Modelo Relacional: Revisão de SQL com PostgreSQL

QXD0099 - Desenvolvimento de Software para Persistência

Universidade Federal do Ceará - *Campus* Quixadá

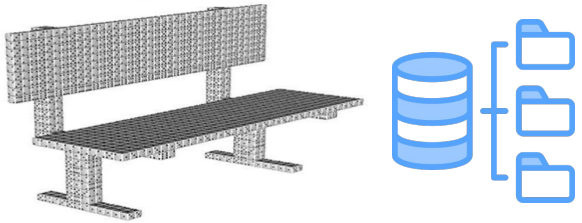
Prof. Francisco Victor da Silva Pinheiro
victorpinheiro@ufc.br

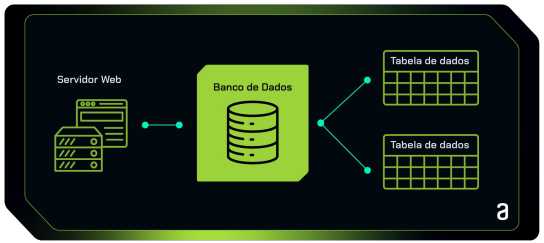
Agenda

- Introdução aos bancos de dados
- Bancos de dados relacionais
  - PostgreSQL, SQLite, MariaDB e MySQL
- Elementos fundamentais
- Vantagens dos bancos de dados relacionais
- Normalização
- SQL - Structured Query Language
  - Componentes principais da SQL
  - Criação e manipulação de tabelas
  - Consultas básicas
- Funcionalidades avançadas em SQL
  - Joins e subconsultas
  - Funções agregadas e janelas
  - Transações e segurança
- PEP (Python Enhancement Proposal)
  - DB-API do Python – PEP 249

Introdução aos bancos de dados

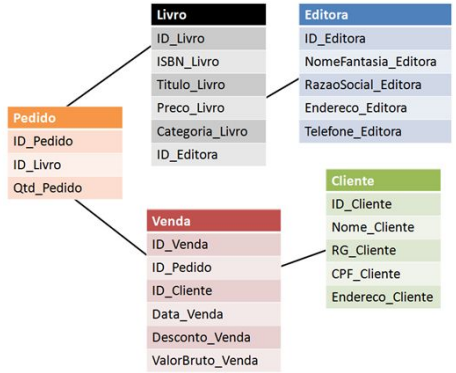
- Um banco de dados é um sistema organizado para armazenar, gerenciar e recuperar informações de maneira eficiente.
- Permite o gerenciamento de dados estruturados para facilitar consultas, atualizações e garantir a integridade.
- Os bancos de dados são controlados por um sistema de gerenciamento de banco de dados (SGBDS). Juntos, os dados, o SGBDS e os aplicativos associados são chamados de sistema de banco de dados.





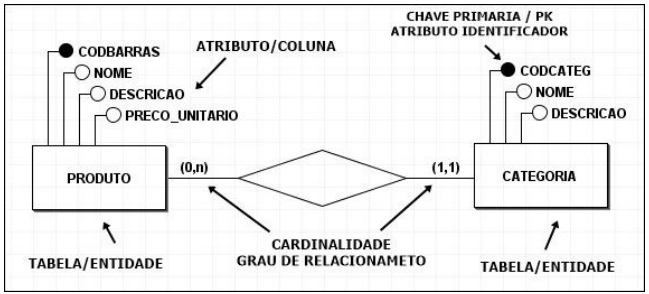
Bancos de dados relacionais

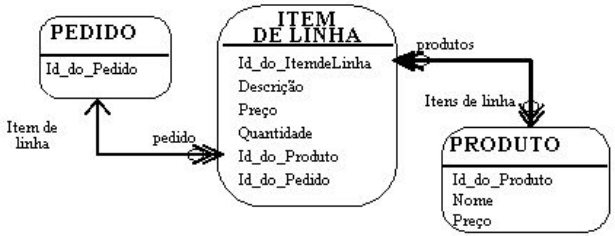
- Baseiam-se no modelo relacional, onde os dados são organizados em relações (tabelas).
- **Cada tabela contém:**
  - **Linhas (Tuplas):** representam instâncias individuais de dados.
  - **Colunas (Atributos):** definem os tipos de dados armazenados.
- Relacionamentos entre as tabelas são estabelecidas por meio de chaves primárias e chaves estrangeiras.



