Utilizacion de Pipelines RAW y CLEAN para la esquematizacion y limpieza de datos. PSET#1

Configuracion y Preparacion de Entorno

BDD (POSTGRES) // Variables de Entorno

Una vez instalada la base de datos con sus respectivas credenciales generamos las variables de entorno con el fin de no otorgar nuestras credenciales explicitamente en el .yaml.

```
setx POSTGRESS_USER "your_username"
setx POSTGRESS_PASSWORD "your_password"
```

Creacion de Entorno Virtual

requirement.txt

```
aiofiles==22.1.0
aiohappyeyeballs==2.4.4
aiohttp==3.10.0
aiosignal==1.2.0
alembic==1.13.3
altair==4.2.0
annotated-types==0.7.0
anyio==3.5.0
argon2-cffi==21.3.0
argon2-cffi-bindings==21.2.0
arrow==1.3.0
```

```
asn1crypto==1.5.1
asttokens==2.0.5
async-timeout==4.0.3
atpublic==5.1
attrs==24.3.0
backcall==0.2.0
bcrypt==4.0.1
beautifulsoup4==4.12.3
bleach==6.2.0
bokeh==3.3.4
Bottleneck==1.4.2
Brotli==1.0.9
cachetools==5.5.1
certifi==2025.1.31
cffi==1.17.1
charset-normalizer==2.0.4
click==8.1.7
cloudpickle==3.0.0
colorama==0.4.6
comm==0.2.1
commonmark==0.9.1
contourpy==1.3.1
croniter==1.3.7
cryptography==41.0.6
cycler==0.12.1
cytoolz==1.0.1
dask==2024.5.0
dask-expr==1.1.0
datadog==0.44.0
debugpy==1.8.11
decorator==5.1.1
defusedxml==0.7.1
Deprecated==1.2.18
distributed==2024.5.0
entrypoints==0.4
exceptiongroup==1.2.0
```

```
executing==0.8.3
Faker==4.14.0
fastjsonschema==2.20.0
filelock==3.17.0
fonttools==4.56.0
fqdn==1.5.1
freezegun==1.2.2
frozenlist==1.5.0
fsspec==2024.12.0
future==1.0.0
gitdb==4.0.7
GitPython==3.1.41
great-expectations==0.15.50
greenlet==3.1.1
h11==0.14.0
HeapDict==1.0.1
httpcore==0.16.3
httpx==0.23.1
idna==3.7
importlib_metadata==8.5.0
inflection==0.5.1
ipykernel==6.15.0
ipython==8.10.0
ipython-genutils==0.2.0
ipywidgets==8.1.5
isoduration==20.11.0
itsdangerous==1.1.0
jedi==0.19.2
Jinja2==3.1.3
joblib==1.4.2
jsonpatch==1.33
jsonpointer==2.1
jsonschema==4.23.0
jsonschema-specifications==2023.7.1
jupyter_client==7.4.4
jupyter_core==5.7.2
```

```
jupyter-events==0.12.0
jupyter-server==1.23.5
jupyter-server-proxy==3.2.1
jupyter_server_terminals==0.5.3
jupyterlab-pygments==0.2.2
jupyterlab_widgets==3.0.13
kiwisolver==1.4.8
ldap3==2.9.1
Levenshtein==0.26.1
lib==4.0.0
lmdb==1.6.2
locket==1.0.0
lz4==4.3.2
mage-ai==0.9.65
makefun==1.15.6
Mako==1.2.3
MarkupSafe==3.0.2
marshmallow==3.19.0
matplotlib==3.10.0
matplotlib-inline==0.1.6
memory-profiler==0.61.0
mistune==2.0.4
mkl_fft==1.3.11
mkl_random==1.2.8
mkl-service==2.4.0
msgpack==1.0.3
multidict==6.1.0
nbclassic==1.1.0
nbclient==0.8.0
