```
import ssl
import string
from datetime import datetime, timedelta
from typing import Dict, List, Optional, Set, Tuple

import nltk
import requests
from airflow import DAG
from airflow.operators.python_operator import PythonOperator
from airflow.providers.postgres.hooks.postgres import PostgresHook
from bs4 import BeautifulSoup
from nltk.corpus import stopwords

from urllib.parse import urlparse, unquote


nltk.download("punkt")
nltk.download("stopwords")

from airflow.models import Variable

try:
    _create_unverified_https_context = ssl._create_unverified_context
except AttributeError:
    pass
else:
    ssl._create_default_https_context = _create_unverified_https_context

default_args = {
    "owner": "airflow",
    "depends_on_past": False,
    "start_date": datetime(2023, 5, 2),
    "email": ["airflow@example.com"],
    "email_on_failure": False,
    "email_on_retry": False,
    "retries": 1,
    "retry_delay": timedelta(minutes=5),
}

dag = DAG(
    "scraper_amazon",
    default_args=default_args,
    description="Classificador de produtos da Amazon",
    schedule_interval="0 21 * * *",
)


def remove_punctuation(text: str) -> str:
    """
    Remove punctuation from the given text.

    Args:
        text (str): The input text.
```

```
    Returns:
        str: The text without punctuation.
    """
    translator = str.maketrans("", "", string.punctuation)
    return text.translate(translator)



def tokenize_and_lower(text: str) -> Set[str]:
    """
    Tokenize the input text and convert it to lowercase.

    Args:
        text (str): The input text.

    Returns:
        Set[str]: A set of unique lowercase tokens.
    """
    cleaned_text = remove_punctuation(text)
    tokens = nltk.word_tokenize(cleaned_text.lower())
    return set(tokens)


def calculate_min_intersection(query: str, percentage: float = 50.0) ->
int:
    """
    Calculate the minimum intersection value based on the input query and
percentage.

    Args:
        query (str): The input query.
        percentage (float, optional): The percentage to calculate the
minimum intersection. Defaults to 50.0.

    Returns:
        int: The minimum intersection value.
    """
    tokens = nltk.word_tokenize(query)
    return int(len(tokens) * (percentage / 100))


def find_most_similar(
    query: str, products: List[Dict[str, str]]
) -> Optional[Dict[str, str]]:
    query_tokens = tokenize_and_lower(query)
    max_intersection = 0
    most_similar_product = None
    min_intersection = calculate_min_intersection(query)

    for product in products:
        try:
            product_tokens = tokenize_and_lower(product["description"])
            intersection = len(query_tokens.intersection(product_tokens))
```

```
            if intersection >= min_intersection and intersection >
max_intersection:
                max_intersection = intersection
                most_similar_product = {
                    "url": product["url"],
                    "description": product["description"],
                }

        except Exception as error:
            print(f"Error processing product: {error}")
            continue

    return most_similar_product


def amazon_scraper(query: str) -> Optional[str]:
    SEARCH_PRODUCTS_URL = "https://www.amazon.com.br/s?k="

    proxy = {"http": Variable.get("PROXY")}

    headers = {
        "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.110
Safari/537.36",
        "Accept-Language": "en-US,en;q=0.5",
    }

    url = f"{SEARCH_PRODUCTS_URL}{query.replace(' ', '+')}"
    try:
        response = requests.get(url, headers=headers, proxies=proxy)
    except Exception as error:
        print("Erro ao fazer requisição:", error)
        return None

    soup = BeautifulSoup(response.content, "html.parser")
    product_list = soup.find_all(
        "div", class_="s-main-slot s-result-list s-search-results sg-row"
    )

    products = []

    for product_section in product_list:
        try:
            product = product_section.find(
                "div", class_="a-section aok-relative s-image-square-
aspect"
            )
            descricao = product.find("img", class_="s-image")["alt"]
            link_section = product_section.find(
                "a", class_="a-link-normal s-no-outline"
            )
            link = f"https://www.amazon.com.br{link_section['href']}"
            # img_link = product.find("img", class_="s-image")["src"]
            # sku = link.split("/dp/")[1].split("/")[0]
```

```
            products.append({"url": link, "description": descricao})

        except Exception as error:
            print("Erro ao buscar produto:", error)
            continue

    if most_similar_product := find_most_similar(query, products):
        print("Produto mais similar encontrado:")
        print(most_similar_product["url"])
        return most_similar_product["url"]
    else:
        print("Nenhum produto similar encontrado.")
        return None


####
def get_connection():
    """
    Get the PostgresHook connection object.

    Returns:
        Connection object.
