**Populando o banco de dados autores**

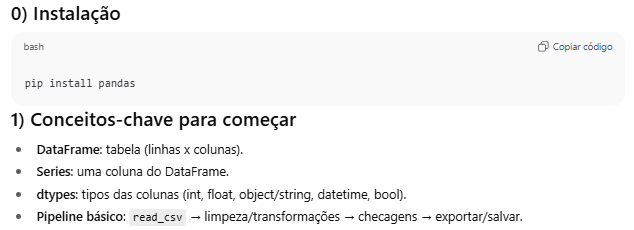
Crie a estrutura:  


Comandos como o popular só funcionam no local acima.

Depois que o popular.py estiver correto:

python manage.py popular --truncate # limpa e insere (bulk)

python manage.py popular --update # upsert (idempotente)



Crie um arquivo .py qualquer para treinar

Teste.py

import pandas as pd

#2. Lendo o CSV de autores

df = pd.read\_csv("population/autores.csv")   # cabeçalhos: nome,sobrenome,data\_nascimento,nacionalidade[,biografia]

print(df.head())                             # primeiras linhas

print(df.shape)                              # (linhas, colunas)

print(df.dtypes)                             # tipos das colunas

# Limpeza básica (o que SEMPRE aparece em planilhas)

# 3.1: Padronizar espaços e caixa

print('\n\n\nSem padrão: \n', df)

print('Padronizar espaços e caixa')

for col in ["nome", "sobrenome", "nacionalidade"]:

    df[col] = df[col].astype(str).str.strip()

print('\n\n\nCom padrão: \n',df)

# 3.2: Datas como datetime (parser seguro)

df["data\_nascimento"] = pd.to\_datetime(df["data\_nascimento"], errors="coerce", format="%Y-%m-%d")

print('\n\n\nDatas como datetime\n',df)

# 3.3: Corrigir capitalização

df["nacionalidade"] = df["nacionalidade"].str.capitalize()  # "Brasileira", "Portuguesa", etc.

print('\n\n\nCorrigir capitalização\n', df)

# 3.4: Remover linhas totalmente vazias (se houver)

df = df.dropna(how="all")

print('\n\n\nRemover linhas totalmente vazias (se houver)\n\n', df)

# 3.6: Preencher valores ausentes opcionais

if "biografia" not in df.columns:

    df["biografia"] = "---"

print('\n\n\nPreencher valores ausentes opcionais\n\n', df)

# Filtrar autores brasileiros

br = df[df["nacionalidade"] == "Brasileira"]

print('\n\n\nFiltrar autores brasileiros\n\n', br)

# Selecionar colunas

apenas\_nomes = df[["nome", "sobrenome"]]

print('\n\n\nSelecionar colunas\n\n', apenas\_nomes)

# Ordenar por data

df = df.sort\_values("data\_nascimento")

print('\n\n\nOrdenar por data\n\n', df)

# Contagens

print(df["nacionalidade"].value\_counts())

**POPULAR.PY**

import pandas as pd

from pathlib import Path

from django.core.management.base import BaseCommand, CommandError

from django.db import transaction

from api.models import Autor

class Command(BaseCommand):

    help = "Importa autores de population/autores.csv usando pandas (com limpeza e validação)."

    def add\_arguments(self, parser):

        parser.add\_argument("--arquivo", default=str(Path("population") / "autores.csv"))

        parser.add\_argument("--truncate", action="store\_true", help="Apaga todos os autores antes de importar")

        parser.add\_argument("--update", action="store\_true", help="Faz upsert (update\_or\_create) em vez de inserir em massa")

    @transaction.atomic

    # @transaction.atomic  👉 garante que tudo dentro do método será executado dentro de uma única transação de banco.

    # Se der erro no meio, o Django desfaz tudo (rollback).

    def handle(self, \*args, \*\*opts): # é o método principal que o Django executa quando você roda python manage.py popular.