Elementos Fundamentais

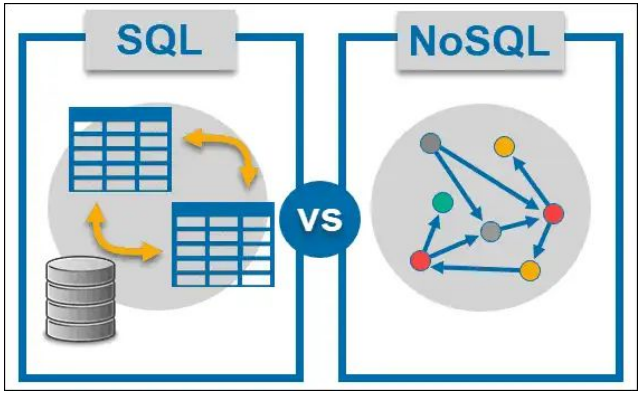
- Tabelas
  - Estruturas principais que contêm os dados.
- Colunas
  - Atributos ou propriedades dos dados.
- Índices
  - Usados para acelerar consultas e ordenações.
- Chaves Primárias
  - Identificadores únicos para registros em uma tabela.
- Chaves Estrangeiras
  - Relacionam uma tabela a outra

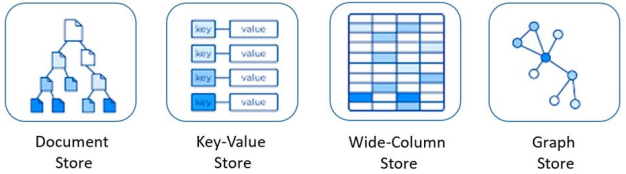




Vantagens dos Bancos de Dados Relacionais

- Redução de redundância
  - Evita a repetição desnecessária de informações.
- Manutenção da integridade
  - Garante consistência entre os dados.
- Flexibilidade
  - Permite consultas complexas e filtragem de informações.





Normalização

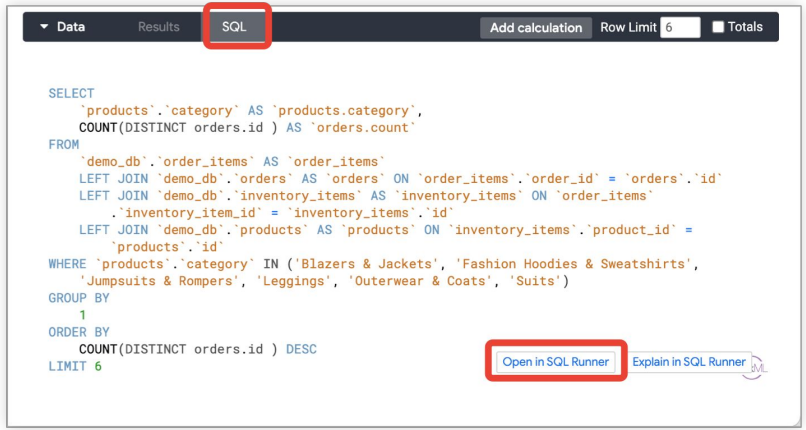
- Processo que organiza os dados para minimizar redundâncias e dependências.
- Divide tabelas maiores em tabelas menores relacionadas.
- **Objetivo:** melhorar a eficiência e evitar problemas como anomalias de inserção, atualização e exclusão.

QUADRO ESQUEMATIZADO DE NORMALIZAÇÃO	
1ª. Forma normal	Atributos atômicos, indivisíveis.
2ª. Forma normal	Ausência de dependências parciais.
3ª. Forma normal	Ausência de dependências transitivas.
Forma normal de Boyce-Codd	Ausência de dependências entre os atributos não chave.
4ª. Forma normal	Ausência de dependências multivaloradas.
5ª. Forma normal	Ausência de dependências de junção.

SQL – Structured Query Language

- Linguagem declarativa projetada especificamente para gerenciar e manipular dados em sistemas de bancos de dados relacionais.