nbconvert==7.16.5
nbformat==5.10.4
nest-asyncio==1.6.0
newrelic==8.8.0
notebook==6.5.7
notebook_shim==0.2.3
numexpr==2.10.1
```

```
numpy==1.26.4
overrides==7.7.0
packaging==24.2
pandas==2.0.3
pandocfilters==1.5.0
paramiko==3.5.0
parso==0.8.4
partd==1.4.2
pickleshare==0.7.5
pillow==10.3.0
pip==25.0
platformdirs==3.10.0
polars==0.19.1
prometheus_client==0.21.0
prompt-toolkit==3.0.43
propcache==0.2.0
protobuf==4.21.12
psutil==5.9.8
psycopg2==2.9.10
pure-eval==0.2.2
pyarrow==14.0.1
pyasn1==0.6.1
pycparser==2.21
pydantic==1.10.21
pydantic_core==2.23.4
PyGithub==1.59.0
Pygments==2.15.1
PyJWT==2.6.0
PyNaCl==1.5.0
pyOpenSSL==25.0.0
pyparsing==3.2.0
PySocks==1.7.1
python-dateutil==2.8.2
python-json-logger==3.2.1
python-Levenshtein==0.26.1
pytz==2022.2.1
```

```
pywin32==308
pywinpty==2.0.14
PyYAML==6.0.2
pyzmq==26.2.0
RapidFuzz==3.12.1
redis==5.0.8
referencing==0.30.2
requests==2.31.0
rfc3339-validator==0.1.4
rfc3986==1.5.0
rfc3986-validator==0.1.1
rich==12.5.1
rpds-py==0.22.3
ruamel.yaml==0.17.17
ruamel.yaml.clib==0.2.8
scikit-learn==1.6.1
scipy==1.15.1
Send2Trash==1.8.2
sentry-sdk==1.19.1
setuptools==70.0.0
shellingham==1.5.0
simpervisor==1.0.0
simplejson==3.19.2
six==1.16.0
smmap==4.0.0
sniffio==1.3.0
snowflake==1.0.3
snowflake-connector-python==3.13.2
snowflake.core==1.0.3
snowflake._legacy==1.0.0
sortedcontainers==2.4.0
soupsieve==2.5
SQLAlchemy==1.4.51
sqlglot==26.6.0
sqlglotrs==0.3.14
sshtunnel==0.4.0
```

```
stack-data==0.2.0
tblib==1.7.0
termcolor==1.1.0
terminado==0.17.1
text-unidecode==1.3
thefuzz==0.19.0
threadpoolctl==3.5.0
tinycss2==1.4.0
tomlkit==0.13.2
toolz==1.0.0
tornado==6.3.3
tqdm==4.67.1
traitlets==5.14.3
typer==0.9.0
types-python-dateutil==2.9.0.20241206
typing_extensions==4.5.0
tzdata==2023.3
tzlocal==5.2
uri-template==1.3.0
urllib3==1.26.19
watchdog==4.0.0
wcwidth==0.2.5
webcolors==24.11.1
webencodings==0.5.1
websocket-client==1.8.0
Werkzeug==3.0.3
wheel==0.45.1
widgetsnbextension==4.0.13
win-inet-pton==1.1.0
wrapt==1.17.2
xyzservices==2022.9.0
yarl==1.18.0
zict==3.0.0
zipp==3.21.0
```

```
#Una vez instalado Python
#####################################################
#Instalar Conda
pip install conda
#Abrir Conda Prompt en Windows
#Crear ambiente de entorno
conda create --name CondaEnv python==3.10.13
#Acceder al ambiente
conda activate CondaEnv
#instalar requerimientos
conda install --yes --file requirements.txt
#acceder a la carpeta donde va a correr el proyecto
mage new proyectname
#levantar proyecto (estando adentro de la carpeta)
mage start
```

