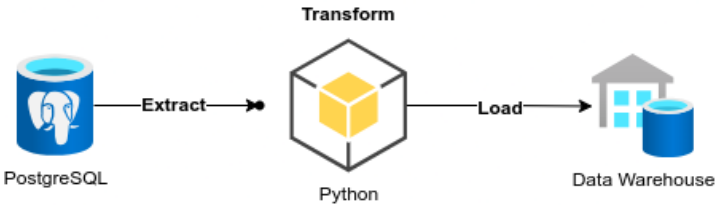
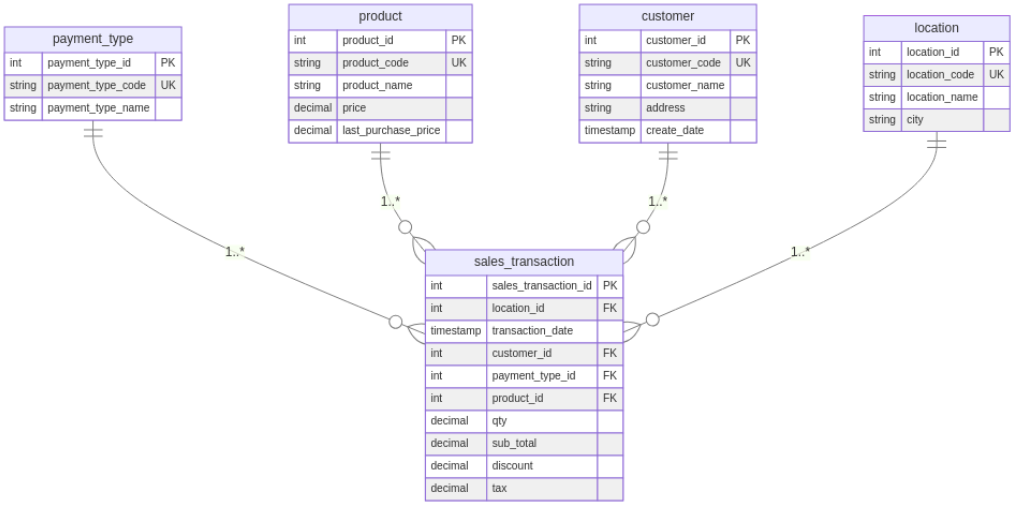
1. This ETL flow extracts data from a PostgreSQL database, transforms it using Python, and loads it into a PostgreSQL-based data warehouse for storage and analysis.



2. Create an ERD based on the data source to better understand the relationships between tables. (ERD design using https://mermaid.live/)



3. Prepare the DDL based on the data source, including the tables, columns, data types, and keys, and then generate the dummy data (DML).
   ❖ DDL

```sql
create table payment_type (
    payment_type_id int not null primary key,
    payment_type_code varchar(10),
    payment_type_name varchar(30)
);

create table customer (
    customer_id int not null primary key,
    customer_code varchar(10) unique,
    customer_name varchar(100),
    address text,
    create_date timestamp
);

create table product (
    product_id int not null primary key,
    product_code varchar(10) unique,
    product_name varchar(100),
    price numeric(20,4),
    last_purchase_price numeric(20,4)
);

create table location (
    location_id int not null primary key,
    location_code varchar(10) unique,
    location_name varchar(100),
    city varchar(50)
);
```

```sql
create table sales_transaction (
    sales_transaction_id int not null primary key,
    location_id int,
    transaction_date timestamp,
    customer_id int,
    payment_type_id int,
    product_id int,
    qty numeric(10,2),
    sub_total numeric(20,4),
    discount numeric(5,2),
    tax numeric(5,2),
    foreign key (payment_type_id) references payment_type (payment_type_id),
    foreign key (customer_id) references customer (customer_id),
    foreign key (product_id) references product (product_id),
    foreign key (location_id) references location (location_id)
);
```

❖ DML

```sql
insert into payment_type
select id,
    concat('PT-0',id::text),
    concat('Type ',id::text)
from generate_series(1, 100) as id;

insert into customer
select id,
    concat('Code-0',id::text),
    concat('Customer ',id::text),
    concat('Address ',id::text),
    date_trunc('minutes', now() - (random() * interval '180 days'))
from generate_series(1, 100) as id;

insert into product
select id,
    concat('PC-0',id::text),
    concat('Product ',id::text),
    random() * (1000 - 10) + 10::numeric,
    random() * (1000 - 10) + 10::numeric
from generate_series(1, 100) as id;

insert into location
select id,
    concat('L-0',id::text),
    concat('Location ',id::text),
    concat('City ',id::text)
from generate_series(1, 100) as id;

insert into sales_transaction
select id,
    id,
    date_trunc('minutes', now() - (random() * interval '180 days')),
    id,
    id,
    id,
    (random() * 100 + 1)::int,
    random() * (1000 - 10) + 10::numeric,
    random() * 50, random() * 10
from generate_series(1, 100) as id;
```

