dag_etl_taskflow.py

```
import os
import sqlalchemy
from datetime import datetime, timedelta
import numpy as np
import pandas as pd
from airflow import DAG
from airflow.decorators import task
from airflow.operators.bash import BashOperator
from airflow.providers.amazon.aws.hooks.s3 import S3Hook
from airflow.providers.amazon.aws.sensors.s3 import S3KeySensor
from airflow.providers.postgres.hooks.postgres import PostgresHook
from airflow.providers.postgres.operators.postgres import PostgresOperator
from airflow.providers.amazon.aws.operators.s3_delete_objects import S3DeleteObjectsOperator

raw_key = 'supermarket_sales.csv'
raw_bucket = 'raw'
raw_local_path = 'data'
file_new_name = 'downloaded_from_minio.csv'
postgres_hook = PostgresHook(postgres_conn_id='postgres_conn')
default_args = {
    'owner': 'Victor',
    'retries': 5,
    'retry_delay': timedelta(minutes=10)
}

with DAG(
        dag_id='dag_etl',
        description='стартует, когда в бакет попадает csv файл',
        start_date=datetime(2023, 4, 27, 0),
        schedule_interval='@daily',
        default_args=default_args
) as dag:
    task_s3_sensor = S3KeySensor(
        task_id='sensor_s3_obj',
        bucket_name=raw_bucket,
        bucket_key=raw_key,
        aws_conn_id='minio_conn',
        mode='poke',
        poke_interval=5,
        timeout=60 # Тут надо выставить 24*60*60 - т.е. все сутки, НО комп сильно устает
    )
    task_create_tables = PostgresOperator(
        task_id='create_nds_tables_if_not_exists',
        postgres_conn_id='postgres_conn',
        sql="""
            CREATE SCHEMA IF NOT EXISTS nds;
            CREATE SCHEMA IF NOT EXISTS stage;

            SET search_path TO nds;

            --// создаем таблицу с ветками //--
            CREATE TABLE IF NOT EXISTS dim_branch(
            id SERIAL PRIMARY KEY,
            branch VARCHAR(100) NOT NULL);

            --// создаем таблицу с городами //--
            CREATE TABLE IF NOT EXISTS dim_city(
            id SERIAL PRIMARY KEY,
            city VARCHAR(100) NOT NULL);

            --// создаем таблицу с типами клиентов //--
            CREATE TABLE IF NOT EXISTS dim_customer_type(
            id SERIAL PRIMARY KEY,
            customer_type VARCHAR(200) NOT NULL);

            --// создаем таблицу с гендерами //--
            CREATE TABLE IF NOT EXISTS dim_gender(
            id SERIAL PRIMARY KEY,
            gender VARCHAR(200) NOT NULL);

            --// создаем таблицу с продуктовыми линейками //--
            CREATE TABLE IF NOT EXISTS dim_product_line(
            id SERIAL PRIMARY KEY,
            product_line VARCHAR(200) NOT NULL);

            --// создаем таблицу с видами оплат //--
            CREATE TABLE IF NOT EXISTS dim_payment(
            id SERIAL PRIMARY KEY,
            payment VARCHAR(100) NOT NULL);

            --// создаем таблицу с фактами //--
            CREATE TABLE IF NOT EXISTS fact_sales(
            invoice_id VARCHAR(15) PRIMARY KEY,
            branch INT NOT NULL REFERENCES dim_branch(id),
            city INT NOT NULL REFERENCES dim_city(id),
            customer_type INT NOT NULL REFERENCES dim_customer_type(id),
            gender INT NOT NULL REFERENCES dim_gender(id),
            product_line INT NOT NULL REFERENCES dim_product_line(id),
            unit_price DOUBLE PRECISION,
            quantity DOUBLE PRECISION,
            "tax_5%" DOUBLE PRECISION,
            total DOUBLE PRECISION,
            date DATE NOT NULL,
            time TIME NOT NULL,
            payment INT NOT NULL REFERENCES dim_payment(id),
            cogs DOUBLE PRECISION,
            gross_margin_percentage DOUBLE PRECISION,
            gross_income DOUBLE PRECISION,
            rating DOUBLE PRECISION);
        """
    )
```

```
task_update_dims = PostgresOperator(
    task_id='update_dim_tables',
    postgres_conn_id='postgres_conn',
    sql="""
        SET search_path TO nds;

        --// Обновляем таблицы в nds сырыми таблицами из stage //--
