Documentação do CRUD Gerenciador de Usuários

Este documento descreve as funcionalidades e os procedimentos disponíveis no sistema de Gerenciador de Usuários, implementado com operações CRUD básicas em um banco de dados PostgreSQL.

Tecnologias Utilizadas

- **Banco de Dados**: PostgreSQL JSON

Funcionalidades Implementadas:

Selecionar Pessoas por Nome

```
SELECT * FROM select_by_name('Luiz Gustavo Macedo');
```

1. Esta função retorna todas as pessoas cujo nome corresponde exatamente ao fornecido como parâmetro. Por exemplo, ao chamar `select_by_name('Luiz Gustavo Macedo')`, o sistema retornará todas as entradas na tabela de pessoas onde o nome seja exatamente "Luiz Gustavo Macedo".

Selecionar Pessoas com Data de Nascimento Maior que uma Data Específica

```
SELECT * FROM select_pessoas_by_data_nascimento('1991-05-11');
```

2. Esta função retorna todas as pessoas que têm a data de nascimento posterior à data especificada no parâmetro. Por exemplo, ao chamar `select_pessoas_by_data_nascimento('1991-05-11')`, o sistema retornará todas as pessoas que nasceram após 11 de maio de 1991.

Selecionar Telefones com DDD Específico

```
SELECT * FROM select_by_ddd('061');
```

3. Esta função retorna todos os registros de telefones que têm o DDD especificado no número de telefone. Por exemplo, ao chamar `select_by_ddd('061')`, o sistema retornará todos os registros de telefones que tenham o DDD "061".

Contar o Número de Registros

```
SELECT count_records();
```

4. Esta função retorna o número total de registros na tabela de pessoas. É útil para verificar quantos registros existem na tabela em um determinado momento.

Selecionar uma Pessoa por ID

```
SELECT * FROM select_pessoa_by_id(1);
```

5. Esta função retorna os dados de uma pessoa específica com base no ID fornecido como parâmetro. Por exemplo, `select_pessoa_by_id(1)` retornará os dados da pessoa cujo ID é 1 na tabela de pessoas.

Selecionar Todas as Pessoas

```
SELECT * FROM select_all_pessoas();
```

6. Esta função retorna todos os registros de pessoas na tabela, mostrando todos os campos e todas as entradas. É útil para visualizar todas as pessoas cadastradas no sistema.

Cada uma dessas seleções oferece uma funcionalidade específica para manipular e visualizar os dados de pessoas dentro do sistema de Gerenciador de Usuários, facilitando a gestão e a consulta de informações conforme necessário.

Documentação para Geração de Dados Falsos com Faker

Introdução

Este documento descreve o processo para gerar um grande conjunto de dados falsos usando a biblioteca `Faker` em Python. O objetivo é criar um arquivo JSON contendo 10.000.000 de registros de pessoas, com informações como nome, telefone, endereço e data de nascimento.

Requisitos

Antes de executar o script, certifique-se de ter instalado o Python e as bibliotecas necessárias. Você pode instalar a biblioteca `Faker` usando o comando:

pip install faker

Script para Geração de Dados Falsos

O script abaixo gera 10.000.000 de registros de pessoas e salva em um arquivo JSON chamado `dados_exemplo_10m.json`.

```
from faker import Faker
import json

fake = Faker('pt_BR')  # Inicializa a biblioteca Faker com a localidade para português do Brasil

num_dados = 10000000  # Define o número de registros a serem gerados

dados = []  # Inicializa uma lista vazia para armazenar os dados gerados
n = 1  # Contador para acompanhar o progresso da geração de dados
for _ in range(num_dados):
    nome = fake.name()
    telefone = fake.phone_number()
    endereco = fake.address()
    data_nascimento = fake.date_of_birth(tzinfo=None, minimum_age=18, maximum_age=90)
    print(f'{n} criados')  # Imprime o progresso da geração de dados
    dados.append({
        'nome': nome,
        'telefone': telefone,
        'endereco': endereco,
        'data_nascimento': data_nascimento.strftime('%Y-%m-%d')
    })
    n = n + 1  # Incrementa o contador

# Salva os dados gerados em um arquivo JSON
with open('dados_exemplo_10m.json', 'w', encoding='utf-8') as f:
    json.dump(dados, f, ensure_ascii=False, indent=4)

print("Dados gerados e salvos em 'dados_exemplo_10m.json'.")
```

Passo a Passo para Executar o Script

Instale a biblioteca `Faker`:

```
pip install faker
```

1. **Copie o script acima para um arquivo Python**: Salve o script em um arquivo chamado `gerar_dados.py` (ou qualquer nome de sua preferência).