Script de Python para Postgress

```
# %%
import pandas as pd
from sqlalchemy import create_engine, Table, MetaData, text, Column, Integer, S




# %%

def create_database(engine, dbname):
    with engine.connect() as connection:
        connection.execution_options(isolation_level="AUTOCOMMIT")
        connection.execute(text(f"CREATE DATABASE {dbname}"))
        print(f"Database '{dbname}' created successfully!")
```

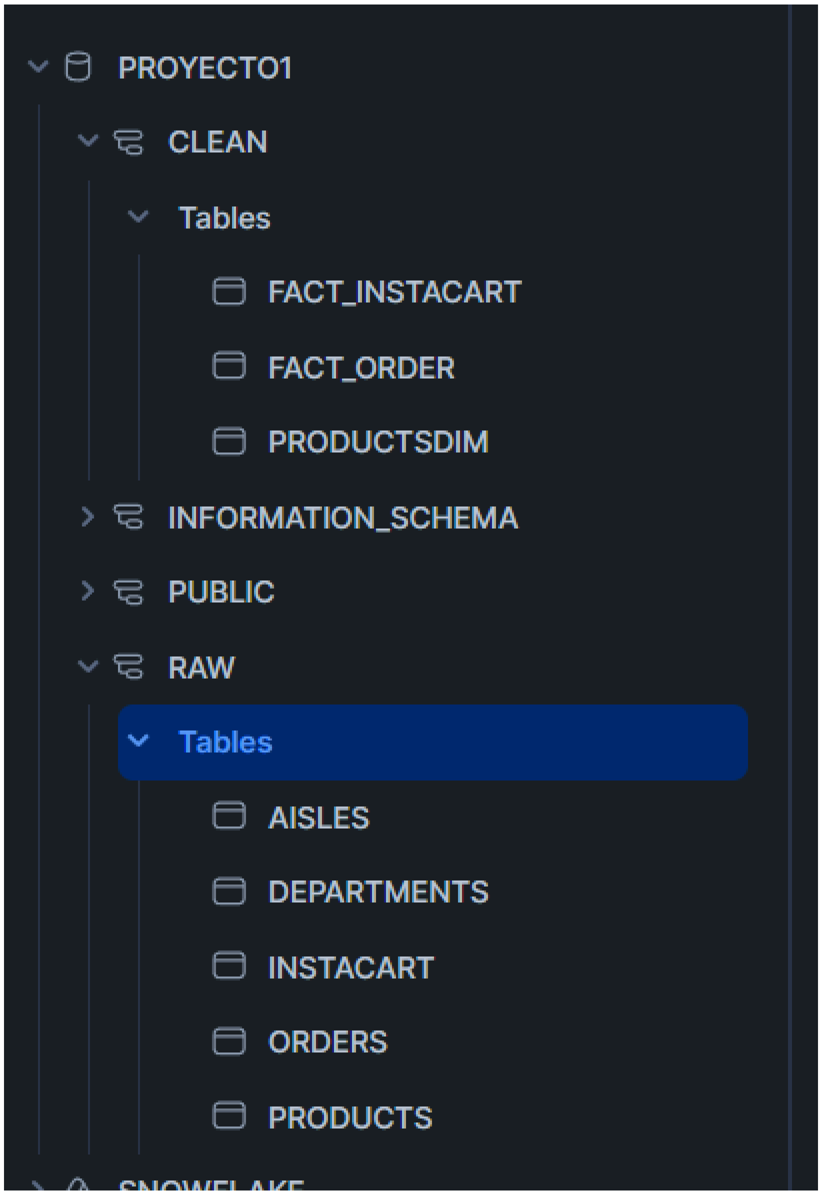
```
# %%
from sqlalchemy import create_engine, MetaData, Table, Column, Integer, String,

def create_tables(engine):
    metadata = MetaData()

    departments = Table('departments', metadata,
                    Column('department_id', Integer, primary_key=True),
                    Column('department', String)
                    )

    aisles = Table('aisles', metadata,
               Column('aisle_id', Integer, primary_key=True),
               Column('aisle', String)
               )

    instacart = Table('instacart', metadata,
                  Column('order_id', BigInteger),
                  Column('user_id', Integer),
                  Column('order_number', Integer),
                  Column('order_dow', Integer),
                  Column('order_hour_of_day', Integer),
                  Column('days_since_prior_order', Float)
                  )

    orders = Table('orders', metadata,
               Column('order_id', BigInteger),
               Column('product_id', Integer),
               Column('add_to_cart_order', Float),
               Column('reordered', Integer)
               )

    products = Table('products', metadata,
                 Column('product_id', Integer),
                 Column('product_name', String),
                Column('aisle_id', Integer),
                Column('department_id', Integer),
                Column('price', Float)
                )

    metadata.create_all(engine)
    print("Tables created successfully!")




# %%
import pandas as pd
from sqlalchemy import create_engine, MetaData, Table, text

def loadInfo(engine, batch_size=1000):
    try:
        aislesTable = pd.read_csv('C:/DataMining/Proyecto1/CSV/aisles.csv', encodi
        departmentsTable = pd.read_csv('C:/DataMining/Proyecto1/CSV/departmen
        instacartTable = pd.read_csv('C:/DataMining/Proyecto1/CSV/instacart_order
        orderproductsTable = pd.read_csv('C:/DataMining/Proyecto1/CSV/order_pro
        productsTable = pd.read_csv('C:/DataMining/Proyecto1/CSV/products.csv',


        csvMap = {
            'aisles': aislesTable,
            'departments': departmentsTable,
            'instacart': instacartTable,
            'orders': orderproductsTable,
            'products': productsTable
        }

        with engine.connect() as connection:
            metadata = MetaData()
            for key, value in csvMap.items():
                table = Table(key, metadata, autoload_with=engine, quote=True)
            for start in range(0, len(value), batch_size):
                end = start + batch_size
                batch = value.iloc[start:end]

                batch_dicts = batch.to_dict(orient='records')

                with connection.begin() as transaction:
                    try:
                        connection.execute(table.insert(), batch_dicts)
                        transaction.commit()
                    except Exception as e:
                        transaction.rollback()
                        print(f"An error occurred while inserting into table {key}: {e}")
                        raise

            print(f"Inserted data into table: {key}")

    except Exception as e:
        print(f"An error occurred: {e}")


# %%
import os

# %%

user= os.getenv('POSTGRES_USER')
password= os.getenv('POSTGRES_PASSWORD')
dbname = 'postgres'
host = 'localhost'
port = '5432'

dbCreated = 'rawproyecto1'



# %%
```

```
engine = create_engine(f'postgresql+psycopg2://{user}:{password}@{host}:{po


# %%
create_database(engine, dbCreated)

# %%
engine = create_engine(f'postgresql+psycopg2://{user}:{password}@{host}:{po

# %%
create_tables(engine)

# %%

loadInfo(engine)
```

```
#Corremos el script mediante
python sqlScript.py
```

Configuracion en SNOWFLAKE

Una vez creado nuestra cuenta de SNOWFLAKE se debe generar el esquema y la base de datos respectivas para que nuestras diferentes pipelines puedan cargar correctamente los datos.

