Mapeamento Objeto-Relacional: Paginação e Limitação de Resultados

QXD0099 - Desenvolvimento de Software para Persistência

Universidade Federal do Ceará - *Campus* Quixadá

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

Agenda

- Paginação com SQLModel
- Retornando Informações de Paginação
- Integração com FastAPI
- Benefícios de Paginação e Limitação
- Paginação Dinâmica

Paginação com SQLModel

- A paginação consiste em dividir um conjunto de resultados em "páginas" e retornar apenas uma parte específica dos dados em cada consulta. Isso é feito utilizando os parâmetros SQL de OFFSET e LIMIT.

- **Estratégia**
  - **LIMIT:** Define o número máximo de resultados retornados pela consulta (tamanho da página).
  - **OFFSET:** Define o ponto de partida da consulta (baseado no número de resultados já exibidos).

Paginação com SQLModel

- Suponha que existe uma tabela Membro e você quer implementar a paginação.

```python
from sqlmodel import Session, select
from database import engine  # Assume que o `engine` foi definido

def get_membros_paginated(page: int, page_size: int):
    """
    Retorna uma lista de membros paginada.
    - page: número da página (1, 2, 3, ...)
    - page_size: número de resultados por página
    """
    offset = (page - 1) * page_size  # Cálculo para começar do item correto

    with Session(engine) as session:
        # Consulta com OFFSET e LIMIT
        query = select(Membro).offset(offset).limit(page_size)
        results = session.exec(query).all()
    return results
```

- **`page`**: Número da página a ser retornada.
- **`page_size`**: Quantidade de resultados por página.
- **`offset`**: Calculado como `(page - 1) * page_size`, garantindo que as páginas sejam indexadas corretamente.

Retornando Informações de Paginação

- Além dos dados, é útil retornar informações como o total de resultados e o número total de páginas.

```python
def get_paginated_data(page: int, page_size: int):
    with Session(engine) as session:
        # Obter o total de registros
        total_records = session.exec(select(Membro).count()).one()

        # Calcular o total de páginas
        total_pages = (total_records + page_size - 1) // page_size

        # Obter os registros da página atual
        offset = (page - 1) * page_size
        query = select(Membro).offset(offset).limit(page_size)
        results = session.exec(query).all()

    return {
        "data": results,
        "total_records": total_records,
        "total_pages": total_pages,
        "current_page": page
    }
```

- **Recebe dois parâmetros**:
  - page: número da página atual.
  - page_size: quantidade de registros por página.
- **Cria uma sessão do banco de dados**:
  - Utiliza Session(engine) para realizar as consultas.
- **Calcula o total de registros**:
  - Executa uma consulta para contar todos os registros na tabela Membro.
- **Determina o total de páginas**:
  - Usa a fórmula (total_records + page_size - 1) // page_size para calcular o número de páginas necessárias.
- **Busca os registros da página atual**:
  - Calcula o offset com (page - 1) * page_size para determinar o ponto inicial.
  - Utiliza offset e limit para retornar apenas os registros da página atual.
- **Retorna um dicionário com os dados**:
  - Inclui os registros da página atual (data), total de registros (total_records), total de páginas (total_pages) e o número da página atual (current_page).

Integração com FastAPI

```python
from fastapi import FastAPI, Query
from sqlmodel import Session, select
from database import engine

app = FastAPI()

@app.get("/membros/")
def get_membros(
    page: int = Query(1, ge=1),  # Número da página, padrão 1, mínimo 1
    page_size: int = Query(10, ge=1, le=100)  # Tamanho da página, entre 1 e 100
):
    with Session(engine) as session:
        total_records = session.exec(select(Membro).count()).one()
        total_pages = (total_records + page_size - 1) // page_size
        offset = (page - 1) * page_size

        query = select(Membro).offset(offset).limit(page_size)
        results = session.exec(query).all()

    return {
        "data": results,
        "pagination": {
            "total_records": total_records,
            "total_pages": total_pages,
            "current_page": page,
            "page_size": page_size
        }
    }
```

Parâmetros da Rota:

- page: Número da página (padrão é 1, mínimo permitido é 1).
- page_size: Quantidade de registros por página (padrão é 10, valores permitidos entre 1 e 100).