Execute o script: Navegue até o diretório onde o arquivo Python está localizado e execute o script com o comando:
`python gerar_dados.py`

2. **Verifique a criação do arquivo JSON**: Após a execução do script, um arquivo chamado `dados_exemplo_10m.json` será gerado no mesmo diretório. Este arquivo contém 10.000.000 de registros de pessoas em formato JSON.

Explicação do Código

- **Importações**:
  - `from faker import Faker`: Importa a biblioteca `Faker`.
  - `import json`: Importa a biblioteca `json` para manipulação de arquivos JSON.
- **Inicialização**:
  - `fake = Faker('pt_BR')`: Inicializa a biblioteca `Faker` com a localidade para português do Brasil.
  - `num_dados = 10000000`: Define o número de registros a serem gerados.
- **Geração de Dados**:
  - O loop `for` gera `num_dados` registros de pessoas.
  - A cada iteração, são gerados nome, telefone, endereço e data de nascimento fictícios.
  - Os dados gerados são adicionados à lista `dados`.
- **Salvamento dos Dados**:
  - O conteúdo da lista `dados` é salvo em um arquivo JSON chamado `dados_exemplo_10m.json` usando a função `json.dump`.

Inserção de Dados JSON em PostgreSQL com Python

Este documento descreve como inserir dados de um arquivo JSON em uma tabela PostgreSQL utilizando Python e a biblioteca `psycopg2`.

Requisitos

Antes de executar o script, certifique-se de ter o PostgreSQL e as bibliotecas necessárias instaladas. Você pode instalar a biblioteca `psycopg2` usando o comando:

sh

Copiar código

```
pip install psycopg2
```

Script para Inserção de Dados

O script abaixo lê dados de um arquivo JSON e insere esses dados na tabela `pessoas` no banco de dados PostgreSQL.

```
import psycopg2
import json
import time

def insert_data(conn_params, json_file):
    try:
        # Conectar ao banco de dados PostgreSQL
        conn = psycopg2.connect(**conn_params)
        cursor = conn.cursor()

        # Ler o arquivo JSON
        with open(json_file, 'r', encoding='utf-8') as f:
            data = json.load(f)

        # Iniciar contagem de tempo
        start_time = time.time()

        # Inserir dados JSON na tabela
        for record in data:
            json_str = json.dumps(record)
            cursor.execute("INSERT INTO pessoas (dados) VALUES
(%s)", (json_str,))

        # Confirmar a transação
        conn.commit()

        # Calcular tempo total de execução
        end_time = time.time()
        execution_time = end_time - start_time

        print(f'Todos os dados do arquivo {json_file} foram
inseridos com sucesso.')
        print(f'Tempo de execução: {execution_time:.2f} segundos.')
```

```
    except psycopg2.Error as e:
        print(f"Erro ao inserir dados: {e}")

    finally:
        # Fechar a conexão
        if conn is not None:
            cursor.close()
            conn.close()

# Parâmetros de conexão com o PostgreSQL
conn_params = {
    'dbname': 'JsonGerador',
    'user': 'postgres',
    'password': '12345678',
    'host': 'localhost',  # ou o endereço do seu servidor PostgreSQL
    'port': 5432  # porta padrão do PostgreSQL
}

# Nome do arquivo JSON a ser lido
json_file = 'dados_2.json'

# Chamar a função para inserir os dados e medir o tempo
insert_data(conn_params, json_file)
```

Passo a Passo para Executar o Script

Instale a biblioteca `psycopg2`:

```
pip install psycopg2
```

1. **Copie o script acima para um arquivo Python**: Salve o script em um arquivo chamado `inserir_dados.py` (ou qualquer nome de sua preferência).
2. **Prepare seu Banco de Dados**:
    - Crie um banco de dados chamado `JsonGerador`.
    - Crie uma tabela chamada `pessoas` com uma coluna chamada `dados` do tipo JSONB.

Exemplo de comando SQL para criar a tabela:

```
CREATE TABLE pessoas (
    dados JSONB
```

```
);
```

Execute o script: Navegue até o diretório onde o arquivo Python está localizado e execute o script com o comando:

```
python inserir_dados.py
```

Descrição das Funcionalidades

1. Conexão ao Banco de Dados

A função `insert_data` conecta ao banco de dados PostgreSQL usando os parâmetros fornecidos (`conn_params`).

2. Leitura do Arquivo JSON

O arquivo JSON é lido e seu conteúdo é armazenado na variável `data`.

3. Inserção dos Dados

Cada registro no JSON é convertido para uma string JSON e inserido na tabela `pessoas`.

4. Medição do Tempo de Execução

O tempo total de execução é calculado e exibido ao final do processo.

5. Tratamento de Erros e Fechamento da Conexão

Qualquer erro durante o processo de inserção é capturado e exibido. A conexão com o banco de dados é fechada no final, garantindo que os recursos sejam liberados adequadamente.