).
  - Integração com operações assíncronas por meio do motor (pymongo[motor]).

PyMongo - exemplo

```python
from pymongo import MongoClient

# Conectar ao MongoDB
client = MongoClient("mongodb://localhost:27017")
db = client["meu_banco"]
colecao = db["usuarios"]

# Criar e inserir um novo usuário
usuario = {"nome": "João", "idade": 25}
colecao.insert_one(usuario)

# Buscar usuários com idade maior ou igual a 20
usuarios = colecao.find({"idade": {"$gte": 20}})
for u in usuarios:
    print(u["nome"])
```

MongoEngine

- Um ODM (Object-Document Mapper) para MongoDB, similar a ORMs como SQLAlchemy no mundo SQL.
- Permite definir esquemas de dados como classes Python.

- **Características:**
  - Define modelos com tipos de campos (e.g., StringField, IntField).
  - Validação de dados embutida.
  - Suporte a relacionamentos entre documentos (embutidos ou referenciados).
  - Interface mais simples e estruturada para CRUD.

MongoEngine - exemplo

```python
from mongoengine import Document, StringField, IntField, connect

connect("meu_banco")

class Usuario(Document):
    nome = StringField(required=True)
    idade = IntField()

# Criar um novo usuário
usuario = Usuario(nome="João", idade=25)
usuario.save()

# Buscar usuários
usuarios = Usuario.objects(idade__gte=20)
for u in usuarios:
    print(u.nome)
```

Motor

- Uma biblioteca assíncrona para interagir com o MongoDB.
- Baseada no PyMongo, mas projetada para funcionar com asyncio.

- **Características:**
  - Suporte total a operações assíncronas no MongoDB.
  - Integrável com frameworks assíncronos como FastAPI e AIOHTTP.
  - Sintaxe muito parecida com PyMongo.

mongodb/**motor**

Motor - exemplo

```python
import motor.motor_asyncio
from fastapi import FastAPI

app = FastAPI()
client = motor.motor_asyncio.AsyncIOMotorClient("mongodb://localhost:27017")
db = client.meu_banco

@app.get("/usuarios")
async def get_usuarios():
    usuarios = await db.usuarios.find().to_list(100)
    return usuarios
```

Beanie

- Um ODM assíncrono para MongoDB, construído sobre Motor e Pydantic.
- Focado em facilidade de uso e integração com FastAPI.

- **Características:**
  - Utiliza modelos Pydantic para definir documentos.
  - Suporte a operações assíncronas.
  - Pipeline de agregação e transações.
  - Migrações de esquemas.

Beanie - exemplo

```python
from beanie import Document, init_beanie
from motor.motor_asyncio import AsyncIOMotorClient
from pydantic import BaseModel

class Usuario(Document):
    nome: str
    idade: int

async def init():
    client = AsyncIOMotorClient("mongodb://localhost:27017")
    await init_beanie(database=client.meu_banco, document_models=[Usuario])

# Operações
novo_usuario = Usuario(nome="Maria", idade=30)
await novo_usuario.insert()

usuarios = await Usuario.find_all().to_list()
```

ODMantic

- Outro ODM assíncrono baseado em Motor e Pydantic.
- Focado em simplicidade e integração com FastAPI.

- **Características:**
  - Modelos definidos com classes Pydantic.
  - Operações assíncronas com sintaxe limpa.
  - Suporte a transações e agregações.

art049/**odmantic**

ODMantic - exemplo

```python
from odmantic import AIOEngine, Model

class Usuario(Model):
    nome: str
    idade: int

engine = AIOEngine()

async def criar_usuario():
    usuario = Usuario(nome="Carlos", idade=35)
    await engine.save(usuario)

async def listar_usuarios():
    usuarios = await engine.find(Usuario, Usuario.idade > 30)
    return usuarios
```

Ming

- Um ODM semelhante ao MongoEngine, mas mais minimalista.
- Permite trabalhar com schemas definidos por classes Python.