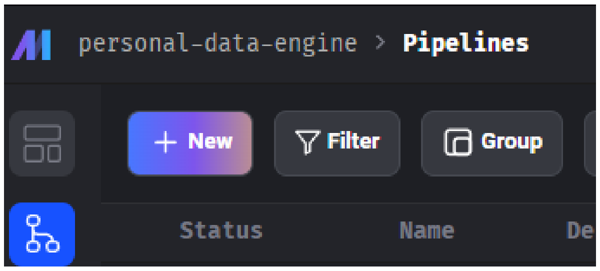


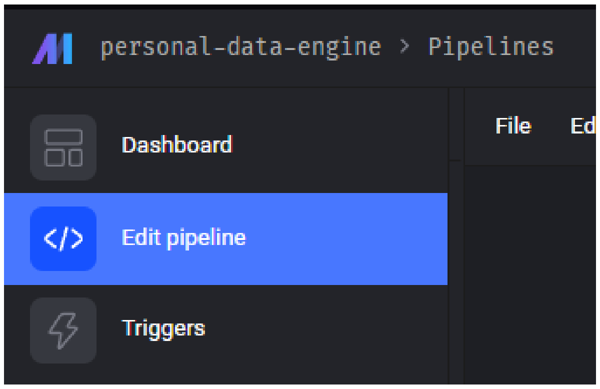
Configuracion y Ejecucion de Tuberia con MageAI (RAW PIPELINE)

```
#Accedemos al MAGE AI
mage new proyectname
#Levantamos el proyecto estado en el directorio de este
```

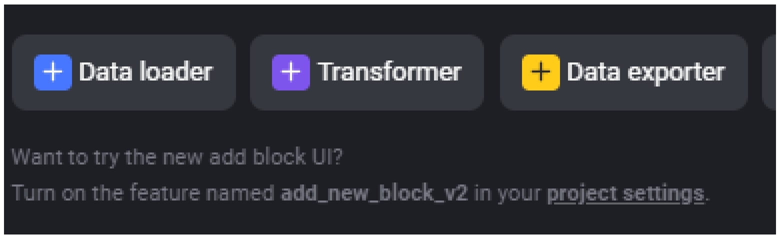
```
mage start
#La UI se abre y procedemos con la generacion de nuestros pipelines
```

Generamos un nuevo pipeline utilizando el boton de “+NEW” y posteriormente hacemos click en “Edit pipeline”





Podemos observar los diferentes metodos para generar diferentes loaders en nuestra pipeline. De esta manera generamos el que necesitamos.



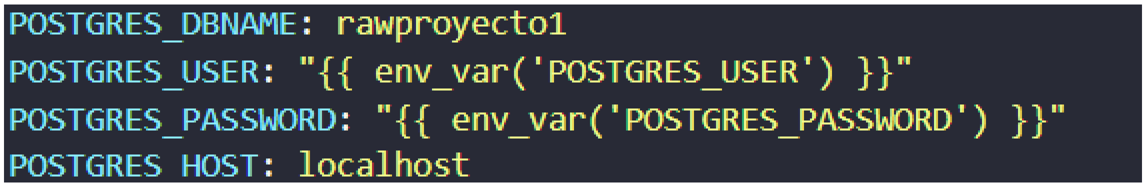
Want to try the new add block UI?
Turn on the feature named **add_new_block_v2** in your project settings.

Preparacion de Permisos

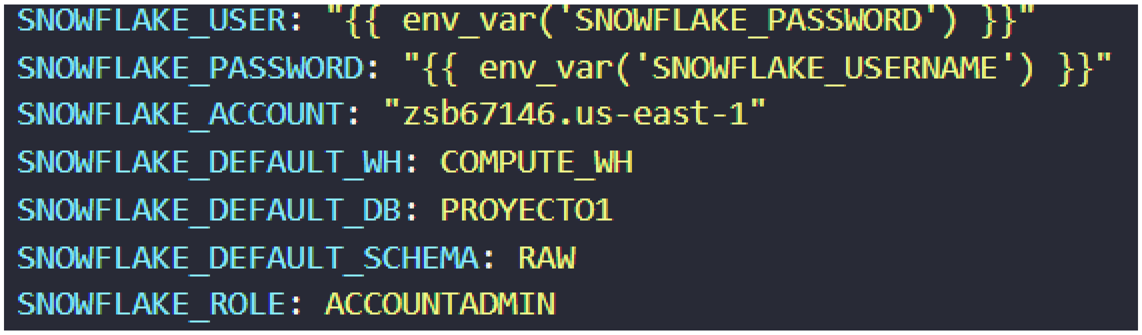
Cabe recalcar que se debe acceder al io_config.yaml en nuestra carpeta de nuestro proyecto de mage con el fin de poder configurar las respectivas configuraciones y permisos para que nuestras pipelines puedan acceder a nuestra base de datos local y al SNOWFLAKE con el fin de poder exportar y cargar toos los datos requeridos.

Se genera las variables de entorno requeridas como se menciono al comienzo del instructivo.

