Arquitetura de Dados

Através do Júpiter (aplicação web de código aberto), fizemos a preparação para analisar os dados.

Utilizamos o Python para estabelecer uma conexão com um arquivo CSV (dados de fontes externas) e transformá-los em um DataFrame (estrutura de dados semelhante a uma tabela), usando a biblioteca Pandas (especializada em manipulação e análise de dados).

Código em Python, voltado para a manipulação de dados e interação com um banco de dados MySQL (processo de ETL (Extração, Transformação e Carregamento).

Conexão com MySQL:

- São importadas as bibliotecas ***mysql.connector*** (como sql) para conectar ao MySQL, **warnings** para manipular avisos, e **pandas** (como pd) para operações de dados;
- É definida uma função chamada ***conexao***, que:

1. Abre e lê um arquivo chamado **arquivo.txt** (senha para acessar o banco de dados);
2. Estabelece conexão com o banco de dados MySQL, utilizando o ***host***, ***user***, e ***database*** fornecidos como parâmetros, juntamente com a senha lida do arquivo.

Essa função **conexao** é chamada com os parâmetros **'localhost'**, **'root'**, e ***'t_final'*** para estabelecer uma conexão, que é armazenada na variável ***conn***.

Conexão com Arquivo CSV:

- Um DataFrame **(df)** é criado a partir de um arquivo CSV chamado dados.csv, usando **pd.read_csv** e a codificação **'ISO-8859-1'**. O separador de campos é definido como **(',',)** e o arquivo não possui um cabeçalho na primeira linha **(header = None)**;
- Os cabeçalhos das colunas são definidos com base na primeira linha dos dados do CSV;
- A primeira linha do DataFrame (que agora contém os cabeçalhos) é descartada para manter apenas os dados.

Inserção de dados no banco de dados

- Várias funções são definidas para inserir dados em diferentes tabelas do banco de dados;
- Cada função lê diferentes colunas do **DataFrame (df)** e insere esses dados no banco de dados, evitando duplicatas;
- É utilizada uma abordagem iterativa para processar cada linha do **DataFrame** e formar comandos SQL de inserção.

Funções de Inserção:

- Cada função começa criando uma lista vazia, como **lista_all**, que será usada para armazenar conjuntos de dados únicos. Essa lista evita a inserção de dados duplicados no banco de dados;
- Cada função itera sobre as linhas do **DataFrame (df)**, usando um **loop for**. O **DataFrame (df)** contém os dados lidos do arquivo **CSV**;
- Dentro do **loop**, para cada linha do DataFrame, a função cria uma lista temporária, como **lista_one**, para armazenar um conjunto específico de dados dessa linha;
- Em cada iteração, a função extrai dados específicos da linha atual do DataFrame. Estes dados são adicionados à lista temporária **(lista_one)**;
- Para evitar duplicatas no banco de dados, antes de adicionar o conjunto de dados **(na lista_one)** à lista principal **(lista_all),** a função verifica se esse conjunto de dados já existe na lista principal. Se o conjunto de dados não estiver na **lista_all**, ele é considerado único e é adicionado à lista;
- Após processar todas as linhas do DataFrame, a função itera sobre a **lista_all**, que agora contém conjuntos de dados únicos e para cada conjunto de dados na **lista_all** e a função forma um comando SQL de inserção;

- A função executa cada comando SQL formado, inserindo os dados no banco de dados e após a execução de cada um, é realizado um **commit** para garantir que as alterações sejam efetivamente salvas no banco de dados.

Criando Banco de Dados (MySQL) e Tabelas

- Utilizamos o comando **CREATE DATABASE T_FINAL** para criar um novo banco de dados chamado t_final;
- Utilizamos o comando **USE T_FINAL** para selecionar o banco de dados t_final para uso;
- Utilizamos o comando **CREATE TABLE** para criar uma nova tabela com um conjunto de colunas e tipos de dados. As chaves primárias **(primary key)** são definidas para identificar de forma única cada registro nessas tabelas;
- Utilizamos o comando **INSERT INTO** 'nome da tabela' **VALUES** para inserir os registros nas tabelas;
- Utilizamos o comando **ALTER TABLE** 'nome da tabela' para modificar a tabela. Ex: ALTER TABLE cliente DISABLE KEYS e ALTER TABLE cliente ENABLE KEYS foram usados para desabilitar e habilitar temporariamente a verificação de chaves durante a inserção em massa de dados, melhorando a performance;
- Utilizamos o comando **TRUNCATE** para remover rapidamente todos os registros de uma tabela, mas mantendo sua estrutura intacta;
- Utilizamos o comando **SELECT** para selecionar registros específicos das tabelas. Ex: **SELECT** cod_cliente, cod_tipo_cliente FROM cliente WHERE cod_empresa;
- Utilizamos o comando **DROP TABLE IF EXISTS** 'nome da tabela' para remover alguma tabela, se ela já existir, garantindo que o script não falhe devido à presença de uma tabela existente.