- **Características:**
  - Suporte a validação de esquema.
  - Suporte a documentos embutidos e referenciados.

- **Vantagens:**
  - Simples e direto.
  - Boa opção para projetos pequenos.

- **Desvantagens:**
  - Menos popular e com suporte limitado em comparação ao MongoEngine.

Ming ODM: Classes and Collections

ODMantic - exemplo

```python
from ming import create_datastore, Document
from ming.fields import StringField, IntField

bind = create_datastore("mongodb://localhost:27017/meu_banco")

class Usuario(Document):
    class __mongometa__:
        session = bind
        name = "usuarios"

    nome = StringField()
    idade = IntField()

Usuario(nome="João", idade=25).m.save()
```

Recomendações

- **Para projetos pequenos:** MongoEngine ou PyMongo.
- **Para projetos assíncronos com FastAPI:** Beanie ou ODMantic.
- **Para controle total:** PyMongo ou Motor.
- Escolha a biblioteca com base na complexidade do projeto, familiaridade com programação assíncrona e necessidade de validação automática.

Características do PyMongo

Conexão e Gerenciamento de Conexão, CRUD, Suporte para Consultas Flexíveis, Operações em Lote, Transações, Suporte para Indexação, Manipulação de ObjectId, Autenticação e Segurança e Flexibilidade com Estruturas JSON/BSON

Conexão e Gerenciamento de Conexão

- Facilita a conexão a um servidor ou cluster MongoDB.
- Suporte nativo para Replica Sets e Sharding, permitindo maior confiabilidade e escalabilidade.
- Gerenciamento automático de pools de conexões.

```
from pymongo import MongoClient

client = MongoClient("mongodb://localhost:27017")
db = client.meu_banco
```

CRUD (Create, Read, Update, Delete)

- Permite realizar operações de CRUD diretamente nas coleções do banco

```
# Inserir um documento
db.minha_colecao.insert_one({"nome": "João", "idade": 25})

#################################################################

# Buscar todos os documentos
documentos = db.minha_colecao.find()
for doc in documentos:
    print(doc)
```

Suporte para Consultas Flexíveis

- Permite consultas complexas usando filtros, operadores lógicos e expressões regulares.

- Suporte para operadores MongoDB, como $gt, $lt, $in, $or, etc.

```
# Buscar documentos com idade maior que 20
resultados = db.minha_colecao.find({"idade": {"$gt": 20}})
```

Operações em Lote

- Suporte para inserções, atualizações e exclusões em lote para maior eficiência.

```
db.minha_colecao.insert_many([
    {"nome": "Maria", "idade": 30},
    {"nome": "Carlos", "idade": 40}
])
```

Transações

- Suporte para transações em ambientes de Replica Sets e clusters sharded.
- Garante consistência em operações que envolvem várias coleções ou documentos

```
with client.start_session() as session:
    with session.start_transaction():
        db.minha_colecao.insert_one({"nome": "Ana"}, session=session)
        db.outra_colecao.insert_one({"produto": "Livro"}, session=session)
```

Suporte para Indexação

- Criação de índices para melhorar a performance das consultas.

```
db.minha_colecao.create_index("nome")
```

Manipulação de ObjectId

- O _id padrão no MongoDB é do tipo ObjectId. O PyMongo fornece métodos para criar e manipular esses IDs.

```
from bson import ObjectId

doc_id = ObjectId("64d1f7c9b2a1b23e4556abcd")
documento = db.minha_colecao.find_one({"_id": doc_id})
```

