SAMN'S MERCADO: END TO END E-COMMERCE PLATFORM

LEARNING TEAM 3 - ASIS, ITUCAL, MERCADO, RIZADA

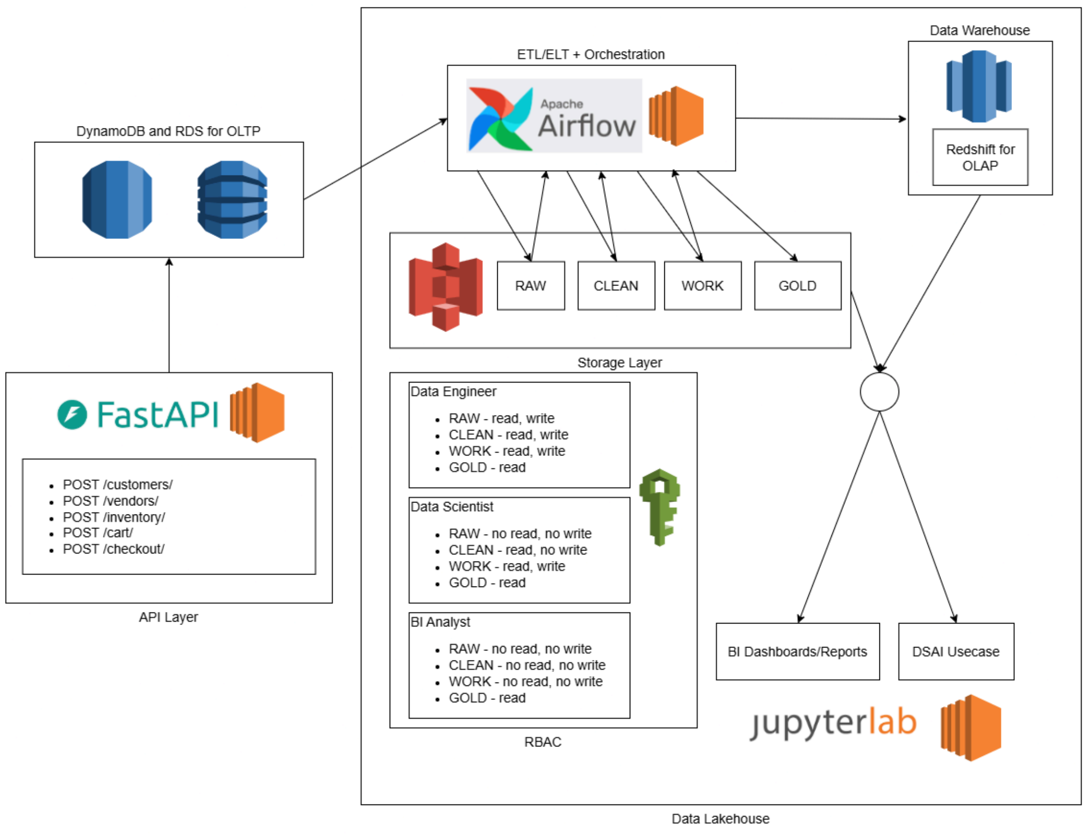
Introduction

This project showcases the data engineering skills learned in the elective by developing **Samn's Mercado**, an e-commerce platform designed to enhance both customer experience and vendor operations.

The platform allows customers to browse and purchase a diverse selection of products while equipping vendors with a powerful analytics suite built on data lakes and data warehouses. By leveraging Data Science, Analytics, and AI (DSAI), vendors can track sales performance, extract actionable insights, and optimize their business strategies.

Architecture Overview

End to end system architecture:

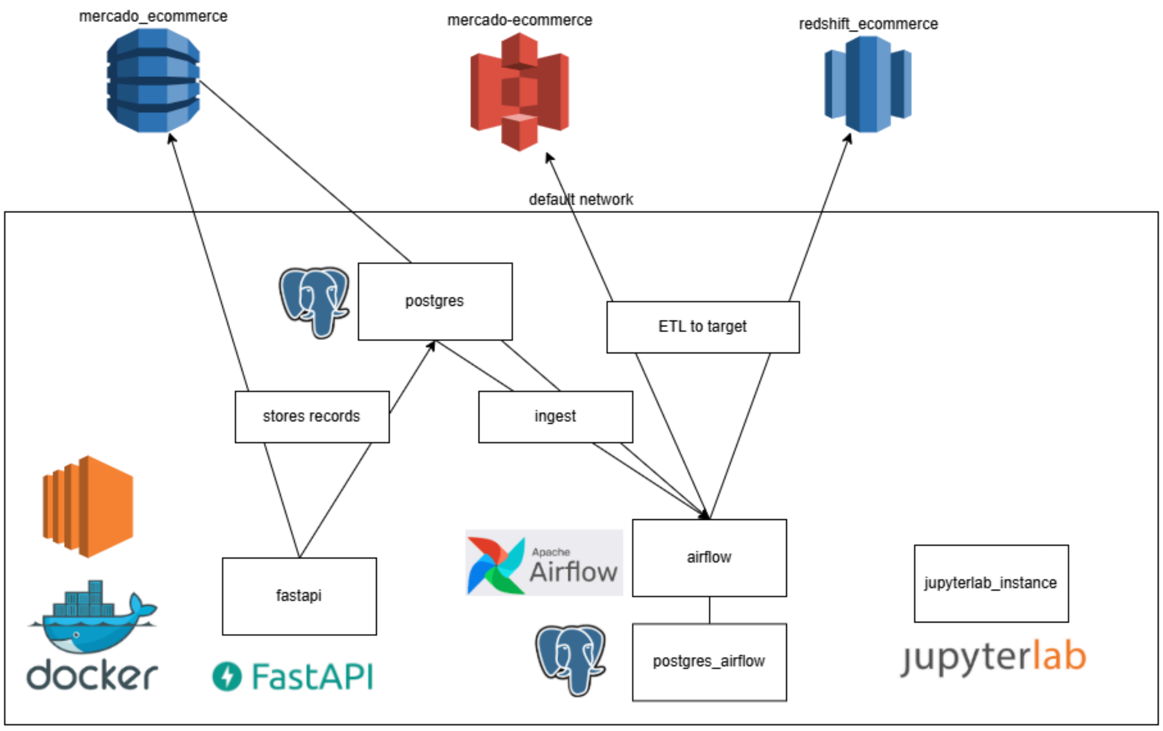


The end-to-end solution includes:

- **API Interfaces**: For data upload.
- **OLTP Processing**: Using DynamoDB and Postgres.
- **Orchestration and ETL/ELT**: Managed by Apache Airflow.
- **Data Lakehouse**: Centralized storage and processing.
- **Storage Layers**: Structured into raw, cleaned, work, and gold zones.

- **RBAC (Role-Based Access Control)**: Ensuring secure access to data.
- **OLTP Data Warehouse**: For structured data storage and analytics.
- **Consumption**: Data visualization and reporting.

Docker Architecture:



Services are deployed via Docker in an EC2 instance for ease of deployment and easy to spin up and tear down. The architecture includes:

- **Postgres for OLTP**: Instead of using RDS, a Postgres instance is contained within the Docker network.
- **Airflow + Postgres**: For orchestration and workflow management.
- **JupyterLab Container**: To run inserts and provide DSAI use cases in consumption.

Directory Structure

The project directory is organized as follows:

- `src` - source codes
  - `airflow` - airflow source codes
  - `api` - fastapi source codes
  - `dockerfiles` - Dockerfile definition for the services
  - `jupyterlab` - Notebooks and Images
  - `sql` - sql references for create tables
- `.github/workflows` - pre-commit hooks via github actions
- `sample_env` - sample env file to be filled up in an actual `.env file`, should specify details about connection details, and aws access keys.
- `.pre-commit-config.yaml` - pre-commit dependencies
- `docker-compose.yml` - docker compose definitions

API Layer

In this layer we use the API endpoints served via FastAPI to:

- Submit customers
- Submit vendors
- Submit inventory of vendors
- Simulate customer adding to cart, checking out and leaving items in their carts

In [1]:
```
%load_ext sql
```

In [2]:
```
import boto3
import requests
import os
import pandas as pd
from sqlalchemy import create_engine
from faker import Faker
import redshift_connector
```

In [3]:
```
import warnings
warnings.filterwarnings('ignore')
```

In [4]:
```
pd.set_option('display.max_rows', 12)
pd.set_option('display.min_rows', 6)
```

We will use faker to submit dummy API requests based on the definitions above.

