Diego González Cantú Nov 24 2023

Evaluación Practica

**Introducción:**

El propósito de este proyecto es demostrar de manera practica mis conocimientos y habilidades para mi puesto con la consultora. En búsqueda de cubrir el rango completo de áreas en las que se busca evaluar decidí enfocar el desarrollo en cubrir análisis de datos fuente, modelado de datos (modelo conceptual y físico), creación de base de datos, creación de objetos en la base de datos, extracción y carga, lenguaje de consulta de datos, análisis y orquestación.

Se desarrollo el proyecto de manera local y se complementa con una propuesta de pasos a alto nivel de cómo migrar el desarrollo hacia Google Cloud.

**Descripción del Proyecto:**

Para este proyecto se tomo el caso practico imaginando un escenario donde los insumos proporcionados serán cargados a un directorio. El flujo debe leer los archivos de esta fuente para procesar, limpiar, normalizar y guardar como archivos limpios. Esto con el fin de ser insertados a la base de datos del equipo de ventas alojada en un servidos de MySQL. Asimismo, la data almacenada deber ser archivada en un object storage como backup historico.

Posteriormente se deben generar vistas que usara el cliente para alimentar un reporte gerencial. El flujo corre de manera periódica orquestado por una instancia de Airflow local.

**Arquitectura General:**

**Local:**

Directorio 'raw' para archivos crudos.

Scripts de limpieza que generan archivos en 'CleanFiles'.

Script de validación e inserción en MySQL.

Modelo conceptual y físico de la base de datos.

Vista de análisis y DAG local para orquestar el flujo.

Script de validación, carga y logs

RAW Files

**Modelado**

Airflow DAG (orquestación)