        # \*args, \*\*opts: recebe os argumentos/flags passados na linha de comando (ex.: --truncate, --update).

        csv\_path = Path(opts["arquivo"]) # pega o caminho do CSV passado por argumento (ou o default definido).

        if not csv\_path.exists(): #Se o arquivo não existir, lança um CommandError → o Django mostra a mensagem de erro e interrompe.

            raise CommandError(f"Arquivo não encontrado: {csv\_path}")

        # 1) Ler CSV

        df = pd.read\_csv(csv\_path)

        # 2) Limpeza básica

        for col in ["nome", "sobrenome", "nacionalidade"]:

            if col in df.columns:

                df[col] = df[col].astype(str).str.strip()

            else:

                df[col] = ""

        if "biografia" not in df.columns:

            df["biografia"] = ""

        df["data\_nascimento"] = pd.to\_datetime(df.get("data\_nascimento"), errors="coerce", format="%Y-%m-%d")

        df["nacionalidade"] = df["nacionalidade"].str.capitalize()

        # 3) Remover vazios/duplicados

        df = df.dropna(how="all")

        df = df.drop\_duplicates(subset=["nome", "sobrenome", "data\_nascimento"], keep="first").reset\_index(drop=True)

        # 4) Validar obrigatórios

        obrigatorios = df["nome"].ne("") & df["sobrenome"].ne("") & df["data\_nascimento"].notna()

        invalidos = df[~obrigatorios] #pega os que não cumprem

        if not invalidos.empty: #Se houver inválidos → mostra aviso e remove do DataFrame.

            self.stdout.write(self.style.WARNING(f"Pulando {len(invalidos)} linha(s) inválida(s)."))

        df = df[obrigatorios]

        #Truncate (limpar tabela antes de importar)

        if opts["truncate"]:#Se o usuário passou --truncate, apaga todos os registros antes de importar.

            self.stdout.write(self.style.WARNING("Limpando tabela api\_autor..."))

            Autor.objects.all().delete()

        criados = 0

        atualizados = 0

        if opts["update"]:

            # UPSERT registro a registro (mais didático, porém mais lento)

            for row in df.itertuples(index=False):

                # df.itertuples() é um método do pandas DataFrame.

                # Ele percorre cada linha e devolve um objeto parecido com uma tupla, chamado namedtuple.

                # Cada campo da linha vira um atributo desse objeto.

                # O parâmetro:

                # index=False → não inclui a coluna de índice (0,1,2...) no resultado.

                # Assim, você só recebe as colunas do CSV.

                obj, created = Autor.objects.update\_or\_create(

                    nome=row.nome,

                    sobrenome=row.sobrenome,

                    data\_nascimento=row.data\_nascimento.date(),

                    defaults={

                        "nacionalidade": row.nacionalidade or None,

                        "biografia": (row.biografia or "").strip() or None,

                    },

                )

                    # Retorno:

                    #  obj → o autor encontrado/criado.

                    #  created → True se foi um novo autor, False se apenas atualizou um existente.

                    # Suponha que já existe no banco:

                    #   Autor(nome="Jorge", sobrenome="Amado", data\_nascimento="1912-08-10", nacionalidade="Brasileira", biografia="...")

                    # Se no CSV vem:

                    #   Jorge,Amado,1912-08-10,Brasileira,Autor famoso

                    # O Django encontra esse autor pelo trio (nome, sobrenome, data\_nascimento).

                    #    Atualiza biografia para "Autor famoso".

                    #    created = False.

                if created:

                    criados += 1

                else:

                    atualizados += 1

        else:

            # Inserção em massa (rápida)

            buffer = []

            for row in df.itertuples(index=False):

                buffer.append(Autor(

                    nome=row.nome,

                    sobrenome=row.sobrenome,

                    data\_nascimento=row.data\_nascimento.date(),

                    nacionalidade=row.nacionalidade or None,

                    biografia=(row.biografia or "").strip() or None,

                ))

            Autor.objects.bulk\_create(buffer, ignore\_conflicts=True)

            criados = len(buffer)

        msg = f"Concluído. Criados: {criados}"

        if opts["update"]:

            msg += f" | Atualizados: {atualizados}"

        self.stdout.write(self.style.SUCCESS(msg))

# python manage.py popular --truncate     # limpa e insere (bulk)

# python manage.py popular --update       # upsert (idempotente)