Tabelas

- create table empresa: Esta entidade contém informações das empresas: cod_empresa (chave primária) e nome_empresa;
- create table tipo_cliente: Esta entidade categoriza os clientes. Tem dois atributos: cod_tipo_cliente (chave primária) e nome_tipo_cliente;
- create table cliente**:** Esta entidade representa os clientes e está relacionada à empresa. Possui três atributos: cod_cliente (chave primária), cod_empresa, e cod_tipo_cliente. O cliente está ligado à empresa (um cliente pertence a uma empresa) e também a um tipo de cliente, indicando uma classificação ou categoria.
- create table produto: Esta entidade contém detalhes dos produtos com cod_produto (chave primária), nome_produto e marca_produto;
- create table cliente_produto: É um relacionamento muitos-para-muitos entre clientes e produtos, indicando quais produtos são comprados por quais clientes, sem uma chave primária definida. Contém cod_produto e cod_cliente;
- create table regiao: Representa regiões geográficas com cod_municipio (chave primária) e nome_municipio como atributos, e uf, indicando a unidade federativa;
- create table regiao_produto: Associa regiões a produtos, indicando quais produtos estão disponíveis em quais regiões. Possui os atributos: cod_municipio e cod_produto;
- create table regiao_cliente: Essa tabela mantém o relacionamento entre clientes e as regiões onde eles residem ou operam, armazenando chaves estrangeiras de ambas as entidades: cod_municipio e cod_cliente;
- create table venda: Registra vendas com atributos: id_venda (chave primária), ano_mes, vol_litros e total_bruto;
- create table venda_regiao: Associa vendas a regiões, indicando onde a venda ocorreu. Possui os atributos: id_venda e cod_municipio;
- create table venda_produto: Associa vendas a produtos, indicando quais produtos foram vendidos em cada venda. Possui os atributos: id_venda e cod_produto;

- create table venda_cliente: Associa vendas a clientes, mostrando qual cliente fez qual venda. Possui os atributos: id_venda e cod_cliente.

Descrição do Modelo Entidade-Relacionamento (MER)

- t_final_empresa e t_final_cliente: Um relacionamento um-para-muitos (1:N), indicando que uma empresa pode ter muitos clientes, mas cada cliente está associado a apenas uma empresa;
- t_final_cliente e t_final_tipo_cliente: Um relacionamento um-para-muitos (1:N), sugerindo que um cliente pertence a um único tipo, mas um tipo de cliente pode se aplicar a muitos clientes diferentes;
- t_final_cliente e t_final_cliente_produto: Um relacionamento muitos-para-muitos (N:M), representando que clientes podem comprar diversos produtos e que produtos podem ser comprados por muitos clientes;
- t_final_cliente e t_final_regiao_cliente: Um relacionamento um-para-muitos (1:N), onde um cliente está associado a uma única região, mas uma região pode conter muitos clientes;
- t_final_venda e t_final_venda_produto: Um relacionamento muitos-para-muitos (N:M), mostrando que uma venda pode envolver vários produtos e um produto pode ser vendido em muitas vendas diferentes;
- t_final_venda e t_final_venda_cliente: Um relacionamento um-para-muitos (1:N), onde uma venda é associada a um único cliente, mas um cliente pode ter várias vendas;
- t_final_venda e t_final_venda_regiao: Um relacionamento muitos-para-muitos (N:M), indicando que uma venda pode ser realizada em várias regiões e que uma região pode ter muitas vendas associadas a ela;
- t_final_produto e t_final_cliente_produto: Um relacionamento muitos-para-muitos (N:M), indicando que um produto pode ser comprado por muitos clientes e um cliente pode comprar muitos produtos;
- t_final_produto e t_final_venda_produto: Um relacionamento um-para-muitos (1:N), onde um produto pode ser parte de muitas vendas, mas cada venda de produto é associada a um único produto específico;
- t_final_produto e t_final_regiao_produto: Um relacionamento muitos-para-muitos (N:M), mostrando que um produto pode ser oferecido em várias regiões e que uma região pode ter muitos produtos disponíveis;
- t_final_regiao e t_final_regiao_produto: Um relacionamento muitos-para-muitos (N:M), onde uma região pode ter muitos produtos e um produto pode estar disponível em muitas regiões diferentes;
- t_final_regiao e t_final_venda_regiao: Um relacionamento um-para-muitos (1:N), onde uma região pode ser o local de muitas vendas, mas cada venda é atribuída a uma única região.