        INSERT INTO dim_branch (branch)
        (SELECT branch FROM stage.dim_branch WHERE branch NOT IN (SELECT branch FROM dim_branch));
        INSERT INTO dim_city (city)
        (SELECT city FROM stage.dim_city WHERE city NOT IN (SELECT city FROM dim_city));
        INSERT INTO dim_customer_type (customer_type)
        (SELECT customer_type FROM stage.dim_customer_type WHERE customer_type NOT IN (SELECT customer_type FROM dim_customer_type));
        INSERT INTO dim_gender (gender)
        (SELECT gender FROM stage.dim_gender WHERE gender NOT IN (SELECT gender FROM dim_gender));
        INSERT INTO dim_product_line (product_line)
        (SELECT product_line FROM stage.dim_product_line WHERE product_line NOT IN (SELECT product_line FROM dim_product_line));
        INSERT INTO dim_payment (payment)
        (SELECT payment FROM stage.dim_payment WHERE payment NOT IN (SELECT payment FROM dim_payment));

        """
)
task_update_fact = PostgresOperator(
    task_id='update_fact_table',
    postgres_conn_id='postgres_conn',
    sql="""
        SET search_path TO nds;

        --// Обновляем таблицу с фактом свежей таблицей с фактами из stage //--
        INSERT INTO fact_sales (invoice_id, branch, city, customer_type, gender,
                                product_line, unit_price, quantity, "tax_5%", total, date,
                                time, payment, cogs, gross_margin_percentage, gross_income, rating)
        (SELECT
                invoice_id, branch, city, customer_type, gender,
                product_line, unit_price, quantity, "tax_5%", total, date::date,
                time, payment, cogs, gross_margin_percentage, gross_income, rating
        FROM stage.fact_sales WHERE invoice_id NOT IN (SELECT invoice_id FROM fact_sales));
        """
)
task_delete_s3_obj = S3DeleteObjectsOperator(
    task_id='delete_s3_obj',
    bucket=raw_bucket,
    keys=raw_key,
    aws_conn_id='minio_conn',
    trigger_rule='none_failed_min_one_success'
)
task_clear_data_directory = BashOperator(
    task_id='clear_data_directory',
    bash_command='rm -rf ${pwd}data/*'
)


@task
def extract_from_s3(bucket_key, bucket_name, local_path):
    hook = S3Hook('minio_conn')
    file_name = hook.download_file(bucket_key, bucket_name, local_path)
    return file_name


@task
def rename_extracted_file(file_name, file_new_name):
    downloaded_file_path = '/'.join(file_name.split('/')[:-1])
    os.rename(src=file_name, dst=f"{downloaded_file_path}/{file_new_name}")
    return downloaded_file_path


@task
def dim_branch(downloaded_file_path, file_new_name):
    df = pd.read_csv(f"{downloaded_file_path}/{file_new_name}")
    df.columns = [column_title.lower().replace(' ', '_') for column_title in df.columns]
    branch = pd.Series(df['branch'].unique(), name='branch')
    branch_df = pd.DataFrame(branch)
    hook = PostgresHook(postgres_conn_id='postgres_conn')
    branch_df.to_sql('dim_branch', hook.get_sqlalchemy_engine(), schema='stage', if_exists='replace')


@task
def dim_city(downloaded_file_path, file_new_name):
    df = pd.read_csv(f"{downloaded_file_path}/{file_new_name}")
    df.columns = [column_title.lower().replace(' ', '_') for column_title in df.columns]
    city = pd.Series(df['city'].unique(), name='city')
    city_df = pd.DataFrame(city)
    hook = PostgresHook(postgres_conn_id='postgres_conn')
    city_df.to_sql('dim_city', hook.get_sqlalchemy_engine(), schema='stage', if_exists='replace')


@task
def dim_customer_type(downloaded_file_path, file_new_name):
    df = pd.read_csv(f"{downloaded_file_path}/{file_new_name}")
    df.columns = [column_title.lower().replace(' ', '_') for column_title in df.columns]
    customer_type = pd.Series(df['customer_type'].unique(), name='customer_type')
    customer_type_df = pd.DataFrame(customer_type)
    hook = PostgresHook(postgres_conn_id='postgres_conn')
    customer_type_df.to_sql('dim_customer_type', hook.get_sqlalchemy_engine(), schema='stage', if_exists='replace')