Ejemplo:



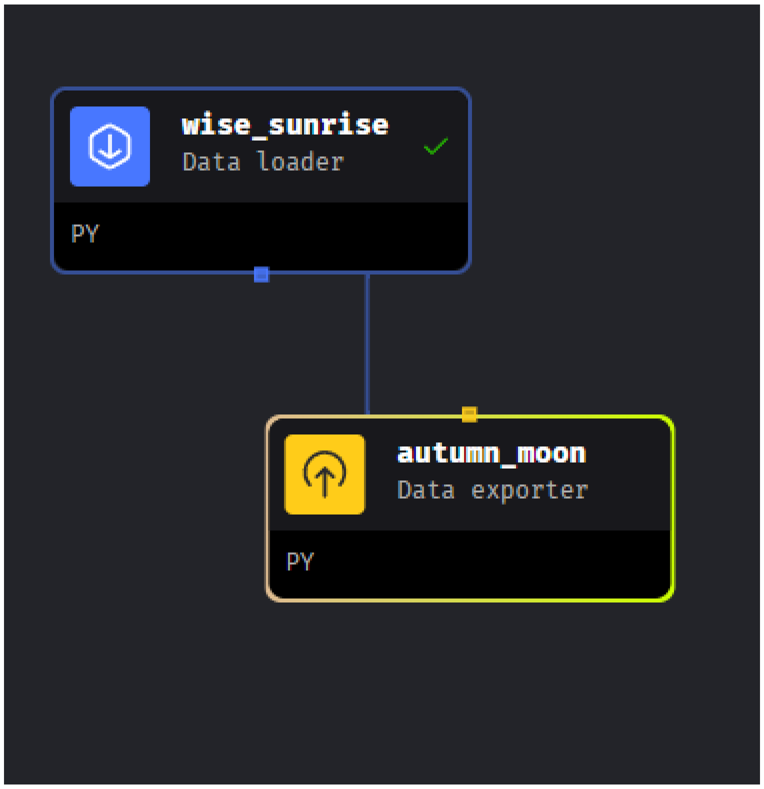
```
POSTGRES_DBNAME: rawproyecto1
POSTGRES_USER: "{{ env_var('POSTGRES_USER') }}"
POSTGRES_PASSWORD: "{{ env_var('POSTGRES_PASSWORD') }}"
POSTGRES_HOST: localhost
```



```
SNOWFLAKE_USER: "{{ env_var('SNOWFLAKE_PASSWORD') }}"
SNOWFLAKE_PASSWORD: "{{ env_var('SNOWFLAKE_USERNAME') }}"
SNOWFLAKE_ACCOUNT: "zsb67146.us-east-1"
SNOWFLAKE_DEFAULT_WH: COMPUTE_WH
SNOWFLAKE_DEFAULT_DB: PROYECTO1
SNOWFLAKE_DEFAULT_SCHEMA: RAW
SNOWFLAKE_ROLE: ACCOUNTADMIN
```

Esquematizacion

Automaticamente se genera un arbol de nuestro pipeline al lado derecho de la UI para que podamos ver como estan conectados entre ellos. Automaticamente cualquier output de cualquier loader estara disponible para el que este conectado a continuacion en base al grafico del arbol.



Data Loader

```
from mage_ai.data_preparation.repo_manager import get_repo_path
from mage_ai.io.config import ConfigFileLoader
from mage_ai.io.postgres import Postgres
from os import path
if 'data_loader' not in globals():
    from mage_ai.data_preparation.decorators import data_loader
if 'test' not in globals():
    from mage_ai.data_preparation.decorators import test


@data_loader
def load_data_from_postgres(*args, **kwargs):
    """
    Template for loading data from a PostgreSQL database.
    Specify your configuration settings in 'io_config.yaml'.
    Docs: https://docs.mage.ai/design/data-loading#postgresql
    """
    dbNames = ["aisles", "departments", "instacart", "orders", "products"]
    tables = {name: [] for name in dbNames}
    config_path = path.join(get_repo_path(), 'io_config.yaml')
    config_profile = 'default'


    with Postgres.with_config(ConfigFileLoader(config_path, config_profile)) as lo
        for name in dbNames:

            query = f"SELECT * FROM {name}"

            tables[name]=loader.load(query)
            tables[name]=tables[name].to_dict(orient="records")

        return tables


"""
@test
def test_output(output, *args) → None:
    Template code for testing the output of the block.
    assert output is not None, 'The output is undefined'
  """
```

Data_exporter

```
from mage_ai.data_preparation.repo_manager import get_repo_path
from mage_ai.io.config import ConfigFileLoader
from mage_ai.io.snowflake import Snowflake
from pandas import DataFrame
from os import path
```

```
if 'data_exporter' not in globals():
    from mage_ai.data_preparation.decorators import data_exporter




@data_exporter
def export_data_to_snowflake(tables: dict, **kwargs) → None:
    """
    Template for exporting data to a Snowflake warehouse.
    Specify your configuration settings in 'io_config.yaml'.

    Docs: https://docs.mage.ai/design/data-loading#snowflake

    """
    for tablename, table in tables.items():


        table_name = tablename
        database = 'PROYECTO1'
        schema = 'RAW'
        config_path = path.join(get_repo_path(), 'io_config.yaml')
        config_profile = 'default'

        with Snowflake.with_config(ConfigFileLoader(config_path, config_profile)) a
            loader.export(
                table,
                table_name,
                database,
                schema,
                if_exists='replace',
            )
```