Vista con kpis

Scripts limpieza de datos

Pon

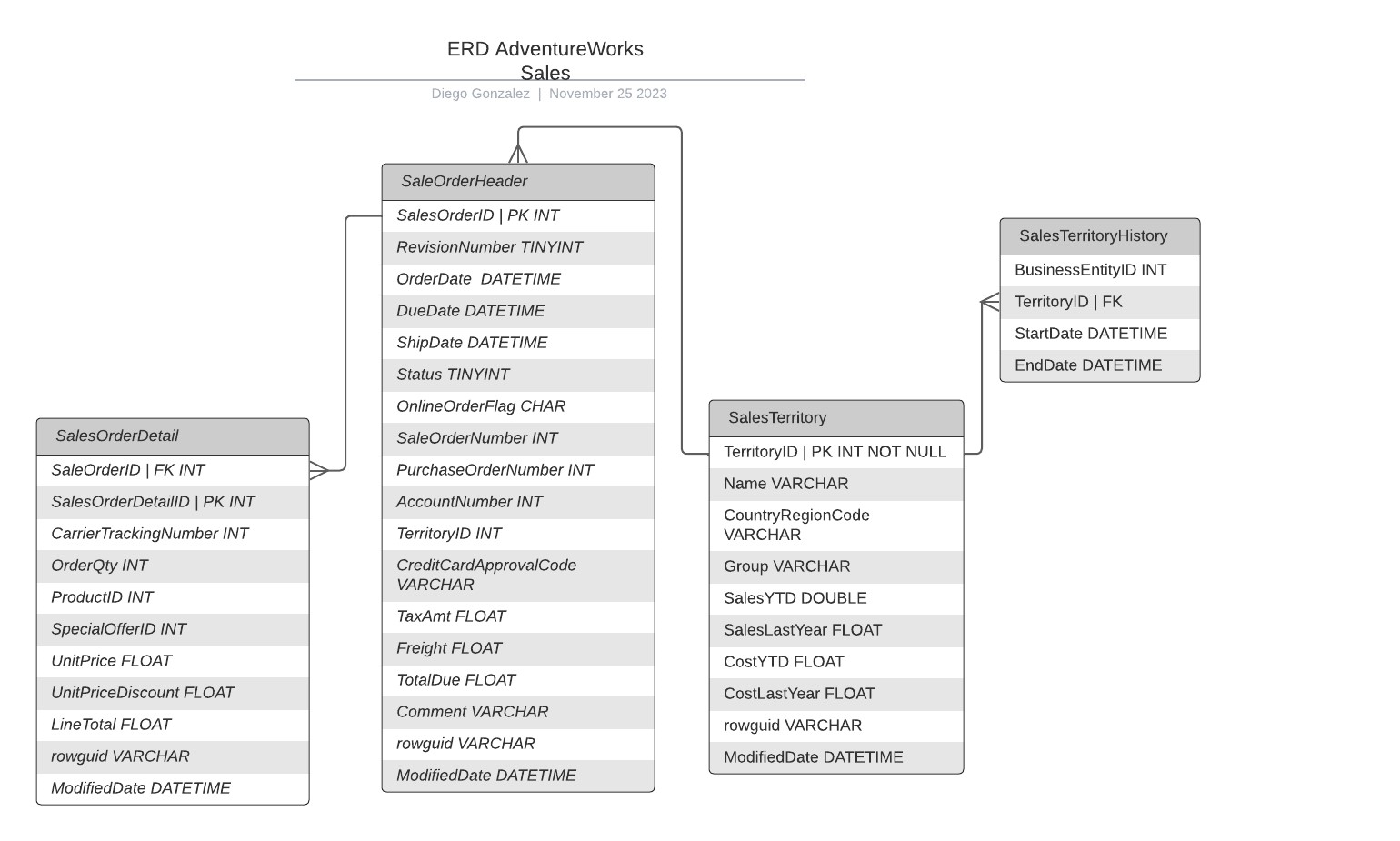
MySQL

Clean Files

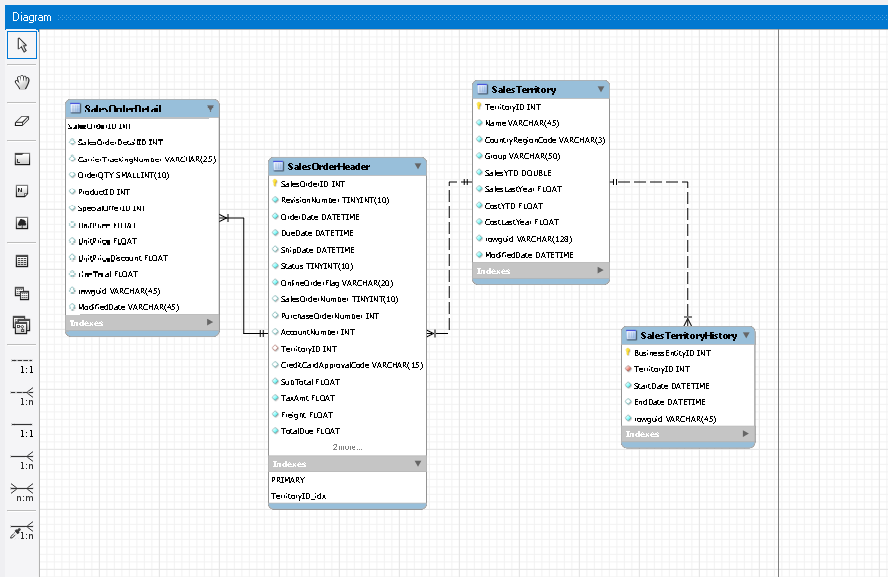
Para acotar el modelo de datos a algo alcanzable en el tiempo proporcionado se tomaron 4 tablas a modelar: SalesOrderdHeader (Transaction Fact Table), SalesOrderDetail (Transaction Fact Table Outrigger), SalesTerritory (Dimension Table), SalesTerritoryHistory (Slowly changing dimensión fact table).