In [5]:
```
fake = Faker()

base_url = "http://fastapi:8000"
headers = {"Content-Type": "application/json"}

print("customers")
# Create customers
for customer_id in range(1, 11):
    data = {
        "first_name": fake.first_name(),
        "last_name": fake.last_name(),
        "email": fake.email(),
        "joined_at": fake.iso8601()
    }
    response = requests.post(f"{base_url}/customers/", json=data, headers=headers)
    print(response.status_code, response.json())

print("vendors")
# Create vendors
for vendor_id in range(1, 6):
    data = {
        "vendor_name": fake.company(),
        "region": fake.country_code()
    }
    response = requests.post(f"{base_url}/vendors/", json=data, headers=headers)
    print(response.status_code, response.json())

print("inventory per vendor")
# Add inventory per vendor
for vendor_id in range(1, 6):
    for item_id in range(1, 11):
        params = {"vendor_id": vendor_id}
        data = {
            "item_name": fake.word(),
            "category": str(1),
            "price": fake.random_int(min=10, max=500),
        }
        response = requests.post(f"{base_url}/inventory/", json=data, headers=headers, params=param
        print(response.status_code, response.json())

print("cart and checkout twice and add to cart again!")
for customer_id in range(1, 11):
    for _ in range(2):  # Checkout twice
```

```
        price1 = fake.random_int(min=10, max=500)
        price2 = fake.random_int(min=10, max=500)
        qty1 = fake.random_int(min=10, max=500)
        qty2 = fake.random_int(min=10, max=500)
        cart_data = {
            "user_id": customer_id,
            "cart": [
                {
                    "item_id": str(fake.random_int(min=1, max=10)),
                    "vendor_id": str(fake.random_int(min=1, max=5)),
                    "qty": qty1,
                    "unit_price": price1,
                    "total_price": qty1*price1,
                },
                {
                    "item_id": str(fake.random_int(min=1, max=10)),
                    "vendor_id": str(fake.random_int(min=1, max=5)),
                    "qty": qty2,
                    "unit_price": price2,
                    "total_price": qty2*price2,
                }
            ]
        }
        response = requests.post(f"{base_url}/cart/", json=cart_data, headers=headers)
        print(response.status_code, response.json())

        response = requests.post(f"{base_url}/checkout/", json={"user_id": customer_id}, headers=he
        print(response.status_code, response.text)

    price1 = fake.random_int(min=10, max=500)
    price2 = fake.random_int(min=10, max=500)
    qty1 = fake.random_int(min=10, max=500)
    qty2 = fake.random_int(min=10, max=500)
    # Add to cart one more time
    cart_data = {
        "user_id": customer_id,
        "cart": [
            {
                "item_id": str(fake.random_int(min=1, max=10)),
                "vendor_id": str(fake.random_int(min=1, max=5)),
                "qty": qty1,
                "unit_price": price1,
                "total_price": qty1*price1,
            },
            {
                "item_id": str(fake.random_int(min=1, max=10)),
                "vendor_id": str(fake.random_int(min=1, max=5)),
                "qty": qty1,
                "unit_price": price1,
                "total_price": qty1*price1,
            }
        ]
    }
    response = requests.post(f"{base_url}/cart/", json=cart_data, headers=headers)
    print(response.status_code, response.json())
```

```
customers
201 {'id': 1, 'first_name': 'Brandon', 'last_name': 'Sims', 'email': 'adrian49@example.com', 'joine
d_at': '2025-03-17T02:33:40.378808'}
201 {'id': 2, 'first_name': 'Ronald', 'last_name': 'Johnson', 'email': 'igarcia@example.org', 'join
ed_at': '2025-03-17T02:33:40.388027'}
201 {'id': 3, 'first_name': 'William', 'last_name': 'Bryant', 'email': 'bobby28@example.org', 'join
ed_at': '2025-03-17T02:33:40.396208'}
201 {'id': 4, 'first_name': 'Sarah', 'last_name': 'Roberts', 'email': 'wlopez@example.org', 'joined
_at': '2025-03-17T02:33:40.403323'}
