



AULA 04

MANIPULAÇÃO DOS DADOS

O QUE VEREMOS HOJE

01 MANIPULAÇÃO DOS DADOS

02 READ

03 CREATE

04 UPDATE

05 DELETE

06 RELACIONAMENTO ENTRE TABELAS

MANIPULAÇÃO DE DADOS

Em um banco de dados, manipulação significa inserir, consultar, atualizar ou remover registros armazenados.

Essas operações são conhecidas como CRUD:

1. Create – Inserir novos registros (INSERT)
2. Read – Consultar dados existentes (SELECT)
3. Update – Alterar registros já salvos (UPDATE)
4. Delete – Remover registros (DELETE)

No SQLAlchemy, essas operações são feitas usando métodos Python em vez de escrever comandos SQL diretamente.



READ – CONSULTAR DADOS (GET)

A operação Read é usada para buscar informações já salvas no banco de dados.

No SQL puro, usamos o comando SELECT. No SQLAlchemy, fazemos isso através dos métodos da classe do modelo.

Formas comuns de consulta:

- `Model.query.all()` → retorna todos os registros
- `Model.query.get(id)` → retorna um registro pelo ID
- `Model.query.filter_by(campo=valor).all()` → retorna registros filtrados por um campo



```
1  # Buscar todos os produtos
2  produtos = Produto.query.all()
3
4  # Buscar um produto pelo ID
5  produto = Produto.query.get(1)
6
7  # Buscar produtos com preço maior que um valor específico
8  produtos_filtrados = Produto.query.filter(Produto.preco > 100).all()
9
10
```

CREATE – INSERIR DADOS (POST)

A operação Create é usada para adicionar novos registros ao banco de dados.

No SQL puro, usamos o comando INSERT. No SQLAlchemy, fazemos isso criando uma instância do modelo e adicionando-a na sessão, confirmando depois com **commit()**.

Formas comuns de inserção:

- Criar um objeto do modelo com os dados desejados.
- Adicionar o objeto à sessão com `db.session.add(objeto)`.
- Confirmar a gravação no banco com `db.session.commit()`.

```
1 # Criar um novo registro
2 novo_registro = Produto(nome="Caneta", preco=2.50, estoque=100)
3
4 # Adicionar à sessão
5 db.session.add(novo_registro)
6
7 # Confirmar no banco
8 db.session.commit()
```

UPDATE – ATUALIZAR DADOS (PUT)

A operação Update é usada para alterar registros existentes no banco de dados.

No SQL puro, usamos o comando UPDATE. No SQLAlchemy, primeiro buscamos o objeto, depois alteramos seus atributos e confirmamos com **commit()**.

Etapas

- Localizar o registro no banco (ex.: `Model.query.get(id)`).
- Modificar os atributos desejados.
- Confirmar a alteração com `db.session.commit()`.

```
1  # Buscar um registro pelo ID
2  produto = Produto.query.get(1)
3
4  # Alterar atributos
5  produto.preco = 3.00
6  produto.estoque = 200
7
8  # Confirmar no banco
9  db.session.commit()
10
```


DELETE – REMOVER DADOS (DELETE)

A operação Delete é usada para remover registros existentes do banco de dados.

No SQL puro, usamos o comando DELETE. No SQLAlchemy, primeiro buscamos o objeto, depois usamos **db.session.delete(objeto)** e confirmamos com **commit()**.

Etapas:

- Localizar o registro no banco (Model.query.get(id)).
- Remover o objeto com db.session.delete().
- Confirmar a exclusão com db.session.commit().



```
1 # Buscar um registro pelo ID
2 produto = Produto.query.get(1)
3
4 # Remover o registro
5 db.session.delete(produto)
6
7 # Confirmar no banco
8 db.session.commit()
9
```

RELACIONAMENTO ENTRE TABELAS

Em bancos de dados relacionais, tabelas podem se conectar para representar informações ligadas entre si. Essa ligação é feita com **chaves estrangeiras (Foreign Keys)**.

Como funciona:

- Uma chave estrangeira aponta para a chave primária de outra tabela.
- No SQLAlchemy:
 - `db.ForeignKey('tabela.campo')` → cria a ligação.
 - `db.relationship()` → facilita o acesso entre os registros.



```
1 class TabelaA(db.Model):
2     id = db.Column(db.Integer, primary_key=True)
3
4 class TabelaB(db.Model):
5     id = db.Column(db.Integer, primary_key=True)
6     tabela_a_id = db.Column(db.Integer, db.ForeignKey('tabela_a.id'))
7
```


ATIVIDADE PRÁTICA

Com base nos modelos de usuários e tarefas (ligadas por user_id como chave estrangeira), implemente a seguinte rota:

Rota GET /users/<id>

- Busque o usuário pelo id informado.
- Se o usuário não existir, retorne uma mensagem de erro com status 404.
- Se existir, retorne:
 - id, nome, email do usuário.
 - Lista de tarefas associadas a ele (cada tarefa com id, titulo, descricao, status, user_id).
- Retorne a resposta com status 200 no sucesso.

ATIVIDADE PRÁTICA

Com base nos modelos de usuários e tarefas (ligadas por user_id como chave estrangeira), implemente a seguinte rota:

Rota DELETE /users/<id>

- Busque o usuário pelo id informado.
- Se o usuário não existir, retorne:
 - Mensagem de erro com status 404.
- Verifique se o usuário possui tarefas associadas:
 - Se possuir, não exclua e retorne:
 - Mensagem de erro informando que o usuário possui tarefas vinculadas.
 - Status 400.
- Caso não possua tarefas:
 - Remova o usuário do banco.
 - Retorne uma mensagem de confirmação.
 - Status 200 no sucesso.

PRÓXIMOS PASSOS NO FLASK

Variáveis de Ambiente

- Armazenar configurações sensíveis (senhas, chaves de API) de forma segura. Ex.: .env, python-dotenv

Validação de Dados

- Garantir que entradas sejam válidas antes de salvar no banco. Ex.: marshmallow, Flask-WTF, pydantic

Blueprints e Estrutura de Projeto

- Organizar rotas, models e configs em módulos separados. Ex.: Blueprint

Integração com Front-end

- Criar páginas HTML e integrar com frameworks como Bootstrap ou até React/Vue. Ex.: render_template, Jinja2

Autenticação e Autorização

- Criar login, logout e proteger rotas. Ex.: Flask-Login, JWT

Testes Automatizados

- Escrever testes para garantir que a API continua funcionando após mudanças. Ex.: pytest, Flask-Testing