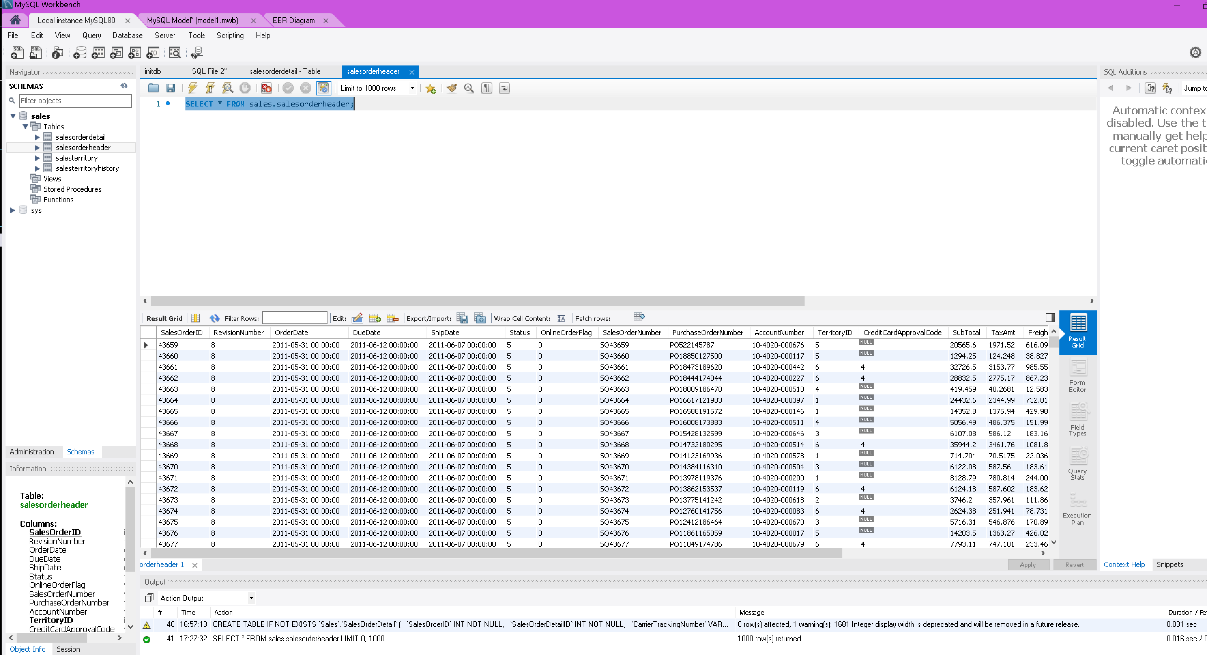
Se analizo el tipo de dato y el script de mssql server que se nos proporciono para generar el modelo lógico y el modelo físico adjuntado.

**Modelo Conceptual:**



**Modelo Fisico:**





**Liberación a Producción ( A alto nivel)**

*Stack: Cloud Storage, Cloud Storage Transfer System,Cloud SQL, Dataflow, Cloud Composer.*

**Configuración de Google Cloud Project:**

Crear una cuenta de Google Cloud, asignar roles y privilegios mediante I&AM.

Habilitar las APIs necesarias: Cloud SQL, Cloud Storage, Dataflow y Cloud Composer.

**Almacenamiento en Cloud Storage:**

Crear un bucket de Cloud Storage para almacenar los archivos raw y clean en lugar de los directorios locales. El bucket de raw debe ser migrado a un **coldline** para ser archivado y que sirva de backup optimizando costos de almacenamiento.

**Transferencia de Archivos a Cloud Storage:**

Cargar o transferir los archivos al bucket de raw. Se puede utilizar el Cloud Storage Transfer System para automatizar y calendarizar.

**Configuración de Cloud SQL:**

Crear una instancia de Cloud SQL del tipo MySQL en la región más óptima para almacenar la base de datos MySQL. Establecer permisos y conexiones para que pueda ser accesible con los otros servicios.

Importar el modelo físico de base de datos y realiza las configuraciones necesarias para la nube.

**Adaptación de Scripts:**

Modificar los scripts locales para leer y escribir desde/hacia Cloud Storage en lugar de rutas locales.

Actualizar las conexiones a la base de datos para apuntar a la instancia de Cloud SQL.

**Configuración de Dataflow:**

Desarrollar un flujo tipo batch en DataFlow que refleje la lógica de limpieza, validación y carga. Para esto se puede considerar usar **Apache Beam** si la volumetría amerita un procesamiento distribuido. El flujo debe llevar la lógica plasmada en el DAG.

**Configuración de Cloud Composer (airflow):**

Crea un entorno de Cloud Composer para orquestar tus flujos de trabajo.

Configura tareas de Composer para ejecutar el flujo de trabajo de Dataflow.

**Optimización y Ajuste:**

Evalúa el rendimiento y los costos en Google Cloud y realiza ajustes según sea necesario.

**Samples de Codigo:**

**Script creación base de datos (DDL):**

SET @OLD\_UNIQUE\_CHECKS=@@UNIQUE\_CHECKS, UNIQUE\_CHECKS=0;

SET @OLD\_FOREIGN\_KEY\_CHECKS=@@FOREIGN\_KEY\_CHECKS, FOREIGN\_KEY\_CHECKS=0;

SET @OLD\_SQL\_MODE=@@SQL\_MODE, SQL\_MODE='ONLY\_FULL\_GROUP\_BY,STRICT\_TRANS\_TABLES,NO\_ZERO\_IN\_DATE,NO\_ZERO\_DATE,ERROR\_FOR\_DIVISION\_BY\_ZERO,NO\_ENGINE\_SUBSTITUTION';