201 {'id': 5, 'first_name': 'Stacy', 'last_name': 'Riley', 'email': 'heather29@example.com', 'joine
d_at': '2025-03-17T02:33:40.411846'}
201 {'id': 6, 'first_name': 'Maria', 'last_name': 'Case', 'email': 'michaelrasmussen@example.net',
'joined_at': '2025-03-17T02:33:40.421316'}
201 {'id': 7, 'first_name': 'Barbara', 'last_name': 'James', 'email': 'scott38@example.com', 'joine
d_at': '2025-03-17T02:33:40.428812'}
201 {'id': 8, 'first_name': 'Joseph', 'last_name': 'Miller', 'email': 'nicolepreston@example.com',
'joined_at': '2025-03-17T02:33:40.437615'}
201 {'id': 9, 'first_name': 'Melissa', 'last_name': 'Thomas', 'email': 'carol19@example.com', 'join
ed_at': '2025-03-17T02:33:40.446074'}
201 {'id': 10, 'first_name': 'Roberta', 'last_name': 'Brown', 'email': 'lambertbryan@example.org',
'joined_at': '2025-03-17T02:33:40.456996'}
vendors
201 {'id': 1, 'vendor_name': 'Moyer-Vance', 'region': 'BZ', 'joined_at': '2025-03-17T02:33:40.46629
1'}
201 {'id': 2, 'vendor_name': 'Todd and Sons', 'region': 'MN', 'joined_at': '2025-03-17T02:33:40.475
478'}
201 {'id': 3, 'vendor_name': 'Adams PLC', 'region': 'SZ', 'joined_at': '2025-03-17T02:33:40.48274
5'}
201 {'id': 4, 'vendor_name': 'Mendoza Ltd', 'region': 'BZ', 'joined_at': '2025-03-17T02:33:40.49069
1'}
201 {'id': 5, 'vendor_name': 'Williams Inc', 'region': 'PS', 'joined_at': '2025-03-17T02:33:40.4988
85'}
inventory per vendor
200 {'id': 1, 'item_name': 'necessary', 'category': '1', 'price': 314.0, 'vendor_id': 1, 'updated_a
t': '2025-03-17T02:33:40.508823'}
200 {'id': 2, 'item_name': 'consumer', 'category': '1', 'price': 79.0, 'vendor_id': 1, 'updated_a
t': '2025-03-17T02:33:40.517968'}
200 {'id': 3, 'item_name': 'begin', 'category': '1', 'price': 334.0, 'vendor_id': 1, 'updated_at':
'2025-03-17T02:33:40.525865'}
200 {'id': 4, 'item_name': 'already', 'category': '1', 'price': 285.0, 'vendor_id': 1, 'updated_a
t': '2025-03-17T02:33:40.535122'}
200 {'id': 5, 'item_name': 'charge', 'category': '1', 'price': 32.0, 'vendor_id': 1, 'updated_at':
'2025-03-17T02:33:40.542649'}
200 {'id': 6, 'item_name': 'agent', 'category': '1', 'price': 496.0, 'vendor_id': 1, 'updated_at':
'2025-03-17T02:33:40.550962'}
200 {'id': 7, 'item_name': 'board', 'category': '1', 'price': 103.0, 'vendor_id': 1, 'updated_at':
'2025-03-17T02:33:40.560429'}
200 {'id': 8, 'item_name': 'book', 'category': '1', 'price': 296.0, 'vendor_id': 1, 'updated_at':
'2025-03-17T02:33:40.569722'}
200 {'id': 9, 'item_name': 'service', 'category': '1', 'price': 207.0, 'vendor_id': 1, 'updated_a
t': '2025-03-17T02:33:40.578550'}
200 {'id': 10, 'item_name': 'window', 'category': '1', 'price': 475.0, 'vendor_id': 1, 'updated_a
t': '2025-03-17T02:33:40.586792'}
200 {'id': 11, 'item_name': 'guy', 'category': '1', 'price': 328.0, 'vendor_id': 2, 'updated_at':
'2025-03-17T02:33:40.594762'}
200 {'id': 12, 'item_name': 'toward', 'category': '1', 'price': 54.0, 'vendor_id': 2, 'updated_at':
'2025-03-17T02:33:40.603811'}
200 {'id': 13, 'item_name': 'stuff', 'category': '1', 'price': 49.0, 'vendor_id': 2, 'updated_at':
'2025-03-17T02:33:40.612914'}
200 {'id': 14, 'item_name': 'too', 'category': '1', 'price': 56.0, 'vendor_id': 2, 'updated_at': '2
025-03-17T02:33:40.623289'}
200 {'id': 15, 'item_name': 'still', 'category': '1', 'price': 338.0, 'vendor_id': 2, 'updated_at':
'2025-03-17T02:33:40.631919'}
200 {'id': 16, 'item_name': 'opportunity', 'category': '1', 'price': 454.0, 'vendor_id': 2, 'update
d_at': '2025-03-17T02:33:40.639421'}
200 {'id': 17, 'item_name': 'table', 'category': '1', 'price': 254.0, 'vendor_id': 2, 'updated_at':
'2025-03-17T02:33:40.648303'}
200 {'id': 18, 'item_name': 'job', 'category': '1', 'price': 425.0, 'vendor_id': 2, 'updated_at':
'2025-03-17T02:33:40.656069'}
200 {'id': 19, 'item_name': 'other', 'category': '1', 'price': 344.0, 'vendor_id': 2, 'updated_at':
```

```
'2025-03-17T02:33:40.664236'}
200 {'id': 20, 'item_name': 'standard', 'category': '1', 'price': 438.0, 'vendor_id': 2, 'updated_a
t': '2025-03-17T02:33:40.672717'}
200 {'id': 21, 'item_name': 'spend', 'category': '1', 'price': 348.0, 'vendor_id': 3, 'updated_at':