- Padronizada e amplamente adotada por diversos sistemas de gerenciamento de banco de dados (SGBDs) como SQLite, MySQL, PostgreSQL, Oracle e SQL Server.



Componentes principais da SQL

- **DDL (Data Definition Language): Linguagem de Definição de Dados.**
  - CREATE TABLE: cria uma nova tabela.
  - ALTER TABLE: modifica a estrutura de uma tabela existente.
  - DROP TABLE: exclui uma tabela.

- **DML (Data Manipulation Language): Linguagem de Manipulação de Dados.**
  - INSERT INTO: adiciona novos registros em uma tabela.
  - UPDATE: atualiza dados existentes.
  - DELETE: remove registros específicos.

Componentes principais da SQL

- **DQL (Data Query Language): Linguagem de Consulta de Dados.**
  - SELECT: utilizado para consultar dados em tabelas.
- **DCL (Data Control Language): Linguagem de Controle de Dados.**
  - GRANT: concede permissões a usuários.
  - REVOKE: remove permissões.
- **TCL (Transaction Control Language): Linguagem de Controle de Transações.**
  - COMMIT: aplica definitivamente as alterações realizadas.
  - ROLLBACK: reverte alterações em caso de erros.

Criação e manipulação de tabelas

- Comando **CREATE TABLE**

```sql
CREATE TABLE Pessoa (
    id INT AUTO_INCREMENT PRIMARY KEY,
    nome VARCHAR(100) NOT NULL,
    email VARCHAR(100),
    cpf CHAR(11) UNIQUE
);
```

- Alteração de tabelas com **ALTER TABLE**

```sql
ALTER TABLE Pessoa ADD telefone VARCHAR(15);

ALTER TABLE Pessoa ADD endereco VARCHAR(15);

ALTER TABLE Pessoa ADD sobrenome VARCHAR(15);
```

- Remoção de tabelas com **DROP TABLE**

```sql
DROP TABLE Pessoa;
```

Consultas básicas

- Inserindo dados

```
INSERT INTO Pessoa (nome, email, cpf)
VALUES ('João Silva', 'joao@email.com',
'12345678901');
```

- Consulta básica

```
SELECT * FROM Pessoa;
```

- Consulta com filtros

```
SELECT nome, email FROM Pessoa WHERE cpf = '12345678901';
```

```
SELECT cpf, email FROM Pessoa WHERE nome = pedro;
```

Consultas básicas

- Atualizar registros

```sql
UPDATE Pessoa
SET email = 'novoemail@email.com'
WHERE id = 1;
```

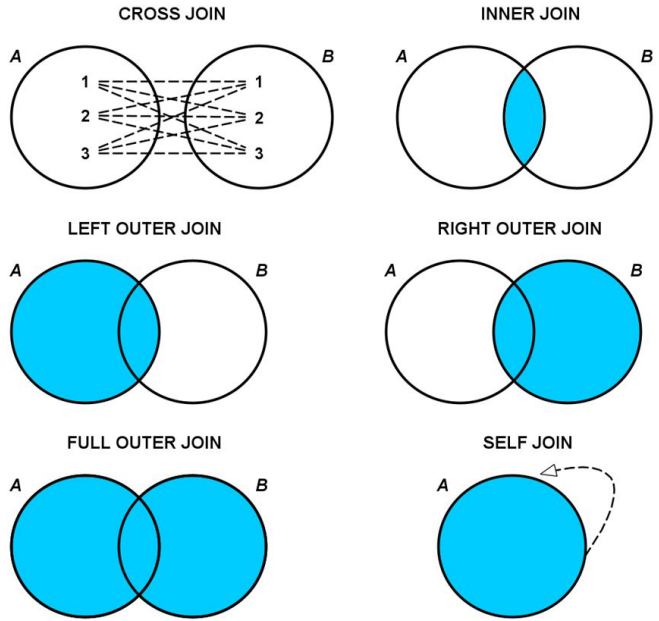
- Excluir registros

```sql
DELETE FROM Pessoa WHERE id = 1;

DELETE FROM Emprego WHERE id = 100;
```