-- -----------------------------------------------------

-- Schema Sales

-- -----------------------------------------------------

CREATE SCHEMA IF NOT EXISTS `Sales` DEFAULT CHARACTER SET utf8 ;

USE `Sales` ;

-- -----------------------------------------------------

-- Table `Sales`.`SalesTerritory`

-- -----------------------------------------------------

DROP TABLE IF EXISTS `Sales`.`SalesTerritory` ;

CREATE TABLE IF NOT EXISTS `Sales`.`SalesTerritory` (

`TerritoryID` INT NOT NULL,

`Name` VARCHAR(45) NOT NULL,

`CountryRegionCode` VARCHAR(3) NOT NULL,

`Group` VARCHAR(50) NOT NULL,

`SalesYTD` DOUBLE NOT NULL DEFAULT 0.0,

`SalesLastYear` FLOAT NOT NULL,

`CostYTD` FLOAT NOT NULL,

`CostLastYear` FLOAT NOT NULL,

`rowguid` VARCHAR(128) NOT NULL,

`ModifiedDate` DATETIME NOT NULL,

PRIMARY KEY (`TerritoryID`))

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `Sales`.`SalesOrderHeader`

-- -----------------------------------------------------

DROP TABLE IF EXISTS `Sales`.`SalesOrderHeader` ;

CREATE TABLE IF NOT EXISTS `Sales`.`SalesOrderHeader` (

`SalesOrderID` INT NOT NULL,

`RevisionNumber` TINYINT(10) NOT NULL,

`OrderDate` DATETIME NOT NULL,

`DueDate` DATETIME NOT NULL,

`ShipDate` DATETIME NULL,

`Status` TINYINT(10) NOT NULL,

`OnlineOrderFlag` VARCHAR(20) NOT NULL,

`SalesOrderNumber` TINYINT(10) NULL,

`PurchaseOrderNumber` INT NULL,

`AccountNumber` INT NULL,

`TerritoryID` INT NULL,

`CreditCardApprovalCode` VARCHAR(15) NULL,

`SubTotal` FLOAT NOT NULL DEFAULT 0.0,

`TaxAmt` FLOAT NOT NULL DEFAULT 0.0,

`Freight` FLOAT NOT NULL DEFAULT 0.0,

`TotalDue` FLOAT NOT NULL DEFAULT 0.0,

`Comment` VARCHAR(128) NULL,

`rowguid` VARCHAR(128) NOT NULL,

PRIMARY KEY (`SalesOrderID`),

INDEX `TerritoryID\_idx` (`TerritoryID` ASC) VISIBLE,

CONSTRAINT `TerritoryID`

FOREIGN KEY (`TerritoryID`)

REFERENCES `Sales`.`SalesTerritory` (`TerritoryID`)

ON DELETE RESTRICT

ON UPDATE CASCADE)

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `Sales`.`SalesTerritoryHistory`

-- -----------------------------------------------------

DROP TABLE IF EXISTS `Sales`.`SalesTerritoryHistory` ;

CREATE TABLE IF NOT EXISTS `Sales`.`SalesTerritoryHistory` (

`BusinessEntityID` INT NOT NULL,

`TerritoryID\_KEY` INT NOT NULL,

`StartDate` DATETIME NOT NULL,

`EndDate` DATETIME NULL,

`rowguid` VARCHAR(45) NOT NULL,

PRIMARY KEY (`BusinessEntityID`),

INDEX `TerritoryID\_idx` (`TerritoryID\_KEY` ASC) VISIBLE,

CONSTRAINT `TerritoryID\_KEY`

FOREIGN KEY (`TerritoryID\_KEY`)

REFERENCES `Sales`.`SalesTerritory` (`TerritoryID`)

ON DELETE RESTRICT

ON UPDATE CASCADE)

ENGINE = InnoDB;

-- -----------------------------------------------------

-- Table `Sales`.`SalesOrderDetail`

-- -----------------------------------------------------

DROP TABLE IF EXISTS `Sales`.`SalesOrderDetail` ;

CREATE TABLE IF NOT EXISTS `Sales`.`SalesOrderDetail` (

`SalesOrderID` INT NOT NULL,

`SalesOrderDetailID` INT NULL,

`CarrierTrackingNumber` VARCHAR(25) NULL,

`OrderQTY` SMALLINT(10) NULL,

`ProductID` INT NULL,

`SpecialOfferID` INT NULL,

`UnitPrice` FLOAT NULL,

`UnitPriceDiscount` FLOAT NULL,

`LineTotal` FLOAT NULL,

`rowguid` VARCHAR(45) NULL,

`ModifiedDate` VARCHAR(45) NULL,

PRIMARY KEY (`SalesOrderID`),

CONSTRAINT `SalesOrderID`

FOREIGN KEY (`SalesOrderID`)