'2025-03-17T02:33:40.681874'}
200 {'id': 22, 'item_name': 'indicate', 'category': '1', 'price': 95.0, 'vendor_id': 3, 'updated_a
t': '2025-03-17T02:33:40.690143'}
200 {'id': 23, 'item_name': 'management', 'category': '1', 'price': 189.0, 'vendor_id': 3, 'updated
_at': '2025-03-17T02:33:40.698969'}
200 {'id': 24, 'item_name': 'speak', 'category': '1', 'price': 203.0, 'vendor_id': 3, 'updated_at':
'2025-03-17T02:33:40.709145'}
200 {'id': 25, 'item_name': 'water', 'category': '1', 'price': 222.0, 'vendor_id': 3, 'updated_at':
'2025-03-17T02:33:40.718118'}
200 {'id': 26, 'item_name': 'rate', 'category': '1', 'price': 92.0, 'vendor_id': 3, 'updated_at':
'2025-03-17T02:33:40.728106'}
200 {'id': 27, 'item_name': 'civil', 'category': '1', 'price': 462.0, 'vendor_id': 3, 'updated_at':
'2025-03-17T02:33:40.737006'}
200 {'id': 28, 'item_name': 'official', 'category': '1', 'price': 474.0, 'vendor_id': 3, 'updated_a
t': '2025-03-17T02:33:40.744976'}
200 {'id': 29, 'item_name': 'whole', 'category': '1', 'price': 437.0, 'vendor_id': 3, 'updated_at':
'2025-03-17T02:33:40.753876'}
200 {'id': 30, 'item_name': 'provide', 'category': '1', 'price': 263.0, 'vendor_id': 3, 'updated_a
t': '2025-03-17T02:33:40.762176'}
200 {'id': 31, 'item_name': 'again', 'category': '1', 'price': 210.0, 'vendor_id': 4, 'updated_at':
'2025-03-17T02:33:40.769871'}
200 {'id': 32, 'item_name': 'next', 'category': '1', 'price': 89.0, 'vendor_id': 4, 'updated_at':
'2025-03-17T02:33:40.778397'}
200 {'id': 33, 'item_name': 'moment', 'category': '1', 'price': 489.0, 'vendor_id': 4, 'updated_a
t': '2025-03-17T02:33:40.786756'}
200 {'id': 34, 'item_name': 'industry', 'category': '1', 'price': 491.0, 'vendor_id': 4, 'updated_a
t': '2025-03-17T02:33:40.794251'}
200 {'id': 35, 'item_name': 'present', 'category': '1', 'price': 366.0, 'vendor_id': 4, 'updated_a
t': '2025-03-17T02:33:40.802494'}
200 {'id': 36, 'item_name': 'civil', 'category': '1', 'price': 28.0, 'vendor_id': 4, 'updated_at':
'2025-03-17T02:33:40.810669'}
200 {'id': 37, 'item_name': 'not', 'category': '1', 'price': 284.0, 'vendor_id': 4, 'updated_at':
'2025-03-17T02:33:40.818316'}
200 {'id': 38, 'item_name': 'growth', 'category': '1', 'price': 297.0, 'vendor_id': 4, 'updated_a
t': '2025-03-17T02:33:40.826259'}
200 {'id': 39, 'item_name': 'good', 'category': '1', 'price': 188.0, 'vendor_id': 4, 'updated_at':
'2025-03-17T02:33:40.833770'}
200 {'id': 40, 'item_name': 'wait', 'category': '1', 'price': 288.0, 'vendor_id': 4, 'updated_at':
'2025-03-17T02:33:40.843051'}
200 {'id': 41, 'item_name': 'technology', 'category': '1', 'price': 412.0, 'vendor_id': 5, 'updated
_at': '2025-03-17T02:33:40.851317'}
200 {'id': 42, 'item_name': 'raise', 'category': '1', 'price': 461.0, 'vendor_id': 5, 'updated_at':
'2025-03-17T02:33:40.860822'}
200 {'id': 43, 'item_name': 'agent', 'category': '1', 'price': 209.0, 'vendor_id': 5, 'updated_at':
'2025-03-17T02:33:40.870142'}
200 {'id': 44, 'item_name': 'identify', 'category': '1', 'price': 136.0, 'vendor_id': 5, 'updated_a
t': '2025-03-17T02:33:40.878949'}
200 {'id': 45, 'item_name': 'growth', 'category': '1', 'price': 274.0, 'vendor_id': 5, 'updated_a
t': '2025-03-17T02:33:40.885689'}
200 {'id': 46, 'item_name': 'new', 'category': '1', 'price': 472.0, 'vendor_id': 5, 'updated_at':
'2025-03-17T02:33:40.895083'}
200 {'id': 47, 'item_name': 'sport', 'category': '1', 'price': 23.0, 'vendor_id': 5, 'updated_at':
'2025-03-17T02:33:40.903467'}
200 {'id': 48, 'item_name': 'section', 'category': '1', 'price': 237.0, 'vendor_id': 5, 'updated_a
t': '2025-03-17T02:33:40.912562'}
200 {'id': 49, 'item_name': 'popular', 'category': '1', 'price': 124.0, 'vendor_id': 5, 'updated_a
t': '2025-03-17T02:33:40.919968'}
200 {'id': 50, 'item_name': 'theory', 'category': '1', 'price': 227.0, 'vendor_id': 5, 'updated_a
t': '2025-03-17T02:33:40.926696'}
cart and checkout twice and add to cart again!
200 {'user_id': 1, 'cart': [{'item_id': '1', 'qty': 12, 'vendor_id': '3', 'unit_price': '209', 'tot
al_price': '2508'}, {'item_id': '7', 'qty': 486, 'vendor_id': '1', 'unit_price': '354', 'total_pric
e': '172044'}]}