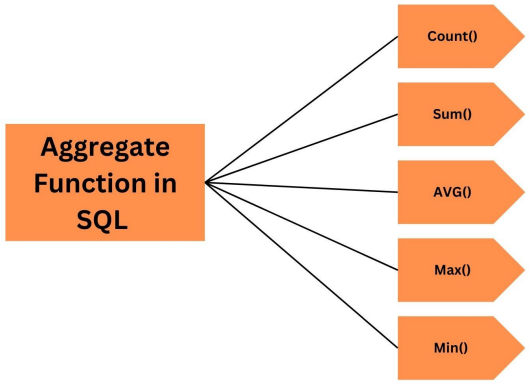
Funcionalidades avançadas da SQL

- **Junções:** Permitem combinar dados de várias tabelas.
  - Tipos:
    - **INNER JOIN** retorna registros que possuem correspondência em ambas as tabelas.
    - **LEFT JOIN** retorna todos os registros da tabela da esquerda, mesmo sem correspondência.
    - **RIGHT JOIN** semelhante ao **LEFT JOIN**, mas com prioridade para a tabela da direita.
    - **FULL JOIN** retorna todos os registros, combinando os correspondentes.



Funcionalidades avançadas da SQL

- **Funções Agregadas:**
  - Permitem combinar dados de várias tabelas.
  - São as seguintes:
    - **COUNT:** conta registros.
    - **SUM:** soma valores numéricos.
    - **AVG:** calcula média.
    - **MAX** e **MIN:** identificam maior e menor valor.



- **Subconsultas:**
  - Consultas aninhadas dentro de outra consulta.

```
SELECT nome FROM alunos WHERE id_curso = (SELECT id FROM cursos WHERE nome_curso = 'Matemática');
```

Joins e subconsultas

- Inner Join

```
SELECT Pessoa.nome,
Pedido.valor
FROM Pessoa
INNER JOIN Pedido ON Pessoa.id =
Pedido.pessoa_id;
```

- Left Join

```
SELECT Pessoa.nome,
Pedido.valor
FROM Pessoa
LEFT JOIN Pedido ON Pessoa.id =
Pedido.pessoa_id;
```

Joins e subconsultas

- Subconsulta em WHERE

```
SELECT nome
FROM Pessoa
WHERE id IN
(SELECT pessoa_id FROM Pedido WHERE valor > 100);
```

Funções agregadas e janelas

- Funções Agregadas
  - COUNT, SUM, AVG, MAX, MIN

```
SELECT
    COUNT(*) AS total_orders,
    SUM(total_amount) AS total_sum,
    AVG(total_amount) AS average,
    MAX(total_amount) AS max_amount,
    MIN(total_amount) AS min_amount
FROM
    orders;
```

- Funções de Janela
  - Ranking e Particionamento

```
SELECT nome, valor,
       RANK() OVER (PARTITION BY
pessoa_id
ORDER BY valor DESC)
AS rank
FROM Pedido;
```

Transações e segurança

- Transações
  - Uso de BEGIN, COMMIT e ROLLBACK

```
BEGIN;
UPDATE Conta SET saldo = saldo - 100
WHERE id = 1;
UPDATE Conta SET saldo = saldo + 100
WHERE id = 2;
COMMIT;
```

- Controle de Acesso
  - Criando usuários e permissões

```
CREATE USER 'usuario'@'localhost'
IDENTIFIED BY 'senha';
GRANT SELECT,
INSERT ON Pessoa TO
'usuario'@'localhost';
```

PEP (Python Enhancement Proposal)

- PEP - Proposta de Aprimoramento do Python.
- Documento técnico que descreve novos recursos, mudanças ou diretrizes para o desenvolvimento da linguagem Python.
- Servem como um registro formal das decisões da comunidade Python sobre o idioma e sua biblioteca padrão.
- Cada PEP passa por revisão antes de ser aprovada, rejeitada ou marcada como obsoleta.

