Untitled

January 9, 2026

[11]:
```
#bloco de bibliotecas
import pandas as pd
from pathlib import Path
from datetime import datetime
import uuid
from sqlalchemy import create_engine
from urllib.parse import quote_plus
from sqlalchemy import text
```

[12]:
```
#bloco com funções de normalização

#função para averiguar nome das colunas para achar com o nome "data"
def nome_col_data (col_nome):
    return "data" in str(col_nome).strip().lower()

#função pegando do nome "data" trasnsformando para o tipo data
def transformar_data (serie):
    return pd.to_datetime(
        serie,
        errors = "coerce"
    )

#função para transformar para o tipo texto, deixando tudo minúsculo e tirando␣
 ↪espaços em branco no começo e final
def transformar_texto (serie):
    return (
        serie
        .astype("string")
        .str.strip()
        .str.lower()
    )

#função para chamar cada função no seu caso
def normalizar_base (df):
    df = df.copy()
```

```
    #verifica se tem "data" no nome da coluna, chama as funções de␣
 ↪transformação data
    colunas_data = [col_data for col_data in df.columns if␣
 ↪nome_col_data(col_data)]
    for data in colunas_data:
        df[data] = transformar_data(df[data])

    #verifica se a coluna não esta no tipo data e nem como númerico, chama as␣
 ↪funções de transformação texto
    for x in df.columns:
        if x in colunas_data:
            continue
        if pd.api.types.is_object_dtype(df[x]) or pd.api.types.
 ↪is_string_dtype(df[x]):
            df[x] = transformar_texto(df[x])
    return df

#função para chamar sql mais facil
def run_sql(engine, sql: str, params: dict | None = None):
    with engine.begin() as conn:
        conn.execute(text(sql), params or {})

#função para subir cada tabela staging
def staging_um_a_um(df, engine, schema, table, truncate_first: bool = True,␣
 ↪chunksize: int = 10_000):
    if truncate_first:
        truncate_table(engine, schema, table)
    df.to_sql(
        name = table,
        con = engine,
        schema = schema,
        if_exists = "append",
        index = False,
        chunksize = chunksize,
        method = "multi"
    )

#função para subir todas de uma vez chamando a função "staging_um_a_um"
def alimentar_staging(base_normalizada, engine, schema, truncate_first: bool =␣
 ↪True, chunksize: int = 10_000):
    for tabela, df in base_normalizada.items():
        staging_um_a_um(
            df = df,
            engine = engine,
            schema = schema,
            table = str(tabela),
```

```
            truncate_first=truncate_first,
            chunksize=chunksize
        )

#função para limpar o stagingpode
def truncate_table(engine, schema, table):
    with engine.begin() as conn:
        conn.execute(text(f'TRUNCATE TABLE "{schema}"."{table}";'))

#função que alimenta as dimensões
def alimenta_dim(engine):
    sqls = [

        #DIM_TEMPO
        """
        INSERT INTO dw.dim_tempo (data_id, data, ano, mes, dia)
        SELECT DISTINCT
          (EXTRACT(YEAR FROM d)::int * 10000
         + EXTRACT(MONTH FROM d)::int * 100
         + EXTRACT(DAY FROM d)::int) AS data_id,
          d::date AS data,
          EXTRACT(YEAR FROM d)::int AS ano,
          EXTRACT(MONTH FROM d)::int AS mes,
          EXTRACT(DAY FROM d)::int AS dia
        FROM (
            SELECT "Data_Competencia" AS d FROM stg."Faturamento"
            UNION
            SELECT "Data_Competencia" FROM stg."Custos_Despesas"
            UNION