200 {"message":"Transaction completed","transaction_id":"cd019aa9-a29c-401c-ba45-0a1ffbf1e04c"}
200 {'user_id': 1, 'cart': [{'item_id': '9', 'qty': 182, 'vendor_id': '3', 'unit_price': '12', 'tot
al_price': '2184'}, {'item_id': '5', 'qty': 162, 'vendor_id': '1', 'unit_price': '392', 'total_pric
```

```
e': '63504'}]}
200 {"message":"Transaction completed","transaction_id":"f65a97d8-d16e-4f6d-9d0f-3db9a7b34d78"}
200 {'user_id': 1, 'cart': [{'item_id': '6', 'qty': 139, 'vendor_id': '2', 'unit_price': '92', 'tot
al_price': '12788'}, {'item_id': '5', 'qty': 139, 'vendor_id': '3', 'unit_price': '92', 'total_pric
e': '12788'}]}
200 {'user_id': 2, 'cart': [{'item_id': '7', 'qty': 40, 'vendor_id': '1', 'unit_price': '94', 'tota
l_price': '3760'}, {'item_id': '10', 'qty': 475, 'vendor_id': '1', 'unit_price': '192', 'total_pric
e': '91200'}]}
200 {"message":"Transaction completed","transaction_id":"c2020d4d-2aeb-480c-b4b8-a3ba3cc99ba7"}
200 {'user_id': 2, 'cart': [{'item_id': '8', 'qty': 218, 'vendor_id': '2', 'unit_price': '54', 'tot
al_price': '11772'}, {'item_id': '2', 'qty': 47, 'vendor_id': '5', 'unit_price': '432', 'total_pric
e': '20304'}]}
200 {"message":"Transaction completed","transaction_id":"b99634b3-f13c-4194-b651-c73f867493ef"}
200 {'user_id': 2, 'cart': [{'item_id': '1', 'qty': 450, 'vendor_id': '1', 'unit_price': '151', 'to
tal_price': '67950'}, {'item_id': '9', 'qty': 450, 'vendor_id': '5', 'unit_price': '151', 'total_pr
ice': '67950'}]}
200 {'user_id': 3, 'cart': [{'item_id': '4', 'qty': 153, 'vendor_id': '3', 'unit_price': '212', 'to
tal_price': '32436'}, {'item_id': '2', 'qty': 83, 'vendor_id': '1', 'unit_price': '377', 'total_pri
ce': '31291'}]}
200 {"message":"Transaction completed","transaction_id":"886f753a-fdf6-4bf0-99a7-a20d0ee24075"}
200 {'user_id': 3, 'cart': [{'item_id': '9', 'qty': 159, 'vendor_id': '4', 'unit_price': '141', 'to
tal_price': '22419'}, {'item_id': '6', 'qty': 26, 'vendor_id': '5', 'unit_price': '441', 'total_pri
ce': '11466'}]}
200 {"message":"Transaction completed","transaction_id":"be5ad69b-9604-4cd6-89aa-dc6303ab5816"}
200 {'user_id': 3, 'cart': [{'item_id': '7', 'qty': 41, 'vendor_id': '4', 'unit_price': '219', 'tot
al_price': '8979'}, {'item_id': '1', 'qty': 41, 'vendor_id': '1', 'unit_price': '219', 'total_pric
e': '8979'}]}
200 {'user_id': 4, 'cart': [{'item_id': '8', 'qty': 34, 'vendor_id': '2', 'unit_price': '401', 'tot
al_price': '13634'}, {'item_id': '2', 'qty': 257, 'vendor_id': '3', 'unit_price': '307', 'total_pri
ce': '78899'}]}
200 {"message":"Transaction completed","transaction_id":"acbe7f2c-92cb-4ce8-8513-268dac1188e6"}
200 {'user_id': 4, 'cart': [{'item_id': '9', 'qty': 236, 'vendor_id': '5', 'unit_price': '88', 'tot
al_price': '20768'}, {'item_id': '9', 'qty': 65, 'vendor_id': '2', 'unit_price': '372', 'total_pric
e': '24180'}]}
200 {"message":"Transaction completed","transaction_id":"2e5aef87-bb55-40c6-b801-dbc39e9a5b1a"}
200 {'user_id': 4, 'cart': [{'item_id': '6', 'qty': 216, 'vendor_id': '3', 'unit_price': '199', 'to
tal_price': '42984'}, {'item_id': '7', 'qty': 216, 'vendor_id': '4', 'unit_price': '199', 'total_pr
ice': '42984'}]}
200 {'user_id': 5, 'cart': [{'item_id': '3', 'qty': 291, 'vendor_id': '4', 'unit_price': '247', 'to
tal_price': '71877'}, {'item_id': '10', 'qty': 363, 'vendor_id': '3', 'unit_price': '98', 'total_pr
ice': '35574'}]}
200 {"message":"Transaction completed","transaction_id":"8c8de6b8-bc43-42a5-90a5-c098c9fe0192"}
200 {'user_id': 5, 'cart': [{'item_id': '4', 'qty': 413, 'vendor_id': '2', 'unit_price': '258', 'to
tal_price': '106554'}, {'item_id': '4', 'qty': 284, 'vendor_id': '2', 'unit_price': '15', 'total_pr
ice': '4260'}]}
200 {"message":"Transaction completed","transaction_id":"743ade43-dcab-4078-aaeb-ccfb5387e3ea"}
200 {'user_id': 5, 'cart': [{'item_id': '5', 'qty': 228, 'vendor_id': '1', 'unit_price': '202', 'to
tal_price': '46056'}, {'item_id': '1', 'qty': 228, 'vendor_id': '3', 'unit_price': '202', 'total_pr
ice': '46056'}]}
200 {'user_id': 6, 'cart': [{'item_id': '1', 'qty': 66, 'vendor_id': '1', 'unit_price': '277', 'tot