EDA y Analisis de Calidad de Datos

En la carpeta donde esta este mismo archivo podemos analizar el analisis de la calida de datos en el .ipnby que se agrego con sus respectivos outputs.

```
# %%
import pandas as pd
import numpy as np

# %% [markdown]
# Conexion hacia el SNOWFLAKE

# %%
#snowflake connection
import snowflake.connector
import os

conn=snowflake.connector.connect(

    user=os.getenv('SNOWFLAKE_USERNAME'),
    password=os.getenv('SNOWFLAKE_PASSWORD'),
    account='zsb67146.us-east-1',
    warehouse='COMPUTE_WH',
    database='PROYECTO1',
    schema='RAW'

)

# %% [markdown]
# Guardado de tablas

# %%
list_tables_query = 'SHOW TABLES'
tables = pd.read_sql(list_tables_query, conn)
print(tables)
```

```
# %%

conn.cursor().execute("USE DATABASE PROYECTO1")
conn.cursor().execute("USE SCHEMA RAW")

# %%
#query to get the data from the table

names= ["AISLES","DEPARTMENTS","INSTACART","ORDERS","PRODUCTS"]

tables={}

for name in names:
   query= f"select * from {name}"
   tables[name]=pd.read_sql(query,conn)

# %%
for name in names:
    print(f"Shape of {name} table is {tables[name].shape}")

# %% [markdown]
# Valores repetidos en

# %%
tableFrequency={}

for name in names:
    print(name)
    tableFrequency[name] = tables[name].iloc[:, 0].value_counts()
    tableFrequencyofFrequency = tableFrequency[name].value_counts()

    # Filter frequencies where the count is two or greater
    filteredFrequency = tableFrequencyofFrequency[tableFrequencyofFrequency.i

    print(filteredFrequency)
    print("\n")
```

```
# %% [markdown]
# We remove all the duplicate values in the tables

# %%
# drop duplicates on all tables
for name in names:
    tables[name].drop_duplicates(inplace=True)

# %%
tableFrequency={}

for name in names:
    print(name)
    tableFrequency[name] = tables[name].iloc[:, 0].value_counts()
    tableFrequencyofFrequency = tableFrequency[name].value_counts()

    # Filter frequencies where the count is two or greater
    filteredFrequency = tableFrequencyofFrequency[tableFrequencyofFrequency.i

    print(filteredFrequency)
    print("\n")

# %% [markdown]
# Solamente habian duplicados en INSTACART la frequencia de las otras ordene

# %% [markdown]
# Now that we have dropped the duplicates we will be demonstrating the data fo

# %% [markdown]
# DATA INFO

# %% [markdown]
# AISLES TABLE

# %%
```