            SELECT "Data_Competencia" FROM stg."Orcamento"
            UNION
            SELECT "Data_Emissao" FROM stg."Contas_Receber"
            UNION
            SELECT "Data_Vencimento" FROM stg."Contas_Receber"
        ) x
        WHERE d IS NOT NULL
        ON CONFLICT (data) DO NOTHING;
        """,

        #DIM_CLIENTE
        """
        INSERT INTO dw.dim_cliente (cliente)
        SELECT DISTINCT "Cliente" FROM stg."Faturamento"
        WHERE "Cliente" IS NOT NULL
        UNION
        SELECT DISTINCT "Cliente" FROM stg."Contas_Receber"
        WHERE "Cliente" IS NOT NULL
```

```
        ON CONFLICT (cliente) DO NOTHING;
        """,

        #DIM_UNIDADE_NEGOCIO
        # recebe: Faturamento.Unidade_Negocio + Orcamento.Categoria
        """
        INSERT INTO dw.dim_unidade_negocio (unidade_negocio)
        SELECT DISTINCT "Unidade_Negocio" AS unidade_negocio
        FROM stg."Faturamento"
        WHERE "Unidade_Negocio" IS NOT NULL
        UNION
        SELECT DISTINCT "Categoria" AS unidade_negocio
        FROM stg."Orcamento"
        WHERE "Categoria" IS NOT NULL
        ON CONFLICT (unidade_negocio) DO NOTHING;
        """,

        #DIM_TIPO_CONTRATO
        """
        INSERT INTO dw.dim_tipo_contrato (tipo_contrato)
        SELECT DISTINCT "Tipo_Contrato"
        FROM stg."Faturamento"
        WHERE "Tipo_Contrato" IS NOT NULL
        ON CONFLICT (tipo_contrato) DO NOTHING;
        """,

        #DIM_CATEGORIA
        """
        INSERT INTO dw.dim_categoria (categoria)
        SELECT DISTINCT "Categoria"
        FROM stg."Custos_Despesas"
        WHERE "Categoria" IS NOT NULL
        ON CONFLICT (categoria) DO NOTHING;
        """,

        #DIM_TIPO_DESPESA
        """
        INSERT INTO dw.dim_tipo_despesa (tipo_despesa)
        SELECT DISTINCT "Tipo_Despesa"
        FROM stg."Custos_Despesas"
        WHERE "Tipo_Despesa" IS NOT NULL
        ON CONFLICT (tipo_despesa) DO NOTHING;
        """,

        #DIM_CENTRO_CUSTO
        """
        INSERT INTO dw.dim_centro_custo (centro_custo)
```

```
        SELECT DISTINCT "Centro_Custo"
        FROM stg."Custos_Despesas"
        WHERE "Centro_Custo" IS NOT NULL
        ON CONFLICT (centro_custo) DO NOTHING;
        """,

        #DIM_STATUS
        """
        INSERT INTO dw.dim_status (status)
        SELECT DISTINCT "Status"
        FROM stg."Contas_Receber"
        WHERE "Status" IS NOT NULL
        ON CONFLICT (status) DO NOTHING;
        """,

        #DIM_TIPO_ORCAMENTO
        """
        INSERT INTO dw.dim_tipo_orcamento (tipo)
        SELECT DISTINCT "Tipo"
        FROM stg."Orcamento"
        WHERE "Tipo" IS NOT NULL
        ON CONFLICT (tipo) DO NOTHING;
        """
    ]

    with engine.begin() as conn:
        for sql in sqls:
            conn.execute(text(sql))

#função que alimenta as fatos
def alimenta_fato(engine):
    sqls = [

        #FATO FATURAMENTO
        """
        INSERT INTO dw.fato_faturamento (
            data_id, unidade_negocio_id, cliente_id, tipo_contrato_id,
            receita_bruta, impostos, receita_liquida,
            batch_id, tempo_carga
        )
        SELECT
            dt.data_id,
            dun.unidade_negocio_id,
            dc.cliente_id,
            dtc.tipo_contrato_id,
            s."Receita_Bruta",
            s."Impostos",
```

```
            s."Receita_Liquida",
            s.batch_id,
            s.tempo_carga
        FROM (
            SELECT DISTINCT ON ("Data_Competencia", "Unidade_Negocio",␣