Principais PEPs

PEP	Título	Descrição	Categoria
PEP 8	Estilo de Código do Python	Define diretrizes para escrever código Python legível e padronizado. - Indentação de 4 espaços. - Linhas com no máximo 79 caracteres. - Nomes de variáveis e funções em snake_case. - Classes em CamelCase.	Informativa
PEP 20	Zen do Python	Lista os princípios de design da linguagem Python, como simplicidade e legibilidade. - "Explícito é melhor que implícito.“ - "A simplicidade é melhor que a complexidade.“ - "Embora prática supere a pureza."	Informativa
PEP 257	Convenções de Docstrings	- Estabelece como documentar funções, classes e módulos com docstrings.	Informativa

Principais PEPs

PEP	Título	Descrição	Categoria
PEP 249	DB-API	Define o padrão para interação com bancos de dados relacionais no Python	Funcionalidade
PEP 405	Ambientes Virtuais (venv)	Introduziu suporte oficial para ambientes virtuais no Python.	Funcionalidade
PEP 484	Sugestões de Tipos (Type Hints)	Introduziu anotações de tipos em Python para melhorar a legibilidade e a validação de código.	Funcionalidade
PEP 498	f-strings (Strings Formatadas)	Adicionou suporte para strings literais interpoladas (f-strings).	Funcionalidade

Principais PEPs

PEP	Título	Descrição	Categoria
PEP 376	Estrutura de Instalação de Pacotes	Padronizou a forma como os pacotes são instalados e gerenciados no Python.	Processo
PEP 602	Ciclo de Lançamento Anual do Python	Define um ciclo anual de lançamentos para versões principais do Python.	Processo

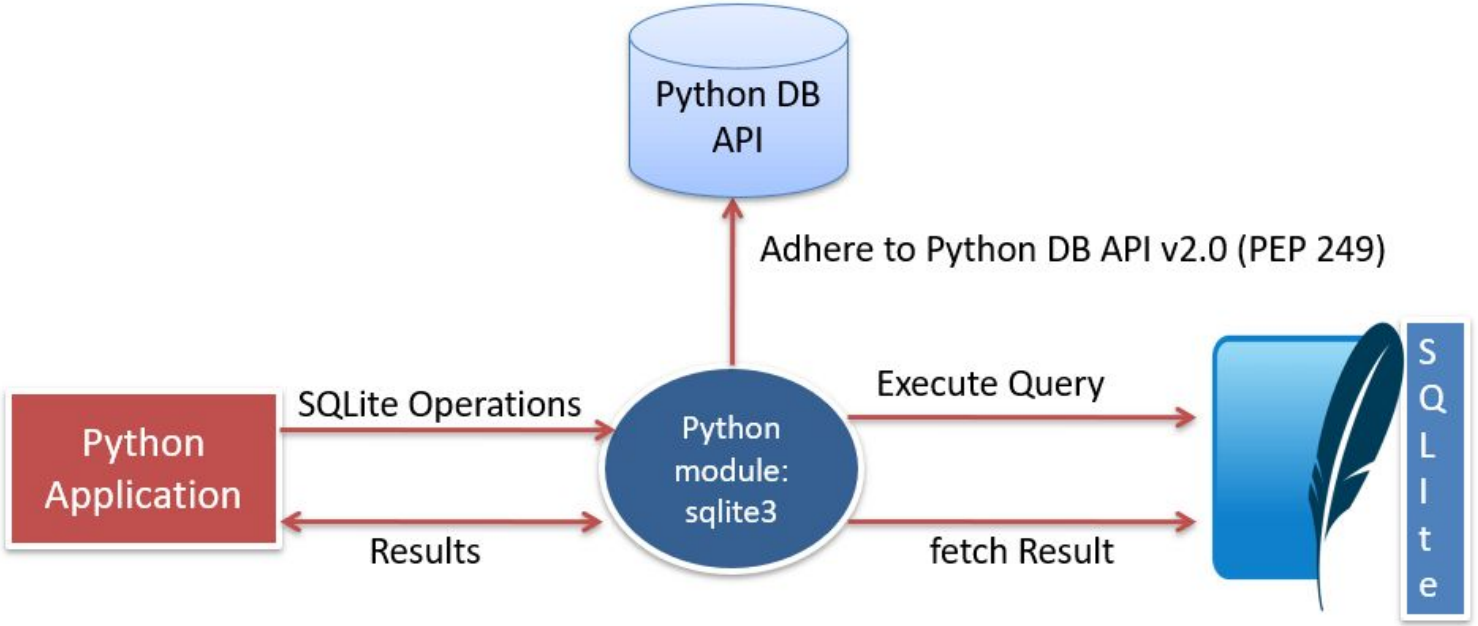
DB-API do Python – PEP 249

- DB-API (Database Application Programming Interface) v2.0 ou PEP 249
  - Padrão definido para permitir que aplicações Python se conectem a diferentes bancos de dados de maneira uniforme, garantindo portabilidade e simplicidade no desenvolvimento.

