Universidade Federal do Ceará
Campus Quixadá
QXD0099 - Desenvolvimento de Software para Persistência
Prof. Francisco Victor da Silva Pinheiro

Trabalho Prático 2 - Mapeamento Objeto-Relacional com SQLAlchemy e SQLModel

Descrição Geral do Trabalho

Neste trabalho, você desenvolverá uma API para gerenciar um domínio específico utilizando Mapeamento Objeto-Relacional (ORM). O objetivo é aplicar os conceitos de ORM com SQLAlchemy e SQLModel, empregando um banco de dados relacional à sua escolha (PostgreSQL, SQLite, MySQL ou MariaDB). A aplicação deve oferecer funcionalidades CRUD completas, suporte à paginação e migrações de banco de dados com Alembic. Além disso, serão incluídas boas práticas de performance e logs para monitoramento das operações.

Objetivos Específicos

- Entender os conceitos de Mapeamento Objeto-Relacional.
- Desenvolver modelos e funcionalidades CRUD usando SQLAlchemy ou SQLModel.
- Implementar relacionamentos entre entidades e suporte à paginação.
- Realizar migrações de esquema utilizando Alembic.
- Configurar logs para monitorar e otimizar o desempenho da aplicação.
- Aplicar boas práticas de organização e desenvolvimento em Python.

Cada dupla trabalhará com um tema diferente, contendo pelo menos 5 entidades relacionadas.

Passo a Passo do Trabalho

1. Definir uma entidade e criar uma classe python

- **Objetivo:** Escolher um tema, onde cada tema possui pelo menos 5 entidades (um "objeto" ou "coisa" representativa) que faça sentido no contexto do domínio escolhido no TP2. Essas entidades devem ter **pelo menos (5) cinco atributos**.
- **Exemplo:** Se o domínio for "vendas", vocês poderiam definir uma entidade chamada **Produto** com atributos como **id, nome, categoria, preco, data_criacao, descriminacao,** e **estoque**.
- **Implementação:** Você deve implementar uma classe Python para representar essa entidade usando a biblioteca **pydantic**, que ajudará a garantir que os dados fornecidos estão no formato esperado.

2. Introdução ao ORM e Configuração do Banco de Dados

- Escolher um banco de dados entre PostgreSQL, SQLite, MySQL ou MariaDB.
- Configurar a conexão ao banco escolhido no projeto.
- Criar as tabelas relacionadas ao tema da dupla, com pelo menos 5 entidades interligadas.
- As entidades devem possuir pelo menos um de cada relacionamentos desses:
    - 1:1
    - 1:N
    - N:N

3. Criar uma API REST com FastAPI

Usando FastAPI, vocês vão criar endpoints para implementar cada funcionalidade solicitada. Cada funcionalidade será implementada em um endpoint específico. Abaixo estão os detalhes de cada funcionalidade que a API deverá oferecer.

F1. Inserir uma entidade no banco de dados

- **Objetivo**: Implementar um endpoint para cadastrar uma nova entidade no sistema.
- **Detalhes**: Quando o endpoint for acessado com um JSON contendo os dados da entidade, a API deverá adicionar essa entidade ao banco de dados escolhido (PostgreSQL, SQLite, MySQL ou MariaDB).
    - **Exemplo**: Enviar um JSON com os dados de um novo produto, e a API salvará esses dados no banco, onde cada registro representará um produto diferente.

F2. Listar todas as entidades do banco

- **Objetivo**: Implementar um endpoint para retornar todos os registros da entidade cadastrada.
- **Detalhes**: O endpoint deverá consultar o banco de dados e retornar todas as entidades cadastradas no formato JSON. Isso permitirá visualizar todos os dados registrados até o momento.
    - **Exemplo**: Se houver três produtos cadastrados no banco, ao acessar esse endpoint, o usuário verá um JSON contendo os três produtos.