al_price': '18282'}, {'item_id': '8', 'qty': 63, 'vendor_id': '4', 'unit_price': '246', 'total_pric
e': '15498'}]}
200 {"message":"Transaction completed","transaction_id":"fa909580-a169-49b2-8ad1-2bfdb3492e56"}
200 {'user_id': 6, 'cart': [{'item_id': '1', 'qty': 374, 'vendor_id': '5', 'unit_price': '150', 'to
tal_price': '56100'}, {'item_id': '7', 'qty': 180, 'vendor_id': '3', 'unit_price': '381', 'total_pr
ice': '68580'}]}
200 {"message":"Transaction completed","transaction_id":"170fdaaa-f2e6-4809-b42c-23357be591c3"}
200 {'user_id': 6, 'cart': [{'item_id': '2', 'qty': 283, 'vendor_id': '3', 'unit_price': '203', 'to
tal_price': '57449'}, {'item_id': '2', 'qty': 283, 'vendor_id': '3', 'unit_price': '203', 'total_pr
ice': '57449'}]}
200 {'user_id': 7, 'cart': [{'item_id': '8', 'qty': 405, 'vendor_id': '2', 'unit_price': '46', 'tot
al_price': '18630'}, {'item_id': '3', 'qty': 310, 'vendor_id': '5', 'unit_price': '145', 'total_pri
ce': '44950'}]}
200 {"message":"Transaction completed","transaction_id":"0ce03b4e-0d43-4758-a8d0-89aae5e7dd01"}
200 {'user_id': 7, 'cart': [{'item_id': '1', 'qty': 343, 'vendor_id': '5', 'unit_price': '233', 'to
tal_price': '79919'}, {'item_id': '2', 'qty': 203, 'vendor_id': '4', 'unit_price': '124', 'total_pr
ice': '25172'}]}
200 {"message":"Transaction completed","transaction_id":"5981a307-48c4-41c5-8b39-1a810d21f877"}
200 {'user_id': 7, 'cart': [{'item_id': '6', 'qty': 17, 'vendor_id': '4', 'unit_price': '203', 'tot
al_price': '3451'}, {'item_id': '10', 'qty': 17, 'vendor_id': '2', 'unit_price': '203', 'total_pric
```

```
e': '3451'}]}
200 {'user_id': 8, 'cart': [{'item_id': '9', 'qty': 424, 'vendor_id': '3', 'unit_price': '297', 'to
tal_price': '125928'}, {'item_id': '7', 'qty': 165, 'vendor_id': '5', 'unit_price': '267', 'total_p
rice': '44055'}]}
200 {"message":"Transaction completed","transaction_id":"3806ea7b-6719-44df-9d2a-cf315efa4ccb"}
200 {'user_id': 8, 'cart': [{'item_id': '4', 'qty': 311, 'vendor_id': '3', 'unit_price': '125', 'to
tal_price': '38875'}, {'item_id': '6', 'qty': 357, 'vendor_id': '4', 'unit_price': '384', 'total_pr
ice': '137088'}]}
200 {"message":"Transaction completed","transaction_id":"99ab131a-fe9c-462a-b2fb-0b22d4904d3b"}
200 {'user_id': 8, 'cart': [{'item_id': '2', 'qty': 477, 'vendor_id': '2', 'unit_price': '20', 'tot
al_price': '9540'}, {'item_id': '10', 'qty': 477, 'vendor_id': '5', 'unit_price': '20', 'total_pric
e': '9540'}]}
200 {'user_id': 9, 'cart': [{'item_id': '4', 'qty': 29, 'vendor_id': '2', 'unit_price': '306', 'tot
al_price': '8874'}, {'item_id': '2', 'qty': 21, 'vendor_id': '3', 'unit_price': '52', 'total_pric
e': '1092'}]}
200 {"message":"Transaction completed","transaction_id":"b5a98d65-60e3-4817-bbed-165404bea440"}
200 {'user_id': 9, 'cart': [{'item_id': '7', 'qty': 340, 'vendor_id': '3', 'unit_price': '413', 'to
tal_price': '140420'}, {'item_id': '6', 'qty': 310, 'vendor_id': '5', 'unit_price': '392', 'total_p
rice': '121520'}]}
200 {"message":"Transaction completed","transaction_id":"7900a407-1ccc-4026-99e5-65f197141c93"}
200 {'user_id': 9, 'cart': [{'item_id': '6', 'qty': 131, 'vendor_id': '3', 'unit_price': '278', 'to
tal_price': '36418'}, {'item_id': '4', 'qty': 131, 'vendor_id': '4', 'unit_price': '278', 'total_pr
ice': '36418'}]}
200 {'user_id': 10, 'cart': [{'item_id': '2', 'qty': 75, 'vendor_id': '2', 'unit_price': '151', 'to
tal_price': '11325'}, {'item_id': '8', 'qty': 172, 'vendor_id': '3', 'unit_price': '442', 'total_pr
ice': '76024'}]}
200 {"message":"Transaction completed","transaction_id":"14749d2a-5d2f-4880-b848-c0eb419db0a0"}
200 {'user_id': 10, 'cart': [{'item_id': '1', 'qty': 53, 'vendor_id': '1', 'unit_price': '129', 'to
tal_price': '6837'}, {'item_id': '2', 'qty': 428, 'vendor_id': '4', 'unit_price': '290', 'total_pri
ce': '124120'}]}
200 {"message":"Transaction completed","transaction_id":"f23bc93c-4cb4-40e4-90d3-95e1ee81a4e5"}
200 {'user_id': 10, 'cart': [{'item_id': '4', 'qty': 288, 'vendor_id': '5', 'unit_price': '479', 't
otal_price': '137952'}, {'item_id': '7', 'qty': 288, 'vendor_id': '1', 'unit_price': '479', 'total_
price': '137952'}]}
```