↪"Cliente", "Tipo_Contrato")
                *
            FROM stg."Faturamento"
            ORDER BY
                "Data_Competencia", "Unidade_Negocio", "Cliente",␣
↪"Tipo_Contrato",
                tempo_carga DESC NULLS LAST
        ) s
        JOIN dw.dim_tempo dt
          ON dt.data = s."Data_Competencia"::date
        JOIN dw.dim_unidade_negocio dun
          ON dun.unidade_negocio = s."Unidade_Negocio"
        JOIN dw.dim_cliente dc
          ON dc.cliente = s."Cliente"
        JOIN dw.dim_tipo_contrato dtc
          ON dtc.tipo_contrato = s."Tipo_Contrato"
        ON CONFLICT (data_id, unidade_negocio_id, cliente_id, tipo_contrato_id)
        DO UPDATE SET
            receita_bruta   = EXCLUDED.receita_bruta,
            impostos        = EXCLUDED.impostos,
            receita_liquida = EXCLUDED.receita_liquida,
            batch_id        = EXCLUDED.batch_id,
            tempo_carga     = EXCLUDED.tempo_carga;
        """,

        #FATO CUSTOS/DESPESAS
        """
        INSERT INTO dw.fato_custos_despesas (
            data_id, categoria_id, tipo_despesa_id, centro_custo_id,
            valor, batch_id, tempo_carga
        )
        SELECT
            dt.data_id,
            dcat.categoria_id,
            dtd.tipo_despesa_id,
            dcc.centro_custo_id,
            s."Valor",
            s.batch_id,
            s.tempo_carga
        FROM (
            SELECT DISTINCT ON ("Data_Competencia", "Categoria",␣
↪"Tipo_Despesa", "Centro_Custo")
```

```
        *
    FROM stg."Custos_Despesas"
    ORDER BY
        "Data_Competencia", "Categoria", "Tipo_Despesa", "Centro_Custo",
        tempo_carga DESC NULLS LAST
) s
JOIN dw.dim_tempo dt
  ON dt.data = s."Data_Competencia"::date
JOIN dw.dim_categoria dcat
  ON dcat.categoria = s."Categoria"
JOIN dw.dim_tipo_despesa dtd
  ON dtd.tipo_despesa = s."Tipo_Despesa"
JOIN dw.dim_centro_custo dcc
  ON dcc.centro_custo = s."Centro_Custo"
ON CONFLICT (data_id, categoria_id, tipo_despesa_id, centro_custo_id)
DO UPDATE SET
    valor       = EXCLUDED.valor,
    batch_id    = EXCLUDED.batch_id,
    tempo_carga = EXCLUDED.tempo_carga;
""",

#FATO ORÇAMENTO
"""
INSERT INTO dw.fato_orcamento (
    data_id, tipo_orcamento_id, unidade_negocio_id,
    valor_orcado, batch_id, tempo_carga
)
SELECT
    dt.data_id,
    dto.tipo_orcamento_id,
    dun.unidade_negocio_id,
    s."Valor_Orcado",
    s.batch_id,
    s.tempo_carga
FROM (
    SELECT DISTINCT ON ("Data_Competencia", "Tipo", "Categoria")
        *
    FROM stg."Orcamento"
    ORDER BY
        "Data_Competencia", "Tipo", "Categoria",
        tempo_carga DESC NULLS LAST
) s
JOIN dw.dim_tempo dt
  ON dt.data = s."Data_Competencia"::date
JOIN dw.dim_tipo_orcamento dto
  ON dto.tipo = s."Tipo"
JOIN dw.dim_unidade_negocio dun
```

```
          ON dun.unidade_negocio = s."Categoria"
        ON CONFLICT (data_id, tipo_orcamento_id, unidade_negocio_id)
        DO UPDATE SET
            valor_orcado = EXCLUDED.valor_orcado,
            batch_id     = EXCLUDED.batch_id,
            tempo_carga  = EXCLUDED.tempo_carga;
        """,

        #FATO CONTAS A RECEBER
        """
        INSERT INTO dw.fato_contas_receber (
            cliente_id, data_emissao_id, data_vencimento_id, status_id,
            valor, dias_atraso, batch_id, tempo_carga
        )
        SELECT
            dc.cliente_id,
            dte.data_id,
            dtv.data_id,
            ds.status_id,
            s."Valor",
            s."Dias_Atraso",