F3. CRUD completo da entidade

- **Objetivo**: Implementar endpoints para as operações de Create (criar), Read (ler), Update (atualizar) e Delete (excluir) para a entidade.
  Detalhes: O CRUD permite que o usuário realize todas as operações fundamentais para gerenciar uma entidade. Em todos os casos, o banco de dados deverá ser atualizado conforme as mudanças. As operações incluem:
    - **Criar**: Já implementado na funcionalidade F1.
    - **Ler**: Listagem de todos os registros, conforme F2.
    - **Atualizar**: Modificar um registro específico no banco, utilizando o identificador único (como o ID).

    - **Excluir**: Remover um registro específico no banco, também utilizando o identificador único.
        - **Exemplo**: O usuário poderá modificar o nome de um produto ou excluir um produto específico.

F4. Mostrar a quantidade de entidades

- **Objetivo**: Implementar um endpoint para mostrar a quantidade total de entidades cadastradas.
- **Detalhes**: Este endpoint deve contar o número de registros no banco e retornar essa contagem ao usuário. Importante: deve refletir a contagem real, mesmo que o banco seja atualizado externamente (fora do sistema).
    - **Exemplo**: Se há 10 produtos cadastrados no banco, o endpoint deve retornar { "quantidade": 10 }.

F5. Implementar paginação e limitação de resultados

- **Objetivo**: Implementar um endpoint para retornar os registros de forma paginada.
- **Detalhes**: A API deve aceitar parâmetros como page e limit para retornar apenas uma parte dos registros por vez, permitindo a navegação entre páginas de resultados.
    - **Exemplo**: /produtos?page=2&limit=5 retornará os produtos da página 2, com um limite de 5 produtos por página.

F6. Filtrar entidades por atributos específicos

- **Objetivo**: Implementar um endpoint para filtrar as entidades cadastradas com base em atributos específicos.
- **Detalhes**: O usuário pode fornecer parâmetros de filtragem, como nome, categoria ou intervalo de preços. A API deve retornar apenas as entidades que atenderem aos critérios definidos.
    - **Exemplo**: Se a entidade Produto tiver um atributo categoria, o usuário poderá filtrar para ver apenas os produtos da categoria "Eletrônicos".

F7. Criar migrações com Alembic

- **Objetivo**: Implementar um sistema de migração para gerenciar alterações no esquema do banco de dados.
- **Detalhes**: As migrações devem ser configuradas com o Alembic, permitindo criar e alterar tabelas de forma controlada. Devem ser realizadas pelo menos duas migrações:
    - Migração inicial para criar o esquema do banco.
    - Migração adicional para alterar ou adicionar um campo em uma das tabelas.

F8. Configurar logs para monitoramento

- **Objetivo**: Implementar um sistema de logging para registrar as operações realizadas na API.

- **Detalhes**: Cada operação executada na API, como inserções, exclusões ou falhas, deve ser registrada em um arquivo de log para fins de auditoria e monitoramento.
    - **Exemplo**: Quando um novo produto for adicionado, o sistema deve registrar uma mensagem como "Produto inserido com sucesso" com a data e hora da operação.

Estrutura do projeto

- Organização modular com separação clara de responsabilidades:
    - Models (entidades)
    - Rotas (endpoints)
    - Configurações em arquivo(s) externo(s) à aplicação (.env)
    - etc.

Consultas Requeridas

- Consultas por ID
- Listagens filtradas por relacionamentos
- Buscas por texto parcial
- Filtros por data/ano
- Agregações e contagens
- Classificações e ordenações
- Consultas complexas envolvendo múltiplas entidades

Pontos importantes

- É necessário gravar um vídeo de, no máximo, 5 minutos, demonstrando o funcionamento do trabalho e explicando, de forma sucinta e clara, algum aspecto relevante da implementação.

- Além disso, deve ser elaborado um relatório descrevendo as atividades realizadas por cada membro da dupla.

- **Observação:** Se o trabalho for realizado individualmente, o relatório deve detalhar as atividades realizadas, bem como apontar o que não foi concluído, se aplicável.

"A melhor maneira de predizer o futuro é inventá-lo."
Alan Kay