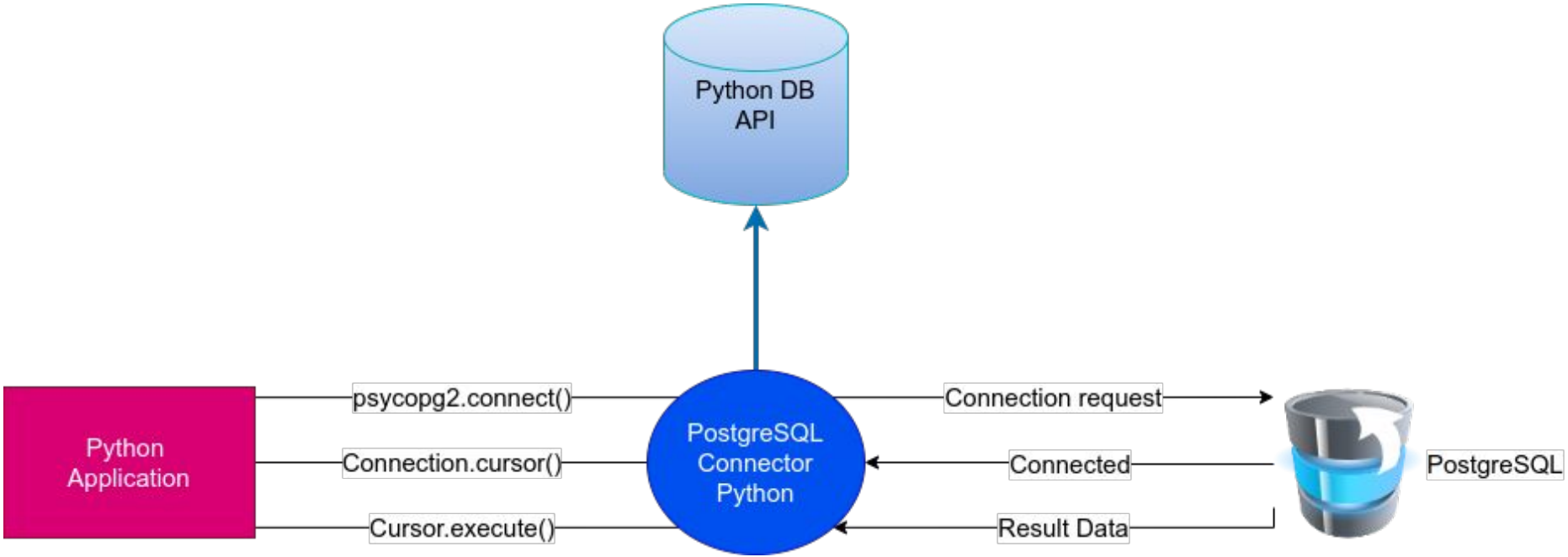
Benefícios do Padrão DB-API

- **Portabilidade**
  - Código escrito com a DB-API pode ser facilmente adaptado para diferentes bancos de dados.
- **Consistência**
  - Todos os módulos compatíveis seguem a mesma interface, reduzindo a curva de aprendizado.
- **Simplicidade**
  - Estrutura intuitiva e fácil de integrar com outras bibliotecas Python.
- **Flexibilidade**
  - Permite que desenvolvedores criem módulos para qualquer banco de dados relacional que respeite o padrão.