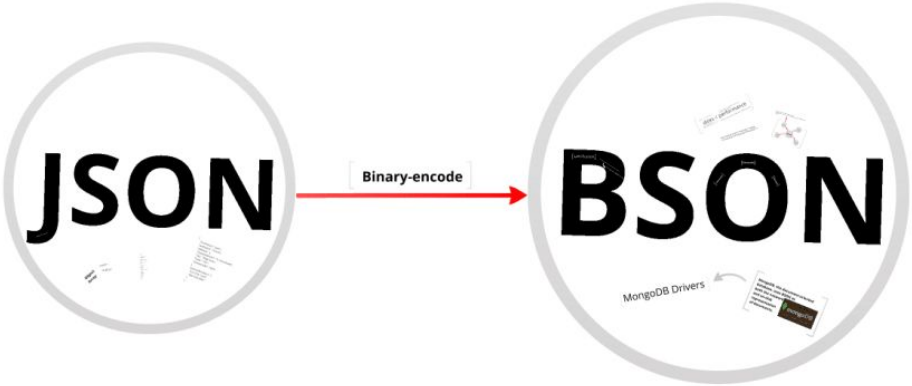
Autenticação e Segurança

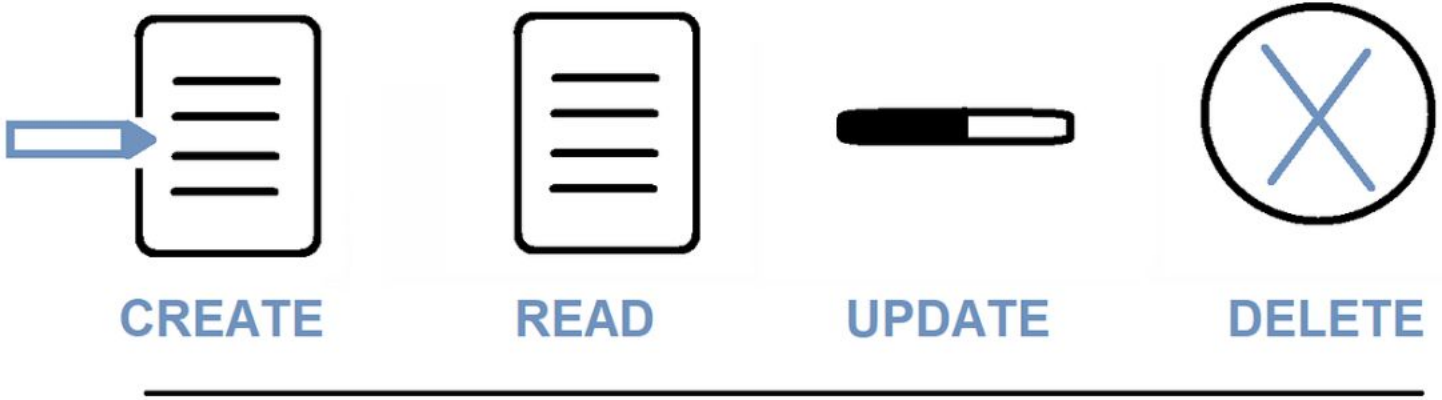
- Suporte para autenticação com usuários e senhas.
- Permite conexão usando SSL/TLS para maior segurança.

```
client = MongoClient("mongodb://user:password@localhost:27017/?authSource=admin")
```

Flexibilidade com Estruturas JSON/BSON

- Os dados no MongoDB são armazenados no formato BSON (Binary JSON), e o PyMongo converte automaticamente entre JSON (Python dict) e BSON.





C R U D

FastAPI python™ mongoDB®


LET'S
CODE

Referências

- Curso completo de FastAPI por Eduardo Mendes
  - https://fastapidozero.dunossauro.com/
  - https://github.com/dunossauro/fastapi-do-zero
  - Playlist no YouTube
- https://www.mongodb.com/docs/
- MongoDB: https://www.mongodb.com/pt-br
- FastAPI - https://fastapi.tiangolo.com/
- Pydantic - https://pydantic.dev/
- SQLAlchemy - https://www.sqlalchemy.org/
- SQLModel - https://sqlmodel.tiangolo.com

Obrigado!

Dúvidas?

Universidade Federal do Ceará - *Campus* Quixadá

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