4. Create a data model using the STAR schema to be used as the structure in the destination (data warehouse). Fact table: `fact_sales_transaction`; Dimension tables: `dim_payment_type`, `dim_customer`, `dim_product`, and `dim_location`.

```sql
create table dim_payment_type (
    payment_type_id int not null primary key,
    payment_type_code varchar(10),
    payment_type_name varchar(30)
);

create table dim_customer (
    customer_id int not null primary key,
    customer_code varchar(10) unique,
    customer_name varchar(100),
    address text,
    create_date timestamp
);

create table dim_product (
    product_id int not null primary key,
    product_code varchar(10) unique,
    product_name varchar(100),
    price numeric(20,4),
    last_purchase_price numeric(20,4)
);

create table dim_location (
    location_id int not null primary key,
    location_code varchar(10) unique,
    location_name varchar(100),
    city varchar(50)
);
```

```sql
create table fact_sales_transaction (
    sales_transaction_id int not null primary key,
    location_id int,
    month int,
    year int,
    customer_id int,
    payment_type_id int,
    product_id int,
    qty numeric(10,2),
    sub_total numeric(20,4),
    discount numeric(5,2),
    tax numeric(5,2),
    foreign key (payment_type_id) references dim_payment_type (payment_type_id),
    foreign key (customer_id) references dim_customer (customer_id),
    foreign key (product_id) references dim_product (product_id),
    foreign key (location_id) references dim_location (location_id)
);
```

5. Create a configuration file (`config.py`) to store the parameters for the source and destination connections, as well as the table mappings.
   - ❖ Setup connection for Source and Destination

```python
#!python config

#--- Setup connection to Source
source_conn = {
    'user': 'gugus',
    'password': 'gugus12345',
    'host': 'localhost',
    'port': '5432',
    'database': 'pos_system'
}

#--- Setup connection to Destination
dest_conn = {
    'user': 'gugus',
    'password': 'gugus12345',
    'host': 'localhost',
    'port': '5432',
    'database': 'data_warehouse'
}
```

   - ❖ Create mapping from Source to Destination

```python
#--- Mapping from Source to Destination
etl_config = {
    "payment": {
        "source_table": "payment_type",
        "destination_table": "dim_payment_type",
        "column_mapping": {
            "payment_type_id": "payment_type_id",
            "payment_type_code": "payment_type_code",
            "payment_type_name": "payment_type_name"
        },
        "query": "SELECT * FROM payment_type"
    },
    "customer": {
        "source_table": "customer",
        "destination_table": "dim_customer",
        "column_mapping": {
            "customer_id": "customer_id",
            "customer_code": "customer_code",
            "customer_name": "customer_name",
            "address": "address",
            "create_date": "create_date"
        },
        "query": "SELECT * FROM customer"
    },
    "product": {
        "source_table": "product",
        "destination_table": "dim_product",
        "column_mapping": {
            "product_id": "product_id",
            "product_code": "product_code",
            "product_name": "product_name",
            "price": "price",
            "last_purchase_price": "last_purchase_price"
        },
        "query": "SELECT * FROM product"
    },
    "location": {
        "source_table": "location",
        "destination_table": "dim_location",
        "column_mapping": {
            "location_id": "location_id",
            "location_code": "location_code",
            "location_name": "location_name",
            "city": "city"
        },
        "query": "SELECT * FROM location"
    },
```

```
    "sales": {
        "source_table": ["sales_transaction", "payment_type", "customer", "product", "location"],
        "destination_table": "fact_sales_transaction",
        "column_mapping": {
            "sales_transaction_id": "sales_transaction_id",
            "location_id": "location_id",
            "month": "month",
            "year": "year",
            "customer_id": "customer_id",
            "payment_type_id": "payment_type_id",
            "product_id": "product_id",
            "qty": "qty",
            "sub_total": "sub_total",
            "discount": "discount",
            "tax": "tax"
        },
        "query": """
            SELECT
                s.sales_transaction_id,
                l.location_id,
                extract(month from s.transaction_date) as month,
                extract(year from s.transaction_date) as year,
                c.customer_id,
                pt.payment_type_id,
                p.product_id,
                sum(s.qty) qty,
                sum(s.sub_total) sub_total,
                s.discount,
                s.tax
            FROM sales_transaction s
            INNER JOIN location l ON s.location_id = l.location_id
            INNER JOIN payment_type pt ON s.payment_type_id = pt.payment_type_id
            INNER JOIN customer c ON s.customer_id = c.customer_id
            INNER JOIN product p ON s.product_id = p.product_id
            GROUP BY 1,2,3,4,5,6,7,10,11
        """
    }
}
```

6. Create an ETL script (`ETL.ipynb`) to execute the ETL process, starting from installing the required libraries to the final load process into the data warehouse. Use logging to track the entire process.