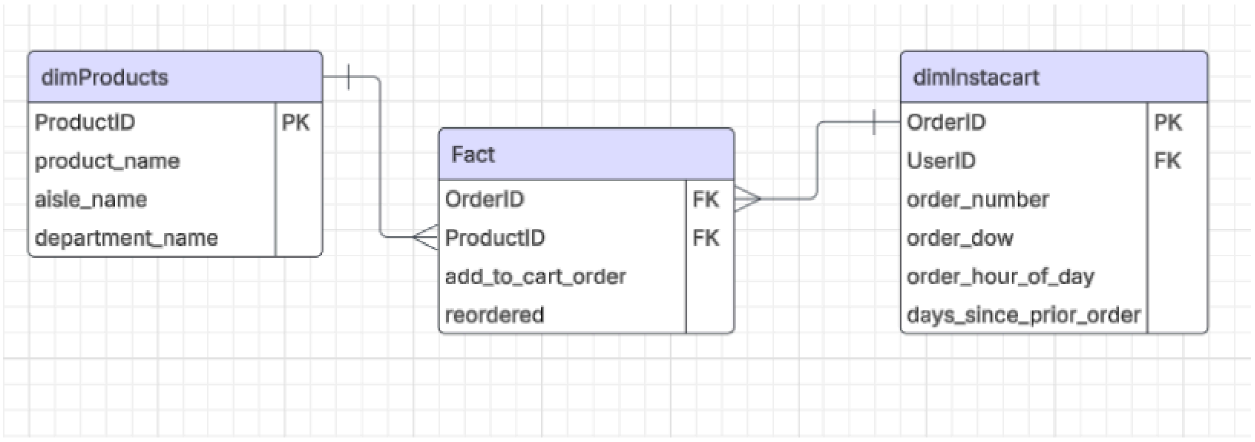
```
#describe
tables["AISLES"].describe()



# %% [markdown]
# DEPARTMENTS TABLE

# %%
tables["DEPARTMENTS"].describe()

# %% [markdown]
# PRODUCTS TABLE

# %%
tables["PRODUCTS"].describe()

# %% [markdown]
# INSTACART_ORDERS TABLE

# %%
tables["INSTACART"][["order_number", "order_dow", "order_hour_of_day", "days

# %% [markdown]
# ORDERS TABLE

# %%
tables["ORDERS"].describe().drop("count")

# %%
#only get reordered column
print("Reordered Column in Orders Table (0,1)")
tables["ORDERS"]["reordered"].describe().drop(['count', '25%', '50%', '75%', 'm

# %% [markdown]
# Eliminacion de valores nulos ("", Nan, NULL)
# %%
missing_values = tables["ORDERS"][tables["ORDERS"]['add_to_cart_order'].isna(

missing_values

# %%
##Missing (NaN) values in ORDERS table

missing_values = tables["ORDERS"][tables["ORDERS"]['add_to_cart_order'].isna(

nan_count = tables["ORDERS"]['add_to_cart_order'].isna().sum()

nan_count

# %% [markdown]
# Ya que hay 836 valores en la columan de 'add_to_cart_order', cuando en total h

# %%
tables["ORDERS"]=tables["ORDERS"][tables["ORDERS"]['add_to_cart_order'].no
missing_values = tables["ORDERS"][tables["ORDERS"]['add_to_cart_order'].isna(
nan_count = tables["ORDERS"]['add_to_cart_order'].isna().sum()
nan_count

# %% [markdown]
# Ahora se va a eliminar los valures nulos en INSTACART , existen valores nulos 
#
# Vamos a convertir en 0 los NaN values y reemplazarlos en estas filas

# %%
missing_values = tables["INSTACART"][tables["INSTACART"]['days_since_prior_

missing_values



# %%
temp=tables["INSTACART"][tables["INSTACART"]["order_number"]<2]
```

```
temp

# %%
tables["INSTACART"].loc[tables["INSTACART"]["order_number"]<2,"days_since_


# %%
temp=tables["INSTACART"][tables["INSTACART"]["order_number"]<2]
temp

# %%
tables["INSTACART"]["order_number"]=tables["INSTACART"]["order_number"].a

# %%
productSize=tables["PRODUCTS"].size

print(f"Products Table Size: {productSize}")



# %% [markdown]
# Dentro de la tabla de productos tenemos nombres de productos que no existen

# %%
tables["PRODUCTS"].head()

# %%
tables["PRODUCTS"].drop(columns=["price"], inplace=True)

# %%
tables["PRODUCTS"].head()

# %%
tables["PRODUCTS"][tables["PRODUCTS"]["product_name"].isnull()]

# %%
```

```
#Convertir a NaN verdadero

tables["PRODUCTS"]["product_name"].replace(["", "NaN"], pd.NA, inplace=True

# %%
tables["PRODUCTS"][tables["PRODUCTS"]["product_name"].isnull()]

# %%
tables["PRODUCTS"]=tables["PRODUCTS"].dropna()

# %%
tables["PRODUCTS"][tables["PRODUCTS"]["product_name"].isnull()]

# %%
tables["PRODUCTS"].head()

# %%
for name in tables:
    print(name)

# %% [markdown]
# Transferencia de datos a Star Schema

# %%
factOrder=tables["ORDERS"]
factInstaCart=tables["INSTACART"]

productsDim=tables["PRODUCTS"]

productsDim.drop(columns=["aisle_id","department_id"], inplace=True)

# %%
aislesNames=tables["AISLES"]["aisle"]
productsDim["aisle_name"]=aislesNames

departmentNames=tables["DEPARTMENTS"]["department"]
```

```
productsDim["department_name"]=departmentNames

# %%
productsDim.head()

# %%
```

Diseño e Implmentacion de la Capa CLEAN y Modelado



CLEAN PIPELINE

Generemos un CLEAN pipeline en MAGE-AI con el fin de tener nuestro data set limpio , generando las respectivas tablas generadoas en el STAR SCHEME.

Data Loader

Coneccion al SNOWFLAKER y exportacion en un mapa para el DATA TRANSFORMER

```
from mage_ai.settings.repo import get_repo_path
from mage_ai.io.config import ConfigFileLoader
from mage_ai.io.snowflake import Snowflake
from os import path
if 'data_loader' not in globals():
    from mage_ai.data_preparation.decorators import data_loader
if 'test' not in globals():
    from mage_ai.data_preparation.decorators import test


@data_loader
def load_data_from_snowflake(*args, **kwargs):
    """
    Template for loading data from a Snowflake warehouse.
    Specify your configuration settings in 'io_config.yaml'.
    """

    dbNames = ["aisles", "departments", "instacart", "orders", "products"]
    tables = {name: [] for name in dbNames}
    config_path = path.join(get_repo_path(), 'io_config.yaml')
    config_profile = 'default'

    with Snowflake.with_config(ConfigFileLoader(config_path, config_profile)) as l
        for name in dbNames:
            query = f"SELECT * FROM {name}"
            tables[name] = loader.load(query)


    return tables




@test
def test_output(output, *args) → None:
    """
    Template code for testing the output of the block.
    """
    assert output is not None, 'The output is undefined'
```

