**Documentação Técnica do Código: Processamento de Dados e Integração com Banco de Dados PostgreSQL usando Docker**

**1. Introdução**

Este documento apresenta a documentação detalhada para o código Python desenvolvido para realizar a leitura e processamento de dados a partir de um arquivo Excel, comparar os dados com os registros existentes em um banco de dados PostgreSQL, identificar novos registros, e inserir esses novos dados no banco de dados. O processo inclui a utilização de containers Docker para manipulação dos arquivos CSV.

**2. Objetivo**

O objetivo do código é automatizar o processo de atualização de registros no banco de dados a partir de um arquivo Excel, garantindo que apenas os registros novos sejam inseridos. Isso é realizado através de um fluxo de ETL (Extract, Transform, Load) que lê os dados do Excel, processa-os, compara com os dados existentes no banco de dados, e carrega apenas os registros novos.

**3. Descrição Geral do Processo ETL**

O processo ETL (Extração, Transformação e Carga) implementado neste projeto pode ser descrito em três etapas principais:

1. **Extração:** Leitura dos dados de um arquivo Excel armazenado em um servidor da empresa.
2. **Transformação:** Processamento dos dados extraídos, incluindo a renomeação de colunas, conversão de datas, e eliminação de registros nulos.
3. **Carga:** Inserção dos novos registros no banco de dados PostgreSQL, utilizando um container Docker para execução dos comandos de inserção.

**4. Tecnologias Utilizadas**

* **Python**: Linguagem de programação principal para desenvolvimento do script.
* **Pandas**: Biblioteca utilizada para manipulação e análise de dados.
* **Docker**: Ferramenta de contêinerização usada para isolar e gerenciar a execução dos processos relacionados ao banco de dados.
* **SQLAlchemy**: Biblioteca usada para conectar-se ao banco de dados PostgreSQL e executar consultas SQL.
* **PostgreSQL**: Sistema de gerenciamento de banco de dados relacional usado para armazenar os dados processados.

**5. Decisões de Projeto**

**5.1 Estrutura do Código**

A estrutura do código foi dividida em funções modulares para cada uma das tarefas principais: leitura e processamento do arquivo Excel, obtenção de registros existentes no banco de dados, salvamento de dados em formato CSV, cópia de arquivos para o Docker e inserção de novos dados no banco de dados. Essa abordagem modular facilita a manutenção e a escalabilidade do código.

**5.2 Configuração do Banco de Dados**

As configurações do banco de dados, incluindo nome, usuário, senha, host e porta, foram centralizadas em um dicionário (db\_config). Isso permite a fácil modificação das credenciais e da configuração do banco de dados sem a necessidade de alterar várias partes do código.

**5.3 Tratamento de Dados no DataFrame**

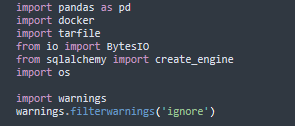
O DataFrame obtido da leitura do Excel passa por um processo de renomeação das colunas para garantir que os nomes estejam padronizados e adequados para inserção no banco de dados. A escolha de renomear as colunas no próprio código evita a necessidade de modificar o arquivo Excel original, preservando a integridade dos dados de entrada.

**5.4 Integração com Docker**

Para garantir que o código possa ser executado em diferentes ambientes sem a necessidade de instalar diretamente o PostgreSQL, optou-se pelo uso de containers Docker. Os arquivos CSV são copiados para o container, onde os comandos de inserção no banco de dados são executados. Isso também aumenta a segurança, uma vez que as operações no banco de dados são isoladas dentro do container.

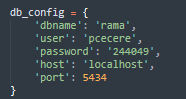
**6. Descrição do Código**

**6.1 Importação das Bibliotecas**

****

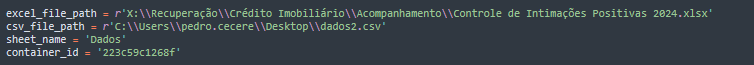
Essas importações incluem as bibliotecas necessárias para leitura e processamento de dados, interação com o Docker, manipulação de arquivos CSV, e conexão com o banco de dados PostgreSQL.

**6.2 Configurações do Banco de Dados**

****

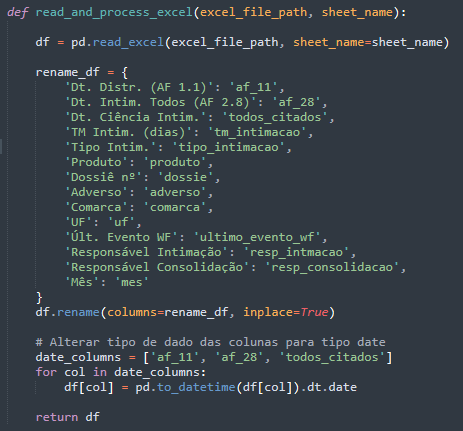
Essas configurações são armazenadas em um dicionário, facilitando a alteração em caso de mudanças no ambiente.