@task
def dim_gender(downloaded_file_path, file_new_name):
    df = pd.read_csv(f"{downloaded_file_path}/{file_new_name}")
    df.columns = [column_title.lower().replace(' ', '_') for column_title in df.columns]
    gender = pd.Series(df['gender'].unique(), name='gender')
    gender_df = pd.DataFrame(gender)
    hook = PostgresHook(postgres_conn_id='postgres_conn')
    gender_df.to_sql('dim_gender', hook.get_sqlalchemy_engine(), schema='stage', if_exists='replace')
```

```
@task
def dim_product_line(downloaded_file_path, file_new_name):
    df = pd.read_csv(f"{downloaded_file_path}/{file_new_name}")
    df.columns = [column_title.lower().replace(' ', '_') for column_title in df.columns]
    product_line = pd.Series(df['product_line'].unique(), name='product_line')
    product_line_df = pd.DataFrame(product_line)
    hook = PostgresHook(postgres_conn_id='postgres_conn')
    product_line_df.to_sql('dim_product_line', hook.get_sqlalchemy_engine(), schema='stage', if_exists='replace')


@task
def dim_payment(downloaded_file_path, file_new_name):
    df = pd.read_csv(f"{downloaded_file_path}/{file_new_name}")
    df.columns = [column_title.lower().replace(' ', '_') for column_title in df.columns]
    payment = pd.Series(df['payment'].unique(), name='payment')
    payment_df = pd.DataFrame(payment)
    hook = PostgresHook(postgres_conn_id='postgres_conn')
    payment_df.to_sql('dim_payment', hook.get_sqlalchemy_engine(), schema='stage', if_exists='replace')


@task
def dim_time():
    """
    Создаем таблицу с временем и признаками времени
    и заливаем сразу в nds
    """
    time_range = pd.date_range(start="00:00", end="23:59", freq="1min")
    df = pd.DataFrame(pd.Series(time_range.strftime("%H:%M:%S"), name='time'))
    day_part = 'day_part'
    df.loc[(df['time'] >= '00:00:00') & (df['time'] < '06:00:00'), day_part] = 'night'
    df.loc[(df['time'] >= '06:00:00') & (df['time'] < '11:00:00'), day_part] = 'morning'
    df.loc[(df['time'] >= '11:00:00') & (df['time'] < '17:00:00'), day_part] = 'noon'
    df.loc[(df['time'] >= '17:00:00') & (df['time'] < '22:00:00'), day_part] = 'evening'
    df.loc[(df['time'] >= '22:00:00') & (df['time'] < '24:00:00'), day_part] = 'night'
    hook = PostgresHook(postgres_conn_id='postgres_conn')
    df.to_sql('dim_time',
              hook.get_sqlalchemy_engine(),
              schema='nds',
              if_exists='replace',
              index=False,
              dtype={'time': sqlalchemy.types.TIME()})


@task
def dim_date():
    """
    Создаем таблицу с датами и признаками дат
    и заливаем сразу в nds
    """
    df = pd.DataFrame(pd.date_range(start="2019-01-01", end="2099-12-31"), columns=['date'])
    df['week_of_year'] = df['date'].dt.isocalendar().week
    df['week_start'] = df['date'].dt.to_period('W-SUN').dt.start_time
    df['day_of_week'] = df['date'].dt.dayofweek + 1
    df['month_number'] = df['date'].dt.month
    df['month_name'] = pd.to_datetime(df['date'], format='%m').dt.month_name()
    df['quarter'] = df['date'].dt.quarter
    df['year'] = df['date'].dt.year
    df['season'] = np.where(df['month_number'].isin([12, 1, 2]), 'winter', 'spring')
    df['season'] = np.where(df['month_number'].isin([6, 7, 8]), 'summer', df['season'])
    df['season'] = np.where(df['month_number'].isin([9, 10, 11]), 'fall', df['season'])
    df['date'] = df['date'].dt.strftime('%Y-%m-%d')
    df['week_start'] = df['week_start'].dt.strftime('%Y-%m-%d')
    hook = PostgresHook(postgres_conn_id='postgres_conn')
    df.to_sql('dim_date',
              hook.get_sqlalchemy_engine(),
              schema='nds',
              if_exists='replace',
              index=False,
              dtype={'date': sqlalchemy.types.Date(),
                     'week_start': sqlalchemy.types.Date()})


@task
def fact_stage(downloaded_file_path, file_new_name):
    """
    Забираем из базы обновленные измерения и их ключи.