Data Transformer

Realizacion del analisis en el EDA en el DATA TRANSFORMER juntamente con la generacion del STAR SCHEMA

```
if 'transformer' not in globals():
    from mage_ai.data_preparation.decorators import transformer
if 'test' not in globals():
    from mage_ai.data_preparation.decorators import test
import pandas as pd

@transformer
def transform(tables: dict, *args, **kwargs):

    #eliminacion de duplicados en todas las tablas
    for name in tables:
        tables[name] = pd.DataFrame(tables[name])
        tables[name].drop_duplicates(inplace=True)

    print(tables["orders"])

    #eliminacion de filas que tienen el valor nulo en add_to_cart_order
    tables["orders"]=tables["orders"][tables["orders"]["add_to_cart_order"].notna

            #Convertir a 0 los valores de days_since_prior_order en 'INSTACART'

    tables["instacart"].loc[tables["instacart"]["order_number"]<2,"days_since_pric

    #Convertir en int order number
    tables["instacart"]["order_number"]=tables["instacart"]["order_number"].asty

    #votar price de la tabla products ya que no hay valores
    tables["products"].drop(columns=["price"], inplace=True)

    #Convertir a NaN verdadero para posterirmente reliminar todos los valores de
    tables["products"]["product_name"].replace(["", "NaN"], pd.NA, inplace=True)
    tables["products"]=tables["products"].dropna()

    #Realizar el STAR ESQUEMA

    factOrder=tables["orders"]
    factInstaCart=tables["instacart"]

    productsDim=tables["products"]

    productsDim.drop(columns=["aisle_id","department_id"], inplace=True)

    aislesNames=tables["aisles"]["aisle"]
    productsDim["aisle_name"]=aislesNames

    departmentNames=tables["departments"]["department"]
    productsDim["department_name"]=departmentNames

    return {

        "fact_order":factOrder,
        "fact_instacart":factInstaCart,
        "productsDim":productsDim

    }


@test
def test_output(output, *args) → None:
    """
    Template code for testing the output of the block.
    """
    assert output is not None, 'The output is undefined'
```

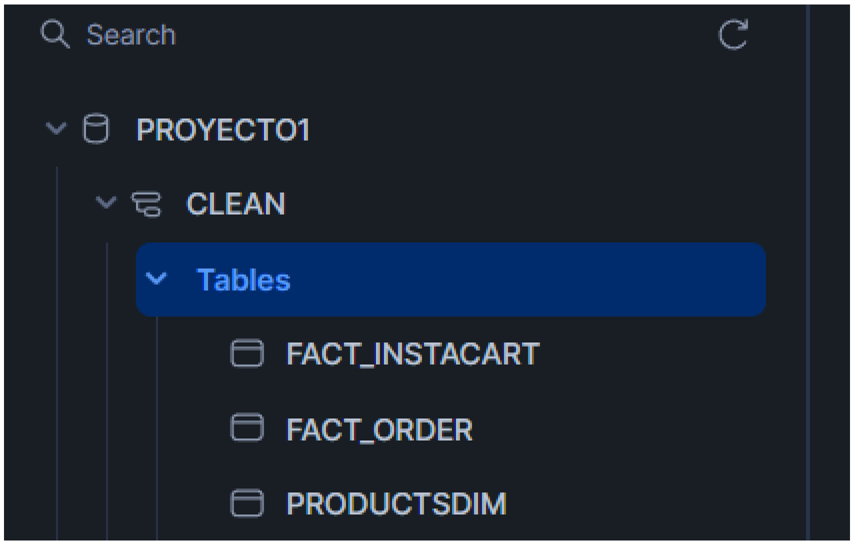
Data Exporter

Exportar el resultado de la transformacion con sus debidas columnas y llaves hacia la base de datos en el SNOWFLAKE

```
from mage_ai.data_preparation.repo_manager import get_repo_path
from mage_ai.io.config import ConfigFileLoader
from mage_ai.io.snowflake import Snowflake
from pandas import DataFrame
from os import path



if 'data_exporter' not in globals():
    from mage_ai.data_preparation.decorators import data_exporter





@data_exporter
def export_data_to_snowflake(tables: dict, **kwargs) → None:
    """
    Template for exporting data to a Snowflake warehouse.
    Specify your configuration settings in 'io_config.yaml'.

    Docs: https://docs.mage.ai/design/data-loading#snowflake

    """
    for tablename, table in tables.items():


        table_name = tablename
        database = 'PROYECTO1'
        schema = 'CLEAN'
        config_path = path.join(get_repo_path(), 'io_config.yaml')
        config_profile = 'default'

        with Snowflake.with_config(ConfigFileLoader(config_path, config_profile)) a
            loader.export(
                table,
                table_name,
                database,
                schema,
                if_exists='replace',
            )
```

SNOWFLAKE CLEAN

Una vez que nuestra pipeline se haya ejecutado correctamente nosotros podremos ver las tablas agregadas a nuestro esquema de CLEAN con todos los cambios analiticos que realizamos para poder limpiar todos nuestros datos.


Search
PROYECTO1
CLEAN
Tables
FACT_INSTACART
FACT_ORDER
PRODUCTSDIM