    """
    return
PostgresHook(postgres_conn_id=Variable.get("POSTGRES_DB")).get_conn()


def fetch_schemas() -> List[str]:
    connection = get_connection()
    cursor = connection.cursor()
    cursor.execute(
        "SELECT schema_name FROM information_schema.schemata WHERE
schema_name NOT IN ('public', 'information_schema', 'pg_catalog',
'pg_toast', 'sku');"
    )
    # Transforma a lista de tuplas em uma lista de strings
    return [result[0] for result in cursor.fetchall()]



def fetch_products(schema: str, id_concorrente) -> List[Tuple]:
    connection = get_connection()
    cursor = connection.cursor()
    cursor.execute(
        f"SELECT * FROM {schema}.produto_novo WHERE id_concorrente =
{id_concorrente};"
    )
    return cursor.fetchall()


def insert_sku(schema: str, data: Tuple):
    connection = get_connection()
    cursor = connection.cursor()
```

```
    cursor.execute(
        f"INSERT INTO {schema}.sku (ean, cod_ref, id_loja, slug) VALUES
(%s, %s, %s, %s)",
        data,
    )
    connection.commit()


def insert_product_not_found(produto: dict, schema: str):
    connection = get_connection()
    cursor = connection.cursor()
    cursor.execute(
        f"INSERT INTO {schema}.produto_nao_encontrado (ean, sku, produto,
departamento, categoria, marca, id_concorrente, nome_concorrente) VALUES
(%s, %s, %s, %s, %s, %s, %s, %s)",
        (
            produto["ean"],
            produto["sku"],
            produto["produto"],
            produto["departamento"],
            produto["categoria"],
            produto["marca"],
            produto["id_concorrente"],
            produto["nome_concorrente"],
        ),
    )
    connection.commit()
    close_connection(connection)


def delete_sku(schema: str, id: str):
    connection = get_connection()
    cursor = connection.cursor()
    cursor.execute(f"DELETE FROM {schema}.produto_novo WHERE id = %s",
(id,))
    connection.commit()


def close_connection():
    connection = get_connection()
    connection.close()


def extract_product_sku(url: str) -> str:
    parsed_url = urlparse(unquote(url))
    path_components = parsed_url.path.strip("/").split("/")
    if "dp" in path_components:
        dp_index = path_components.index("dp")
        if dp_index + 1 < len(path_components):
            return path_components[dp_index + 1]


def executar_consulta():
    try:
```

```
        schemas = fetch_schemas()
        print(f"Schemas: {schemas}")

        for schema_name in schemas:
            try:
                produtos = fetch_products(schema_name, 156)
                if not produtos:
                    print("Nenhum resultado encontrado.")
                    continue

                for produto in produtos:
                    print("====================================")
                    print(produto[3])
                    slug = amazon_scraper(produto[3])
                    print(slug)
                    print("====================================")
                    if slug is None:
                        insert_product_not_found(produto, schema_name)
                        delete_sku(schema_name, produto[0])
                    try:
                        sku = extract_product_sku(slug)
                        insert_sku(
                            schema_name,
                            (
                                produto[1],
                                sku,
                                produto[7],
                                slug,
                            ),
                        )
                        delete_sku(schema_name, produto[0])
                        print("SKU inserido com sucesso:", sku)
                        print("====================================")
                    except Exception as error:
                        print("Erro ao inserir sku:", error)
                        print(error)
                        print("====================================")
            except Exception as schema_error:
                print(f"Erro ao processar o schema {schema_name}:",
schema_error)
                print("====================================")
    except Exception as error:
        print("Erro ao executar consulta:", error)


run_query = PythonOperator(
    task_id="run_query",
    python_callable=executar_consulta,
    dag=dag,
)

run_query
```

```
def run_amazon_scraper_local(query: str) -> None:
    """
    Run the Amazon scraper locally with the given query.

    Args:
        query (str): The input query.
    """
    print("Running Amazon scraper locally...")
    print("Query:", query)

    SEARCH_PRODUCTS_URL = "https://www.amazon.com.br/s?k="
    PROXY = "http://Eiprice-cc-any:DQSXomtV7qri@gw.ntnt.io:5959"
    HEADERS = {
        "User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.110
Safari/537.36",
        "Accept-Language": "en-US,en;q=0.5",
    }

    url = f"{SEARCH_PRODUCTS_URL}{query.replace(' ', '+')}"
    response = requests.get(url, headers=HEADERS, proxies={PROXY})

    soup = BeautifulSoup(response.content, "html.parser")
    product_list = soup.find_all(
        "div", class_="s-main-slot s-result-list s-search-results sg-row"
    )

    products = []

    for product_section in product_list:
        try:
            product = product_section.find(
                "div", class_="a-section aok-relative s-image-square-
aspect"
            )
            descricao = product.find("img", class_="s-image")["alt"]
            link_section = product_section.find(
                "a", class_="a-link-normal s-no-outline"
            )
            link = f"https://www.amazon.com.br{link_section['href']}"

            products.append({"url": link, "description": descricao})

        except Exception as error:
            print("Erro ao buscar produto:", error)
            continue

    if most_similar_product := find_most_similar(query, products):
        print("Produto mais similar encontrado:")
        print(most_similar_product["url"])
    else:
        print("Nenhum produto similar encontrado.")
```

```
if __name__ == "__main__":
    sample_query = "COMPUTADOR GAMER 128GB RAM SSD HD 1TB NOVO"
    run_amazon_scraper_local(sample_query)
```