Descrição do Desafio Técnico

Descrição do Desafio

1. Dados de Entrada

Você receberá um conjunto de arquivos CSV representando tabelas de um banco de dados transacional bruto. Esses dados são referentes à dívida ativa de uma Procuradoria Municipal, cujo item principal é a CDA (Certidão de Dívida Ativa). Neles, você encontrará informações como a natureza do tributo, o saldo de cada dívida, detalhes sobre o devedor e a situação de cada registro.

2. Tarefas Principais

Modelagem do Banco Transacional

- Criar um modelo relacional normalizado (SQL).
- Implementar scripts de criação das tabelas.
- Inserir os dados fornecidos via CSV no banco criado.

Data Warehouse & ETL

- Projetar um Data Warehouse que sirva como base para um dashboard analítico.
- Implementar um processo ETL que:
  - Extraia os dados do banco transacional.
  - Transforme os dados conforme necessário.
  - Carregue os dados no Data Warehouse.

Containerização

Toda a solução de bancos (transacional e DW) deve ser levantada via Docker (Docker Compose).

API (FastAPI)

Implementar uma API com os *endpoints* a seguir.

Endpoints Requeridos e Formatos de Saída

1. GET /cda/search

Parâmetros de consulta (query):

- `numCDA`: string
- `minSaldo`: float
- `maxSaldo`: float
- `minAno`: int
- `maxAno`: int
- `natureza`: string
- `agrupamento_situacao`: int
- `sort_by`: "ano" ou "valor"
- `sort_order`: "asc" ou "desc"

Formato de saída:

```
[
 {
 "numCDA": "string",
 "valor_saldo_atualizado": "float",
 "qtde_anos_idade_cda": "int",
 "agrupamento_situacao": "int",
                         //cancelada, paga ou em cobranca
 "natureza": "string",
 "score": float,
                        // probabilidade de recuperacao
 }
]
```

2. GET /cda/detalhes_devedor

Formato de saída:

```
[
 {
 "name": "string",          // Nome do devedor
 "tipo_pessoa": "string", // PF ou PJ
 "CPF/CNPJ": "string",     // CPF para PF e CNPJ para PJ
 }
]
```

3. GET /resumo/distribuicao_cdas

Formato de saída:

```
[
 {
 "name": "string",
                          // tipo do tributo (IPTU, ISS, etc.)
 "Em cobranca": "float", // percentual
 "Cancelada": "float",   // percentual
 "Quitada": "float"      // percentual
 }
]
```

4. GET /resumo/inscricoes

Formato de saída:

```
[
 {
  "ano": "int",
  "Quantidade": "int"
 }
]
```

5. GET /resumo/montante_acumulado

Formato de saída:

```
[
 {
  "Percentual": "int",  // percentil (ex: 1, 5, 10, ...)
  "IPTU": "float",      // percentual acumulado
  "ISS": "float",
  "Taxas": "float",
  "Multas": "float",
  "ITBI": "float"
 }
]
```

6. GET /resumo/quantidade_cdas

Formato de saída:

```
[
 {
  "name": "string",      // tipo do tributo
  "Quantidade": "int"    // quantidade de CDAs
 }
]
```

7. GET /resumo/saldo_cdas

Formato de saída:

```
[
  {
    "name": "string",              // tipo do tributo
    "Saldo": "float"               // saldo total
  }
]
```

Requisitos Técnicos

- **Banco Transacional:** PostgreSQL, MySQL ou outro banco equivalente.
- **Data Warehouse:** PostgreSQL, MySQL ou outro banco relacional compatível.
- **ETL:** Pode ser implementado em Python ou SQL puro.
- **API:** Python + FastAPI.
- **Containerização:** Docker (docker-compose.yml obrigatório).

Entrega Esperada

- *Scripts* SQL para criação e carga dos bancos.
- *Scripts* ou código do ETL.
- Código da API (com instruções para execução).
- Arquivo `docker-compose.yml` para subir toda a stack.
- Um arquivo `README.md` explicando:
  - Como rodar o projeto;
  - Como testar a API;
  - Qualquer dependência adicional.

Critérios de Avaliação

- Correção da modelagem e normalização do banco.
- Clareza e manutenibilidade do código.
- Estruturação adequada do ETL e do DW.
- Qualidade dos *endpoints* (performance, consistência dos dados, validação dos parâmetros).
- Uso correto de Docker.
- Organização do repositório e documentação.