            s.batch_id,
            s.tempo_carga
        FROM (
            SELECT DISTINCT ON ("Cliente", "Data_Emissao", "Data_Vencimento",␣
↪"Status", "Valor", "Dias_Atraso")
                *
            FROM stg."Contas_Receber"
            ORDER BY
                "Cliente", "Data_Emissao", "Data_Vencimento", "Status",␣
↪"Valor", "Dias_Atraso",
                tempo_carga DESC NULLS LAST
        ) s
        JOIN dw.dim_cliente dc
          ON dc.cliente = s."Cliente"
        LEFT JOIN dw.dim_tempo dte
          ON dte.data = s."Data_Emissao"::date
        LEFT JOIN dw.dim_tempo dtv
          ON dtv.data = s."Data_Vencimento"::date
        LEFT JOIN dw.dim_status ds
          ON ds.status = s."Status"
        ON CONFLICT (cliente_id, data_emissao_id, data_vencimento_id,␣
↪status_id, valor, dias_atraso)
        DO UPDATE SET
            batch_id    = EXCLUDED.batch_id,
            tempo_carga = EXCLUDED.tempo_carga;
        """,
```

```
        #FATO METAS NEGÓCIO
        """
        INSERT INTO dw.fato_metas_negocio (
            unidade_negocio_id,
            margem_meta, crescimento_meta, ticket_medio_meta,
            batch_id, tempo_carga
        )
        SELECT
            dun.unidade_negocio_id,
            s."Margem_Meta",
            s."Crescimento_Meta",
            s."Ticket_Medio_Meta",
            s.batch_id,
            s.tempo_carga
        FROM (
            SELECT DISTINCT ON ("Unidade_Negocio")
                *
            FROM stg."Metas_Negocio"
            ORDER BY
                "Unidade_Negocio",
                tempo_carga DESC NULLS LAST
        ) s
        JOIN dw.dim_unidade_negocio dun
          ON dun.unidade_negocio = s."Unidade_Negocio"
        ON CONFLICT (unidade_negocio_id)
        DO UPDATE SET
            margem_meta       = EXCLUDED.margem_meta,
            crescimento_meta  = EXCLUDED.crescimento_meta,
            ticket_medio_meta = EXCLUDED.ticket_medio_meta,
            batch_id          = EXCLUDED.batch_id,
            tempo_carga       = EXCLUDED.tempo_carga;
        """
    ]

    with engine.begin() as conn:
        for sql in sqls:
            conn.execute(text(sql))
```

[13]:
```
#bloco pincipal

#leitura do arquivo base
arquivo_base = Path(r'C:\Users\Gabriel␣
 ↪Alef\Projeto\new\massa_dados_financeiro_BI.xlsx')

#leitura do arquivo como um objeto, para ler os metadados e abas
xls = pd.ExcelFile(arquivo_base)
```

```
#criação dos dataframes pelo nome da aba dentro do dicionário "abas"
abas = {}
for aba in xls.sheet_names:
    df = pd.read_excel(xls, sheet_name = aba)
    df = df.dropna(how = "all")
    df = df.dropna(axis = 1, how = "all")
    abas[aba] = df

#base normalizada
base_normalizada = {}
for nome_aba, df in abas.items():
    base_normalizada[nome_aba] = normalizar_base(df)

#informações de cada carga da rotina
batch_id = str(uuid.uuid4())
tempo_carga = datetime.now()
for df in base_normalizada.values():
    df["batch_id"] = batch_id
    df["tempo_carga"] = tempo_carga

#conexão com o banco de dados
host = "localhost"
Porta = 5432
Banco = "Alpha"
Usuario = "postgres"
Senha = "123456"
engine = create_engine(
    f"postgresql+psycopg2://{Usuario}:{quote_plus(Senha)}@{host}:{Porta}/
 ↪{Banco}"
)

#alimentação do staging
alimentar_staging(
    base_normalizada=base_normalizada,
    engine=engine,
    schema="stg",
    truncate_first=True,
    chunksize=10_000
)

#alimentação das tabelas fato e dimensão
alimenta_dim(engine)
alimenta_fato(engine)
```

[ ]: