Customer 360 Data Warehouse — Mini Project (Postgres + Python ETL)

A complete, runnable project that demonstrates **DWH modeling** (star schema) and **ETL/integration** of multiple sources (CSV, JSON, mock API), with **analytics SQL** and optional **Airflow DAG**.

🗂️ Repository Structure

```
customer360-dwh/
├─ README.md
├─ docker-compose.yml
├─ .env
├─ requirements.txt
├─ sql/
│  ├─ 00_create_schema.sql
│  ├─ 01_dim_tables.sql
│  ├─ 02_fact_tables.sql
│  └─ 99_sample_analytics.sql
├─ data/
│  ├─ generate_data.py
│  ├─ customers.csv
│  ├─ products.json
│  ├─ sales.csv
│  └─ web_activity.csv
├─ etl/
│  ├─ etl_config.yaml
│  └─ etl_load.py
└─ airflow/ (optional)
   └─ dag_customer360.py
```

🗜️ Quick Start

1) **Install prerequisites** - Docker & Docker Compose - Python 3.10+

2) **Clone & set up**

```
# assuming this markdown is your repo root content
# create files below, then:
pip install -r requirements.txt
```

3) **Start Postgres + Adminer**

```
docker compose up -d
```

- Postgres: `localhost:5432` - Adminer UI: http://localhost:8080 (System: PostgreSQL, Server: db, User: postgres, Pass: postgres, DB: dwh)

4) **Generate sample data**

```
python data/generate_data.py
```

5) **Run ETL**

```
python etl/etl_load.py
```

6) **Try analytics queries** (see `sql/99_sample_analytics.sql`).

⚙️Configuration

.env

```
POSTGRES_HOST=localhost
POSTGRES_PORT=5432
POSTGRES_DB=dwh
POSTGRES_USER=postgres
POSTGRES_PASSWORD=postgres

# ETL options
BATCH_SIZE=5000
```

requirements.txt

```
pandas==2.2.2
SQLAlchemy==2.0.32
psycopg2-binary==2.9.9
pyyaml==6.0.2
python-dateutil==2.9.0.post0
```

docker-compose.yml

```yaml
version: '3.9'
services:
  db:
    image: postgres:16
    container_name: c360_db
    environment:
      POSTGRES_PASSWORD: ${POSTGRES_PASSWORD}
      POSTGRES_USER: ${POSTGRES_USER}
      POSTGRES_DB: ${POSTGRES_DB}
    ports:
      - "${POSTGRES_PORT}:5432"
    volumes:
      - db_data:/var/lib/postgresql/data
  adminer:
    image: adminer
    container_name: c360_adminer
    restart: always
    ports:
      - "8080:8080"
    depends_on:
      - db
volumes:
  db_data:
```

🧱 Data Warehouse Schema (Star)

sql/00_create_schema.sql

```sql
CREATE SCHEMA IF NOT EXISTS staging;
CREATE SCHEMA IF NOT EXISTS analytics;
```

sql/01_dim_tables.sql

```sql
-- Date dimension
CREATE TABLE IF NOT EXISTS analytics.dim_date (
  date_id       DATE PRIMARY KEY,
  year          INT,
  quarter       INT,
  month         INT,
  day           INT,
  day_of_week   INT,
  week_of_year  INT
);

-- Customer dimension (SCD1 for simplicity)
CREATE TABLE IF NOT EXISTS analytics.dim_customer (
  customer_sk   SERIAL PRIMARY KEY,
  customer_id   VARCHAR(50) UNIQUE,
  first_name    VARCHAR(100),
  last_name     VARCHAR(100),
  gender        VARCHAR(20),
  email         VARCHAR(200),
  city          VARCHAR(100),
  state         VARCHAR(100),
  country       VARCHAR(100),
  loyalty_score INT,
  updated_at    TIMESTAMP DEFAULT NOW()
);

-- Product dimension
CREATE TABLE IF NOT EXISTS analytics.dim_product (
  product_sk   SERIAL PRIMARY KEY,
  product_id   VARCHAR(50) UNIQUE,
  name         VARCHAR(200),
  category     VARCHAR(100),
  brand        VARCHAR(100),
  price        NUMERIC(10,2),
  updated_at   TIMESTAMP DEFAULT NOW()
);

-- Channel dimension (web, mobile, store, etc.)
CREATE TABLE IF NOT EXISTS analytics.dim_channel (
  channel_sk  SERIAL PRIMARY KEY,
  channel_id  VARCHAR(50) UNIQUE,
  name        VARCHAR(100)
);
```

sql/02_fact_tables.sql

```sql
-- Sales fact
CREATE TABLE IF NOT EXISTS analytics.fact_sales (
  sales_sk     BIGSERIAL PRIMARY KEY,
  date_id      DATE REFERENCES analytics.dim_date(date_id),
  customer_sk  INT  REFERENCES analytics.dim_customer(customer_sk),
  product_sk   INT  REFERENCES analytics.dim_product(product_sk),
  channel_sk   INT  REFERENCES analytics.dim_channel(channel_sk),
  quantity     INT,
  revenue      NUMERIC(12,2)
);

-- Web activity fact (pageviews / sessions)
CREATE TABLE IF NOT EXISTS analytics.fact_web_activity (
  activity_sk  BIGSERIAL PRIMARY KEY,
  date_id      DATE REFERENCES analytics.dim_date(date_id),
  customer_sk  INT  REFERENCES analytics.dim_customer(customer_sk),
  channel_sk   INT  REFERENCES analytics.dim_channel(channel_sk),
  event_type   VARCHAR(50),      -- page_view, add_to_cart, purchase
  session_id   VARCHAR(100),
  page_url     TEXT
);
```

sql/99_sample_analytics.sql

```sql
-- Top categories by revenue in last 90 days
SELECT p.category, SUM(fs.revenue) AS revenue
FROM analytics.fact_sales fs
JOIN analytics.dim_product p ON p.product_sk = fs.product_sk
JOIN analytics.dim_date d ON d.date_id = fs.date_id
WHERE d.date_id >= CURRENT_DATE - INTERVAL '90 days'
GROUP BY p.category
ORDER BY revenue DESC
LIMIT 10;

-- Customer Lifetime Value (simplified)
SELECT c.customer_id,
       SUM(fs.revenue) AS clv,
       COUNT(DISTINCT fs.date_id) AS active_days
FROM analytics.fact_sales fs
JOIN analytics.dim_customer c ON c.customer_sk = fs.customer_sk
GROUP BY c.customer_id
ORDER BY clv DESC
LIMIT 20;

-- Web funnel: page view -> add_to_cart -> purchase counts (last 30 days)
```

```sql
WITH e AS (
  SELECT d.date_id, wa.event_type, COUNT(*) AS cnt
  FROM analytics.fact_web_activity wa
  JOIN analytics.dim_date d ON d.date_id = wa.date_id
  WHERE d.date_id >= CURRENT_DATE - INTERVAL '30 days'
  GROUP BY d.date_id, wa.event_type
)
SELECT date_id,
       COALESCE(MAX(CASE WHEN event_type='page_view' THEN cnt END),0) AS
page_views,
       COALESCE(MAX(CASE WHEN event_type='add_to_cart' THEN cnt END),0) AS adds,
       COALESCE(MAX(CASE WHEN event_type='purchase' THEN cnt END),0) AS
purchases
FROM e
GROUP BY date_id
ORDER BY date_id DESC;
```

🧪 Sample Data Generation

data/generate_data.py

```python
import os, json, random
import pandas as pd
from datetime import datetime, timedelta

random.seed(42)
os.makedirs('data', exist_ok=True)

# Customers
cities = [
    ("San Diego","CA","USA"),("Austin","TX","USA"),("Seattle","WA","USA"),
    ("New York","NY","USA"),("Boston","MA","USA"),("Chicago","IL","USA")
]
first =
["Alex","Jordan","Taylor","Sam","Riley","Casey","Jamie","Avery","Drew","Morgan"]
last  =
["Lee","Patel","Kim","Garcia","Brown","Davis","Martinez","Wilson","Clark","Nguyen"]

customers = []
for i in range(1, 2001):
    f,l = random.choice(first), random.choice(last)
    city,state,country = random.choice(cities)
    customers.append({
        'customer_id': f"C{i:05d}",
```

```
        'first_name': f,
        'last_name': l,
        'gender': random.choice(['Male','Female','Other']),
        'email': f"{f.lower()}.{l.lower()}{i}@example.com",
        'city': city, 'state': state, 'country': country,
        'loyalty_score': random.randint(0, 100)
    })

pd.DataFrame(customers).to_csv('data/customers.csv', index=False)

# Products
categories = ["Electronics","Apparel","Home","Beauty","Sports"]
brands = ["Acme","Globex","Soylent","Initech","Umbrella"]
products = []
for i in range(1, 501):
    cat = random.choice(categories)
    price = round(random.uniform(5, 500), 2)
    products.append({
        'product_id': f"P{i:04d}",
        'name': f"Product {i}",
        'category': cat,
        'brand': random.choice(brands),
        'price': price
    })

with open('data/products.json','w') as f:
    json.dump(products, f, indent=2)

# Sales (last 180 days)
start_date = datetime.today().date() - timedelta(days=180)
rows = []
for day in range(181):
    date = start_date + timedelta(days=day)
    for _ in range(random.randint(50, 150)):
        cust = random.choice(customers)['customer_id']
        prod = random.choice(products)
        qty = random.randint(1, 3)
        revenue = round(prod['price'] * qty, 2)
        channel = random.choice(['web','mobile','store'])
        rows.append({
            'date': date.isoformat(),
            'customer_id': cust,
            'product_id': prod['product_id'],
            'channel_id': channel,
            'quantity': qty,
            'revenue': revenue
        })
```

```
pd.DataFrame(rows).to_csv('data/sales.csv', index=False)

# Web activity
events = ['page_view','add_to_cart','purchase']
wa = []
for day in range(181):
    date = start_date + timedelta(days=day)
    for _ in range(random.randint(300, 800)):
        cust = random.choice(customers)['customer_id']
        evt = random.choices(events, weights=[0.8, 0.15, 0.05])[0]
        wa.append({
            'date': date.isoformat(),
            'customer_id': cust,
            'channel_id': random.choice(['web','mobile']),
            'event_type': evt,
            'session_id': f"S{random.randint(1, 10_000_000)}",
            'page_url': random.choice(['/','/plp','/pdp','/cart','/search'])
        })

pd.DataFrame(wa).to_csv('data/web_activity.csv', index=False)
print("Sample data generated in ./data")
```

🛠️ETL Pipeline (Python + SQLAlchemy)

etl/etl_config.yaml

```
db:
  host: ${POSTGRES_HOST}
  port: ${POSTGRES_PORT}
  database: ${POSTGRES_DB}
  user: ${POSTGRES_USER}
  password: ${POSTGRES_PASSWORD}

files:
  customers: data/customers.csv
  products: data/products.json
  sales: data/sales.csv
  web_activity: data/web_activity.csv

options:
  batch_size: ${BATCH_SIZE}
```

etl/etl_load.py

```
import os
import yaml
import json
import math
import pandas as pd
from sqlalchemy import create_engine, text
from dateutil import parser

# ---------- Helpers ----------
def env_default(val, fallback):
    return os.getenv(val, fallback)

def get_engine(cfg):
    host = cfg['db']['host']
    port = cfg['db']['port']
    db   = cfg['db']['database']
    user = cfg['db']['user']
    pwd  = cfg['db']['password']
    url = f"postgresql+psycopg2://{user}:{pwd}@{host}:{port}/{db}"
    return create_engine(url, future=True)

# ---------- Load config ----------
with open('etl/etl_config.yaml','r') as f:
    raw = f.read()
    # simple env var interpolation
    for k,v in os.environ.items():
        raw = raw.replace(f"${{{k}}}", v)
    cfg = yaml.safe_load(raw)

engine = get_engine(cfg)
batch_size = int(cfg['options'].get('batch_size', 5000))

# ---------- Bootstrap schema ----------
with engine.begin() as con:
    for path in [
        'sql/00_create_schema.sql',
        'sql/01_dim_tables.sql',
        'sql/02_fact_tables.sql']:
        with open(path,'r') as f:
            con.execute(text(f.read()))

# ---------- Upsert helpers ----------
UPSERT_CUSTOMER = text('''
INSERT INTO analytics.dim_customer (customer_id, first_name, last_name, gender,
email, city, state, country, loyalty_score)
VALUES
(:customer_id, :first_name, :last_name, :gender, :email, :city, :state, :country, :loyalty_score)
```

```
ON CONFLICT (customer_id) DO UPDATE SET
  first_name=EXCLUDED.first_name,
  last_name=EXCLUDED.last_name,
  gender=EXCLUDED.gender,
  email=EXCLUDED.email,
  city=EXCLUDED.city,
  state=EXCLUDED.state,
  country=EXCLUDED.country,
  loyalty_score=EXCLUDED.loyalty_score,
  updated_at=NOW()
RETURNING customer_sk;
''')

UPSERT_PRODUCT = text('''
INSERT INTO analytics.dim_product (product_id, name, category, brand, price)
VALUES (:product_id, :name, :category, :brand, :price)
ON CONFLICT (product_id) DO UPDATE SET
  name=EXCLUDED.name,
  category=EXCLUDED.category,
  brand=EXCLUDED.brand,
  price=EXCLUDED.price,
  updated_at=NOW()
RETURNING product_sk;
''')

UPSERT_CHANNEL = text('''
INSERT INTO analytics.dim_channel (channel_id, name)
VALUES (:channel_id, :name)
ON CONFLICT (channel_id) DO UPDATE SET name=EXCLUDED.name
RETURNING channel_sk;
''')

# ---------- Populate static dims ----------
with engine.begin() as con:
    for ch in [('web','Web'),('mobile','Mobile'),('store','Store')]:
        con.execute(UPSERT_CHANNEL, { 'channel_id': ch[0], 'name': ch[1] })

# ---------- Date dimension loader ----------
from datetime import date, timedelta

def load_dim_date(con, start: date, end: date):
    # generate a pandas dataframe for the range
    days = []
    cur = start
    while cur <= end:
        days.append({
            'date_id': cur,
            'year': cur.year,
```

```
            'quarter': (cur.month-1)//3 + 1,
            'month': cur.month,
            'day': cur.day,
            'day_of_week': cur.isoweekday(),
            'week_of_year': int(cur.strftime('%V'))
        })
        cur += timedelta(days=1)
    df = pd.DataFrame(days)
    # upsert (idempotent)
    temp = 'staging_dim_date'
    df.to_sql(temp, con, schema='staging', if_exists='replace', index=False)
    con.execute(text('''
        INSERT INTO analytics.dim_date (date_id, year, quarter, month, day,
day_of_week, week_of_year)
        SELECT date_id, year, quarter, month, day, day_of_week, week_of_year
        FROM staging.staging_dim_date s
        ON CONFLICT (date_id) DO NOTHING;
        DROP TABLE IF EXISTS staging.staging_dim_date;
    '''))

with engine.begin() as con:
    # infer min/max from files
    sales = pd.read_csv(cfg['files']['sales'])
    min_d = pd.to_datetime(sales['date']).dt.date.min()
    max_d = pd.to_datetime(sales['date']).dt.date.max()
    load_dim_date(con, min_d, max_d)

# ---------- Load customers ----------
customers_df = pd.read_csv(cfg['files']['customers'])
with engine.begin() as con:
    for _, row in customers_df.iterrows():
        con.execute(UPSERT_CUSTOMER, row.to_dict())

# ---------- Load products ----------
with open(cfg['files']['products'],'r') as f:
    products = json.load(f)
with engine.begin() as con:
    for prod in products:
        con.execute(UPSERT_PRODUCT, prod)

# ---------- Helper: get SK from BK ----------
GET_CUSTOMER_SK = text("SELECT customer_sk FROM analytics.dim_customer WHERE
customer_id=:cid")
GET_PRODUCT_SK  = text("SELECT product_sk FROM analytics.dim_product WHERE
product_id=:pid")
GET_CHANNEL_SK  = text("SELECT channel_sk FROM analytics.dim_channel WHERE
channel_id=:chid")
```

```
# Cache small lookups
with engine.begin() as con:
    cust_sk = {r.customer_id: r.customer_sk for r in con.execute(text("SELECT
customer_id, customer_sk FROM analytics.dim_customer"))}
    prod_sk = {r.product_id: r.product_sk for r in con.execute(text("SELECT
product_id, product_sk FROM analytics.dim_product"))}
    ch_sk   = {r.channel_id: r.channel_sk for r in con.execute(text("SELECT
channel_id, channel_sk FROM analytics.dim_channel"))}

# ---------- Load sales fact (batched) ----------
sales_df = pd.read_csv(cfg['files']['sales'])
sales_df['date'] = pd.to_datetime(sales_df['date']).dt.date

# map BK -> SK
sales_df['customer_sk'] = sales_df['customer_id'].map(cust_sk)
sales_df['product_sk']  = sales_df['product_id'].map(prod_sk)
sales_df['channel_sk']  = sales_df['channel_id'].map(ch_sk)

sales_df = sales_df.rename(columns={'date':'date_id'})

# write batches to staging then merge
with engine.begin() as con:
    temp = 'staging_fact_sales'
    # ensure staging schema exists
    con.execute(text('CREATE SCHEMA IF NOT EXISTS staging;'))
    # chunked writes
    chunks = math.ceil(len(sales_df)/batch_size)
    for i in range(chunks):
        chunk = sales_df.iloc[i*batch_size:(i+1)*batch_size]
[['date_id','customer_sk','product_sk','channel_sk','quantity','revenue']]
        chunk.to_sql(temp, con, schema='staging', if_exists='append',
index=False)
    con.execute(text('''
        INSERT INTO analytics.fact_sales (date_id, customer_sk, product_sk,
channel_sk, quantity, revenue)
        SELECT date_id, customer_sk, product_sk, channel_sk, quantity, revenue
        FROM staging.staging_fact_sales;
        DROP TABLE IF EXISTS staging.staging_fact_sales;
    '''))

# ---------- Load web activity fact ----------
wa_df = pd.read_csv(cfg['files']['web_activity'])
wa_df['date'] = pd.to_datetime(wa_df['date']).dt.date
wa_df['customer_sk'] = wa_df['customer_id'].map(cust_sk)
wa_df['channel_sk']  = wa_df['channel_id'].map(ch_sk)
wa_df = wa_df.rename(columns={'date':'date_id'})

with engine.begin() as con:
```

```
    temp = 'staging_fact_web'
    chunks = math.ceil(len(wa_df)/batch_size)
    for i in range(chunks):
        chunk = wa_df.iloc[i*batch_size:(i+1)*batch_size]
[['date_id','customer_sk','channel_sk','event_type','session_id','page_url']]
        chunk.to_sql(temp, con, schema='staging', if_exists='append',
index=False)
    con.execute(text('''
        INSERT INTO analytics.fact_web_activity (date_id, customer_sk,
channel_sk, event_type, session_id, page_url)
        SELECT date_id, customer_sk, channel_sk, event_type, session_id,
page_url
        FROM staging.staging_fact_web;
        DROP TABLE IF EXISTS staging.staging_fact_web;
    '''))

print("ETL complete. Dims and facts loaded.")
```

📟Optional: Airflow DAG

airflow/dag_customer360.py (for reference; if you already run Airflow, point it to this file)

```
from airflow import DAG
from airflow.operators.bash import BashOperator
from datetime import datetime

def ts():
    return datetime.now()

default_args = {
    'owner': 'data-team',
    'depends_on_past': False,
    'retries': 0
}

dag = DAG(
    dag_id='customer360_dwh_daily',
    default_args=default_args,
    schedule_interval='@daily',
    start_date=datetime(2024,1,1),
    catchup=False
)

gen_data = BashOperator(
```

```
    task_id='generate_data',
    bash_command='python /opt/airflow/data/generate_data.py',
    dag=dag
)

run_etl = BashOperator(
    task_id='run_etl',
    bash_command='python /opt/airflow/etl/etl_load.py',
    dag=dag
)

gen_data >> run_etl
```

What This Demonstrates (Resume Bullets)

- **Designed a star-schema DWH** (Postgres) with `dim_customer`, `dim_product`, `dim_channel`, `dim_date`, `fact_sales`, `fact_web_activity` supporting OLAP queries.
- **Built robust ETL** in Python + SQLAlchemy with idempotent **SCD1 upserts** for dimensions, **batch loads** to facts, and a dedicated **staging** schema.
- **Integrated heterogeneous sources**: CSV (customers/sales), JSON (products), and simulated API-like web activity.
- **Automated orchestration**: optional Airflow DAG for daily runs.
- **Analytics**: CLV, category trends, and funnel metrics via SQL.

📈Next Enhancements (if you want to go deeper)

- Implement **SCD Type 2** for customer changes (effective_from/to, is_current).
- Add **data quality checks** (row counts, null checks) and a QA report.
- Introduce **dbt** for modular SQL transforms and documentation.
- Swap Postgres for **Redshift/Snowflake/BigQuery** to showcase cloud DWH skills.
- Add a **Power BI/Tableau** dashboard wired to the facts.

🔗Demo Steps Recap

1. `docker compose up -d` (brings up Postgres + Adminer)
2. `python data/generate_data.py`
3. `python etl/etl_load.py`
4. Run queries from `sql/99_sample_analytics.sql`

You now have a full, portfolio-ready DWH + ETL project that demonstrates integration of large/complex datasets and analytics on top. Paste this into a GitHub repo and you're good to go! ✅