Library

```
#!pip install pandas
#!pip install sqlalchemy
#!pip install psycopg2-binary
```

```
import logging
import pandas as pd

from urllib.parse import quote_plus
from sqlalchemy import create_engine

from config import source_conn
from config import dest_conn
from config import etl_config

import warnings
warnings.filterwarnings('ignore')
```

Logging setup

```
logging.basicConfig(level=logging.INFO)
```

Setup connection from Source to Destination

```
def db_connection(conn_params):
    conn_str = f"postgresql://{conn_params['user']}:{quote_plus(conn_params['password'])}@{conn_params['host']}:{conn_params['port']}/{conn_
    engine = create_engine(conn_str)
    return engine.connect()
```

Validate the ETL configuration (config.py -> can be customize)

```
def validate_config(etl_config):
    mapping_keys = ['source_table', 'query', 'destination_table', 'column_mapping']
    for table_name, table_config in etl_config.items():
        for key in mapping_keys:
            if key not in table_config:
                raise ValueError(f"Missing {key} in config for table {table_name}")
    logging.info("Config validation passed")
```

Extract data from the source

```
def extract(table_config):
    try:
        logging.info(f"Extracting data from {table_config['source_table']}")
        with db_connection(source_conn) as conn:
            df = pd.read_sql(table_config["query"], conn)
        return df
    except Exception as e:
        logging.error(f"Error extracting data from {table_config['source_table']}: {e}")
        raise
```

Transform the extracted data

```
def transform(df, table_config):
    try:
        logging.info(f"Transforming data for {table_config['destination_table']}")
        df.rename(columns=table_config["column_mapping"], inplace=True)
        return df
    except Exception as e:
        logging.error(f"Error transforming data for {table_config['destination_table']}: {e}")
        raise
```

Load the transformed data into the destination table, replacing the data without dropping the table

```
from sqlalchemy import text

def load(df, table_config):
    try:
        logging.info(f"Replacing data in {table_config['destination_table']}")

        #--- Connect to the warehouse database (destination)
        with db_connection(dest_conn) as conn:
            # Step 1: Remove all rows + reset identity using text()
            conn.execute(text(f"TRUNCATE TABLE {table_config['destination_table']} RESTART IDENTITY CASCADE;"))
            conn.commit()

            #--- Step 2: Insert the new records into the table
            df.to_sql(
                table_config["destination_table"],
                conn,
                #--- Insert new data
                if_exists="append",
                index=False
            )

        logging.info(f"Data successfully loaded into {table_config['destination_table']}")

    except Exception as e:
        logging.error(f"Error replacing data in {table_config['destination_table']}: {e}")
        raise
```

Run full ETL process

```python
def run_etl():
    try:
        logging.info("Starting ETL Process")
        validate_config(etl_config)
        for table_name, table_config in etl_config.items():
            df = extract(table_config)
            df = transform(df, table_config)
            load(df, table_config)
        logging.info("ETL Process Completed Successfully!")
    except Exception as e:
        logging.error(f"ETL process failed: {e}")
```

Run Process

```python
if __name__ == "__main__":
    run_etl()
```

```
INFO:root:Starting ETL Process
INFO:root:Config validation passed
INFO:root:Extracting data from payment_type
INFO:root:Transforming data for dim_payment_type
INFO:root:Replacing data in dim_payment_type
INFO:root:Data successfully loaded into dim_payment_type
INFO:root:Extracting data from customer
INFO:root:Transforming data for dim_customer
INFO:root:Replacing data in dim_customer
INFO:root:Data successfully loaded into dim_customer
INFO:root:Extracting data from product
INFO:root:Transforming data for dim_product
INFO:root:Replacing data in dim_product
INFO:root:Data successfully loaded into dim_product
INFO:root:Extracting data from location
INFO:root:Transforming data for dim_location
INFO:root:Replacing data in dim_location
INFO:root:Data successfully loaded into dim_location
INFO:root:Extracting data from ['sales_transaction', 'payment_type', 'customer', 'product', 'location']
INFO:root:Transforming data for fact_sales_transaction
INFO:root:Replacing data in fact_sales_transaction
INFO:root:Data successfully loaded into fact_sales_transaction
INFO:root:ETL Process Completed Successfully!
```

Full source code:
https://github.com/gugusx/mini-project-ETL.git