Dentro da função get_membros:

1. **Sessão de banco de dados**:
   - Cria uma sessão com Session(engine) para executar consultas.
2. **Total de registros**:
   - select(Membro).count(): Conta o número total de registros na tabela Membro.
3. **Total de páginas**:
   - Calculado com (total_records + page_size - 1) // page_size.
4. **Deslocamento (offset)**:
   - Determina o índice inicial dos registros da página atual: (page - 1) * page_size.
5. **Consulta paginada**:
   - Seleciona os registros da página atual com .offset(offset).limit(page_size).

Retorno da Rota

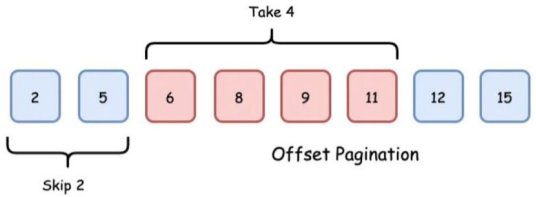
```
{
  "data": [
    {"id": 1, "nome": "João"},
    {"id": 2, "nome": "Maria"}
  ],
  "pagination": {
    "total_records": 50,
    "total_pages": 5,
    "current_page": 1,
    "page_size": 10
  }
}
```

A rota retorna um dicionário com:

- **data**: Lista dos membros na página atual.
- **pagination**: Informações sobre a paginação:
  - total_records: Total de registros no banco.
  - total_pages: Total de páginas disponíveis.
  - current_page: Página atual.
  - page_size: Quantidade de registros por página.

Benefícios de Paginação e Limitação

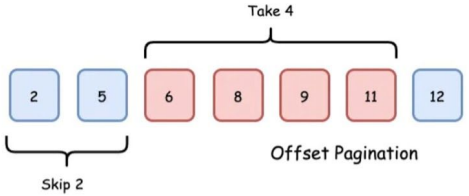
- **Desempenho:**
  - Evita carregar todos os dados de uma vez.
  - Melhora o tempo de resposta do servidor.



- **Facilidade de uso:**
  - Os usuários podem navegar pelos resultados sem serem sobrecarregados.

- **Escalabilidade:**
  - Suporta grandes conjuntos de dados sem esgotar a memória.

Paginação Dinâmica



- Para uma experiência mais fluida, pode-se implementar um mecanismo de "carregamento infinito" (scroll infinito), que utiliza cursor-based pagination.

- **Cursor-Based Pagination com SQLModel**
  - Cursor-based pagination usa um identificador único (como o id) para buscar registros subsequentes.

Paginação Dinâmica

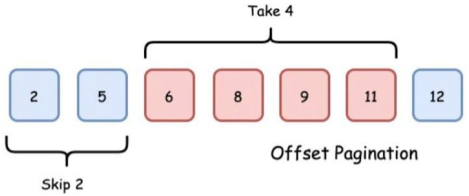
```python
def get_membros_cursor(last_id: Optional[int], page_size: int):
    with Session(engine) as session:
        if last_id:
            query = select(Membro).where(Membro.id > last_id).limit(page_size)
        else:
            query = select(Membro).limit(page_size)

        results = session.exec(query).all()
    return results
```

A função get_membros_cursor implementa **paginação baseada em cursor** para buscar registros da tabela Membro. Ela:

1. **Recebe** um last_id (último ID recuperado) e um page_size (tamanho da página).
2. **Consulta** os registros com ID maior que last_id (se fornecido) ou pega os primeiros page_size registros.
3. **Retorna** uma lista com os resultados.

Paginação com SQLModel



- A paginação com SQLModel é flexível e pode ser adaptada para atender diferentes cenários de exibição de dados.

- Você pode implementar tanto offset-based pagination (com LIMIT e OFFSET) quanto cursor-based pagination, dependendo das necessidades da aplicação e do volume de dados.

- A integração com FastAPI torna o processo ainda mais eficiente e amigável.


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

- FastAPI - https://fastapi.tiangolo.com/
- Pydantic - https://pydantic.dev/
- SQLAlchemy - https://www.sqlalchemy.org/
- SQLModel - https://sqlmodel.tiangolo.com

- https://docs.github.com/pt/rest/using-the-rest-api/using-pagination-in-the-rest-api?apiVersion=2022-11-28

Obrigado!

Dúvidas?

Universidade Federal do Ceará - *Campus* Quixadá

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