    Ключи прежних загрузок остаются неизменными.
    Преобразуем эти пары в словари и меняем в таблице фактов значения на ключи.
    Заливаем пока в stage
    """
    hook = PostgresHook(postgres_conn_id='postgres_conn')
    conn = hook.get_conn()
    cursor = conn.cursor()
    cursor.execute("""SET search_path TO nds; SELECT * FROM dim_branch;""")
    branch = dict(cursor.fetchall())
    cursor.execute("""SET search_path TO nds; SELECT * FROM dim_city;""")
    city = dict(cursor.fetchall())
    cursor.execute("""SET search_path TO nds; SELECT * FROM dim_customer_type;""")
    customer_type = dict(cursor.fetchall())
    cursor.execute("""SET search_path TO nds; SELECT * FROM dim_gender;""")
    gender = dict(cursor.fetchall())
    cursor.execute("""SET search_path TO nds; SELECT * FROM dim_product_line;""")
    product_line = dict(cursor.fetchall())
    cursor.execute("""SET search_path TO nds; SELECT * FROM dim_payment;""")
    payment = dict(cursor.fetchall())
    cursor.close()
    conn.close()
    df = pd.read_csv(f"{downloaded_file_path}/{file_new_name}")
    df.columns = [column_title.lower().replace(' ', '_') for column_title in df.columns]
    df['date'] = pd.to_datetime(df['date'], format="%m/%d/%Y")
    df['time'] = pd.to_datetime(df['time'], format="%H:%M").dt.time
    df['branch'] = df['branch'].map({v: k for k, v in branch.items()})
```

```
    df['city'] = df['city'].map({v: k for k, v in city.items()})
    df['customer_type'] = df['customer_type'].map({v: k for k, v in customer_type.items()})
    df['gender'] = df['gender'].map({v: k for k, v in gender.items()})
    df['product_line'] = df['product_line'].map({v: k for k, v in product_line.items()})
    df['payment'] = df['payment'].map({v: k for k, v in payment.items()})
    df.to_sql('fact_sales', hook.get_sqlalchemy_engine(), schema='stage', if_exists='replace', index=False)


file_name = extract_from_s3(raw_key, raw_bucket, raw_local_path)
downloaded_file_path = rename_extracted_file(file_name, file_new_name)
branch = dim_branch(downloaded_file_path, file_new_name)
city = dim_city(downloaded_file_path, file_new_name)
customer_type = dim_customer_type(downloaded_file_path, file_new_name)
gender = dim_gender(downloaded_file_path, file_new_name)
product_line = dim_product_line(downloaded_file_path, file_new_name)
payment = dim_payment(downloaded_file_path, file_new_name)
time_ = dim_time()
date_ = dim_date()
fact_sales = fact_stage(downloaded_file_path, file_new_name)

task_s3_sensor >> file_name >> downloaded_file_path >> [
    branch, city, customer_type, gender, product_line, payment
] >> task_update_dims
task_update_dims >> fact_sales >> task_update_fact >> task_clear_data_directory
task_s3_sensor >> task_create_tables >> [time_, date_]
downloaded_file_path >> task_delete_s3_obj
```