DB-API do Python – PEP 249

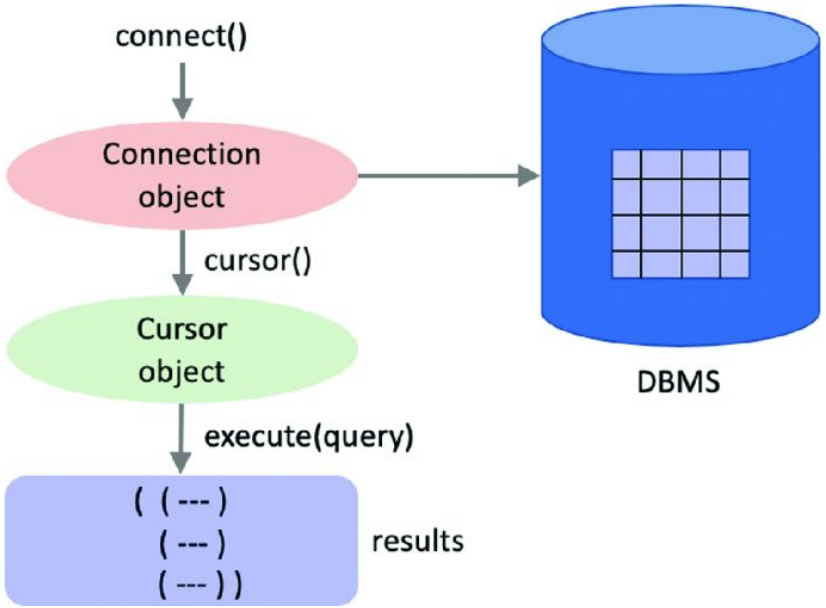


DB-API do Python – PEP 249



Objetos Fundamentais da DB-API

- Conexão (Connection)
- Cursor
- Exceções



Objetos Fundamentais da DB-API

- **Conexão (Connection)**
  - Representa a ligação entre a aplicação e o banco de dados.
  - Fornece métodos para gerenciar a conexão, como abrir, fechar e iniciar transações.

  - Métodos principais
    - **cursor():** cria um cursor para executar comandos SQL.
    - **commit():** confirma transações.
    - **rollback():** desfaz transações.
    - **close():** encerra a conexão.

Objetos Fundamentais da DB-API

- **Cursor**
  - Utilizado para executar comandos SQL e manipular os resultados.
  - Permite realizar operações como consultas, inserções, atualizações e exclusões.

  - Métodos principais:
    - **execute(sql, params=None):** executa um comando SQL.
    - **fetchone():** retorna o próximo registro.
    - **fetchmany(size=n):** retorna até n registros.
    - **fetchall():** retorna todos os registros.
    - **close():** fecha o cursor.

Objetos Fundamentais da DB-API

- **Exceções**
  - Uma hierarquia padrão de exceções ajuda a identificar e tratar erros.
  - Exemplos de exceções comuns
    - **DatabaseError:** erros gerais relacionados ao banco de dados.
    - **OperationalError:** erros operacionais, como problemas de conexão.
    - **ProgrammingError:** erros em instruções SQL ou na lógica da aplicação.

Exemplo - parte 1

```
import psycopg2

# Conectar ao banco de dados
try:
    conn = psycopg2.connect(
        dbname='exemplo',
        user='seu_usuario',
        password='sua_senha',
        host='localhost',
        port='5432'
    )
    cursor = conn.cursor()
```

Exemplo - parte 2

```
    try:
        # Criar tabela
        cursor.execute('''
            CREATE TABLE IF NOT EXISTS alunos (
                id SERIAL PRIMARY KEY,
                nome TEXT NOT NULL
            )
        ''')

        # Inserir dados
        cursor.execute('INSERT INTO alunos (nome) VALUES (%s)', ('Maria',))
        cursor.execute('INSERT INTO alunos (nome) VALUES (%s)', ('João',))
        conn.commit()
    except Exception as e:
        conn.rollback()
        print(f'Erro ao executar operações no banco de dados: {e}')
```

Exemplo - parte 3

```
    # Consultar dados
    cursor.execute('SELECT * FROM alunos')
    resultados = cursor.fetchall()

    for linha in resultados:
        print(linha)

except Exception as e:
    print(f'Erro ao conectar ao banco de dados: {e}')
finally:
    # Fechar conexões
    if cursor:
        cursor.close()
    if conn:
        conn.close()
```

Referências

- PEP 249 – Python Database API Specification v2.0 | peps.python.org
- Python Database API Specification 2.0 — pyfirebirdsql 1.0.0 documentation
- The Novice’s Guide to the Python 3 DB-API | Phil Varner

Obrigado!

Dúvidas?

Universidade Federal do Ceará - *Campus* Quixadá

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