REFERENCES `Sales`.`SalesOrderHeader` (`SalesOrderID`)

ON DELETE RESTRICT

ON UPDATE RESTRICT)

ENGINE = InnoDB;

SET SQL\_MODE=@OLD\_SQL\_MODE;

SET FOREIGN\_KEY\_CHECKS=@OLD\_FOREIGN\_KEY\_CHECKS;

SET UNIQUE\_CHECKS=@OLD\_UNIQUE\_CHECKS;

**Script Vista**

## PARTICIPACION PRODUCTO Y TERRITORIO POR MES Y AÑO

##VARIABLES: AÑO, #MES, #TERRITORIO, #PRODUCTO, #VENTAS

CREATE OR REPLACE VIEW Participaciones AS (

WITH BASE AS ( # Base con suma de las ventas agrupadas al nivel de granularidad

SELECT YEAR(y.OrderDate) AÑO,

MONTH(y.OrderDate) MES,

z.Name AREA,

x.ProductID ID\_PRODUCTO,

ROUND(SUM(x.OrderQTY \* x.LineTotal),2) INGRESOS

FROM sales.salesorderdetail x

LEFT JOIN sales.salesorderheader y

ON x.SalesOrderID = y.SalesOrderID

LEFT JOIN sales.salesterritory z

ON y.TerritoryID = z.TerritoryID

GROUP BY 1,2,3,4 )

SELECT DISTINCT # Se calcula la participacion como la suma al nivel mas bajo dividido por el totalizador

AÑO, MES, AREA, ID\_PRODUCTO,

INGRESOS/SUM(INGRESOS) OVER (PARTITION BY AÑO, MES) 'Participacion en Absoluta del Periodo',

INGRESOS/SUM(INGRESOS) OVER (PARTITION BY AÑO, MES, AREA) 'Participacion en Area',

INGRESOS/SUM(INGRESOS) OVER (PARTITION BY AÑO, MES, ID\_PRODUCTO) 'Participacion en Producto'

FROM BASE

ORDER BY 1,2,3,4 )

Ejemplo script extracción:

#!/usr/bin/env python

# coding: utf-8

def ETL():

    """Script de limpieza para el archivo CSV SalesOrderDetail.

Este script lee un archivo CSV desde el directorio 'Raw', realiza algunas transformaciones

y escribe el resultado en el directorio 'CleanFiles' para su carga en la base de datos MySQL.

"""    import polars as pl

    from pathlib import Path

    from datetime import datetime as dt

    file = 'SalesOrderDetail.csv'

    ruta = Path.cwd() / 'Raw' / file # Se crea un objeto de WindowsPath relativo al cwd

    dtypes = {'column\_8': pl.Float64} # se forza a tomar la columna 8 como float

    schema =  {

    'SalesOrderID': pl.Int32,

    'SalesOrderDetailID': pl.Int32,

    'CarrierTrackingNumber': pl.Utf8,

    'OrderQTY': pl.Int32,

    'ProductID': pl.Int32,

    'SpecialOfferID': pl.Int32,

    'UnitPrice': pl.Float64,

    'UnitPriceDiscount': pl.Float64,

    'LineTotal': pl.Float64,

    'rowguid': pl.Utf8,

    'ModifiedDate': pl.Utf8

                                }

    df = pl.read\_csv(ruta,

    schema = schema,

    separator = '\t',

    has\_header = False,

    dtypes = dtypes)

    fecha = dt.now().strftime("%Y%m%d")

    file\_out = f'{fecha}\_{file}'

    sink = Path.cwd() / 'CleanFiles' / file\_out

    df.write\_csv(sink)

if \_\_name\_\_ == "\_\_main\_\_":

    ETL()

Script de carga, data integrity y logs:

import os

import sys

from pathlib import Path

import pandas as pd

from sqlalchemy import create\_engine

import logging

import configparser

from datetime import datetime as dt

path\_actual = os.path.dirname(os.path.abspath(\_\_file\_\_))

path\_config = os.path.join(path\_actual, '..', 'Config', 'config.ini')

config = configparser.ConfigParser()

config.read(path\_config)

# Configuramos logging para mantener control de las transacciones

logging.basicConfig(filename='flujo\_data.log', level=logging.INFO)