**6.3 Caminhos para os Arquivos**

****

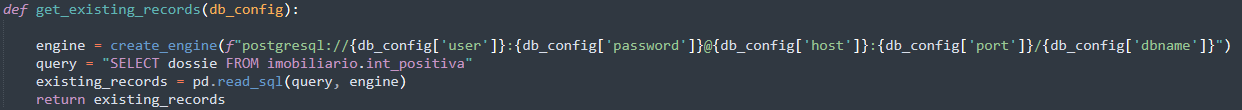
Os caminhos para os arquivos e o ID do container Docker são especificados aqui. A variável sheet\_name define a aba do Excel que contém os dados.

**6.4 Leitura e Processamento do Excel**



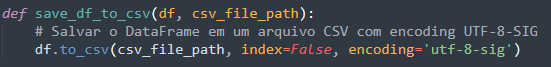
Esta função lê os dados da planilha Excel, renomeia as colunas conforme um dicionário de mapeamento e converte colunas específicas para o formato de data.

**6.5 Obtenção de Registros Existentes**



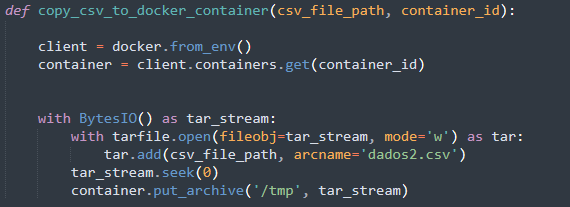
Esta função conecta-se ao banco de dados PostgreSQL usando SQLAlchemy e recupera os registros existentes na tabela imobiliario.int\_positiva.

**6.6 Salvamento de Dados em CSV**

****

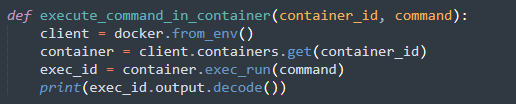
Aqui, o DataFrame processado é salvo como um arquivo CSV, pronto para ser carregado no container Docker.

**6.7 Cópia do CSV para o Docker**

****

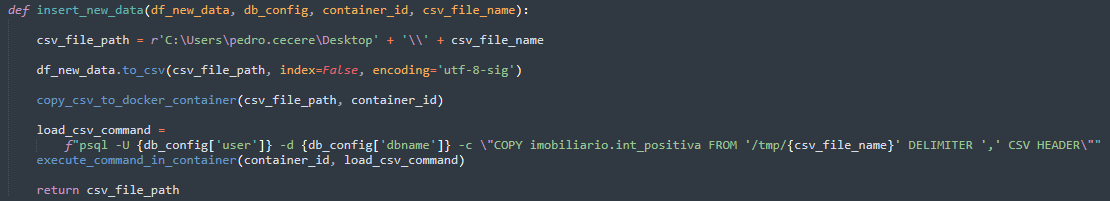
Este trecho copia o arquivo CSV gerado para o container Docker usando a biblioteca docker-py. O arquivo é compactado em um tarball antes de ser transferido para o container.

**6.8 Execução de Comandos no Container**

****

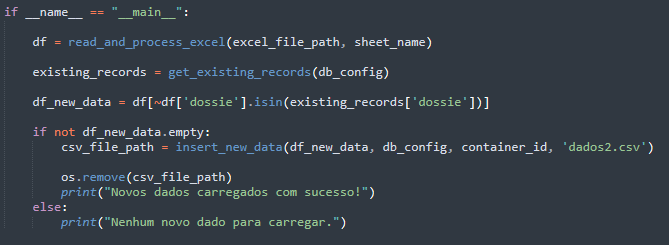
Esta função executa comandos diretamente no container Docker, o que é usado para carregar o CSV no banco de dados PostgreSQL dentro do container.

**6.9 Inserção de Novos Dados**



A função insert\_new\_data é responsável por todo o processo de inserção de novos registros: salvamento do DataFrame como CSV, cópia para o container, e execução do comando de carregamento dos dados no banco de dados.

**6.10 Execução Principal do Código**



O bloco \_\_main\_\_ coordena a execução das funções, garantindo que o processo ocorra na ordem correta: leitura dos dados, comparação com registros existentes, e inserção dos novos dados.

**7. Considerações Finais**

Este código foi desenvolvido para garantir a automação e eficiência no processamento de dados e na atualização do banco de dados. O uso de Docker permite a replicação do ambiente em diferentes sistemas sem necessidade de configuração complexa. A modularização do código facilita a manutenção e permite futuras expansões ou modificações.