# CONSTANTES PARA LAS VARIABLES CREADAS

DB\_USERNAME = config['MySQL']['DB\_USERNAME']

DB\_PASSWORD = config['MySQL']['DB\_PASSWORD']

DB\_HOST     = config['MySQL']['DB\_HOST']

DB\_PORT     = config['MySQL']['DB\_PORT']

DB\_NAME     = config['MySQL']['DB\_NAME']

engine = create\_engine(f"mysql+mysqlconnector://{DB\_USERNAME}:{DB\_PASSWORD}@{DB\_HOST}:{DB\_PORT}/{DB\_NAME}")

csv\_dir = str(Path.cwd() / 'CleanFiles')

def insert\_csv\_data(csv, table\_name):

    try:

        fecha = dt.now().strftime("%Y%m%d")

        file\_out = f'{fecha}\_{csv}'

        df = pd.read\_csv(os.path.join(csv\_dir, file\_out))

        validate\_data(df, table\_name)

        with engine.connect() as connection:

            with connection.begin():

                    df.to\_sql(name=table\_name, con=connection, index=False, if\_exists="replace")

                    logging.info(f"Carga existosa de archivo: {csv\_file}, tabla: {table\_name}.")

    except pd.errors.EmptyDataError:

        logging.warning(f"archivo CSV {csv\_file} vacio!")

    except pd.errors.ParserError as e:

        logging.error(f"Error en parsing del archivo: {csv\_file}: {str(e)}")

    except Exception as e:

        logging.error(f"Error en carga de tabla {table\_name}: {str(e)}")

def validate\_data(df, table\_name):

    columnas = pd.read\_sql(f"DESCRIBE {table\_name}", engine)['Field'].tolist()

    # Validamos columnas

    if not set(df.columns).issubset(set(columnas)):

        raise ValueError(f"Error en columnas tabla: {table\_name}")

def main():

    try:

        # Hacemos un loop por el directorio

        archivos\_csv = [f'{fecha}\_SalesOrderDetail.csv',

                        f'{fecha}\_SalesOrderHeader.csv',

                        f'{fecha}\_SalesTerritory.csv',

                        f'´{fecha}\_SalesTerritoryHistory.csv']

        for csv\_file in archivos\_csv:

            table\_name = os.path.splitext(csv\_file)[0]

            # llamamos la funcion de insertar

            insert\_csv\_data(csv\_file, table\_name)

        logging.info("Carga exitosa.")

    except Exception as e:

        logging.error(f"Error: {str(e)}")

if \_\_name\_\_ == "\_\_main\_\_":

    main()

Dag que orquestra el flujo (asume que airflow puede leer los directorios de los archivos)

from datetime import datetime, timedelta

from airflow import DAG

from airflow.operators.python\_operator import PythonOperator

from airflow.operators.python\_operator import DummyOperator

from pathlib import Path

import sys

sys.path.append('/../Extract)

from SalesOrderDetailETL import ETL as sodetl

from SalesOrderHeaderETL import ETL as sohetl

from SalesTerritoryETL import ETL as stetl

from SalesTerritoryHistory import ETL as sthetl

sys.path.append('/../Load')

from DataLoad import main as load

default\_args = {

    'owner': 'Diego Gonzalez',

    'depends\_on\_past': False,

    'start\_date': datetime(2023, 11, 25),

    'email\_on\_failure': False,

    'email\_on\_retry': False,

    'retries': 1,

    'retry\_delay': timedelta(minutes=5),

}

dag = DAG(

    'sales\_pipeline',

    default\_args=default\_args,

    description='ETL de sales',

    schedule\_interval=timedelta(days=1),  # se corre diariamente

)

# Tasks

start = (DummyOperator, task\_id = 'start')

end = (DummyOperator, task\_id = 'end')

clean\_csv1 = PythonOperator(

    task\_id='run\_sodetl',

    python\_callable=sodetl,

    dag=dag

)

clean\_csv2 = PythonOperator(

    task\_id='run\_sohetl',

    python\_callable=sodetl,

    dag=dag

)

clean\_csv3 = PythonOperator(

    task\_id='run\_stetl',

    python\_callable=stetl,

    dag=dag

)

clean\_csv4 = PythonOperator(

    task\_id='run\_sthetl',

    python\_callable=sthetl,

    dag=dag

)

load\_data = PythonOperator(

    task\_id='run\_load',

    python\_callable=load,

    dag=dag)

# Flujo

start >> [clean\_csv1, clean\_csv2, clean\_csv3, clean\_csv4] >> load\_data >> end