In [6]:
```
OLTP_USER = os.getenv("POSTGRES_USER")
OLTP_PASS = os.getenv("POSTGRES_PASSWORD")
OLTP_HOST = os.getenv("POSTGRES_HOST")
OLTP_DB_NAME = os.getenv("POSTGRES_DB")
OLAP_USER = os.getenv("REDSHIFT_USER")
OLAP_PASS = os.getenv("REDSHIFT_PASSWORD")
OLAP_HOST = os.getenv("REDSHIFT_HOST")
OLAP_DB_NAME = os.getenv("REDSHIFT_DB")
```

In [7]:
```
connection_string = f"postgresql://{OLTP_USER}:{OLTP_PASS}@{OLTP_HOST}:5432/{OLTP_DB_NAME}"
engine = create_engine(connection_string)
get_ipython().run_line_magic('sql', connection_string)
```

```
Connecting to 'postgresql://user:***@db:5432/ecommerce'
```

Viewing OLTP Layer - PostgreSQL and NoSQL

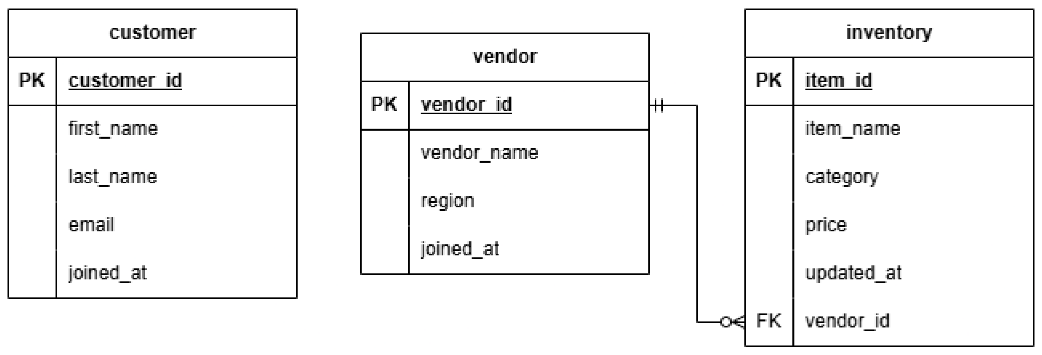
The OLTP layer comprises a **relational database (PostgreSQL)** and a **NoSQL database (DynamoDB)** to efficiently manage transactional data, including user accounts, orders, and payments.

Database Structure

- **PostgreSQL (RDS)** – Ideal for structured data and complex queries:
  - `customer` table
  - `vendor` table
  - `inventory` table
- **DynamoDB (NoSQL)** – Optimized for high-speed, flexible data storage:
  - `cart details`
  - `customer transaction history`

Postgres side

ERD:



PostgreSQL is ideal for managing core entities in an e-commerce platform, such as customers, vendors, and product inventories.

Customer-side benefits:

- **Efficient product searches** using structured queries (e.g., searching by `item_name` or `vendor_name`).
- **Quick user lookups** by attributes like `email` or `first_name`, improving account and order management.

Vendor-side benefits:

- **Seamless vendor management**, allowing quick lookup of clients by client name or `email`.
- **Inventory tracking**, ensuring vendors can monitor stock levels and pricing in real time.

*Create table definition found in `src\sql\*` directory

In [8]:
```
%sql \dt
```

```
Running query in 'postgresql://user:***@db:5432/ecommerce'
```

Out[8]:

Schema	Name	Type	Owner
public	customer	table	user
public	inventory	table	user
public	vendor	table	user

In [9]:
```
table_list = [
    "customer", "inventory", "vendor"
]
```

In [10]:
```
for table in table_list:
    print(f"{table}")
    schema_query = f"columns --table {table}"
    display(get_ipython().run_line_magic('sqlcmd', schema_query))
```

```
customer
```

name	type	nullable	default	autoincrement	comment
id	INTEGER	False	nextval('customer_id_seq'::regclass)	True	None
first_name	TEXT	False	None	False	None
last_name	TEXT	False	None	False	None
email	TEXT	False	None	False	None
joined_at	TIMESTAMP	False	None	False	None

```
inventory
```

name	type	nullable	default	autoincrement	comment
id	INTEGER	False	nextval('inventory_id_seq'::regclass)	True	None
item_name	TEXT	False	None	False	None
category	TEXT	False	None	False	None
price	INTEGER	False	None	False	None
updated_at	TIMESTAMP	False	None	False	None
vendor_id	INTEGER	True	None	False	None

```
vendor
```

name	type	nullable	default	autoincrement	comment
id	INTEGER	False	nextval('vendor_id_seq'::regclass)	True	None
vendor_name	TEXT	False	None	False	None
region	TEXT	False	None	False	None
joined_at	TIMESTAMP	False	None	False	None

In [11]:
```
for table in table_list:
    print(f"{table}")
    display(pd.read_sql(f"select * from {table} limit 100", engine.raw_connection()))
```

```
customer
```

	id	first_name	last_name	email	joined_at
0	1	Brandon	Sims	adrian49@example.com	2025-03-17 02:33:40.378808
1	2	Ronald	Johnson	igarcia@example.org	2025-03-17 02:33:40.388027
2	3	William	Bryant	bobby28@example.org	2025-03-17 02:33:40.396208
3	4	Sarah	Roberts	wlopez@example.org	2025-03-17 02:33:40.403323
4	5	Stacy	Riley	heather29@example.com	2025-03-17 02:33:40.411846
5	6	Maria	Case	michaelrasmussen@example.net	2025-03-17 02:33:40.421316
6	7	Barbara	James	scott38@example.com	2025-03-17 02:33:40.428812
7	8	Joseph	Miller	nicolepreston@example.com	2025-03-17 02:33:40.437615
8	9	Melissa	Thomas	carol19@example.com	2025-03-17 02:33:40.446074
9	10	Roberta	Brown	lambertbryan@example.org	2025-03-17 02:33:40.456996

```
inventory
```

	id	item_name	category	price	updated_at	vendor_id
0	1	necessary	1	314	2025-03-17 02:33:40.508823	1
1	2	consumer	1	79	2025-03-17 02:33:40.517968	1
2	3	begin	1	334	2025-03-17 02:33:40.525865	1
...	...	...	...	...	...	...
47	48	section	1	237	2025-03-17 02:33:40.912562	5
48	49	popular	1	124	2025-03-17 02:33:40.919968	5
49	50	theory	1	227	2025-03-17 02:33:40.926696	5

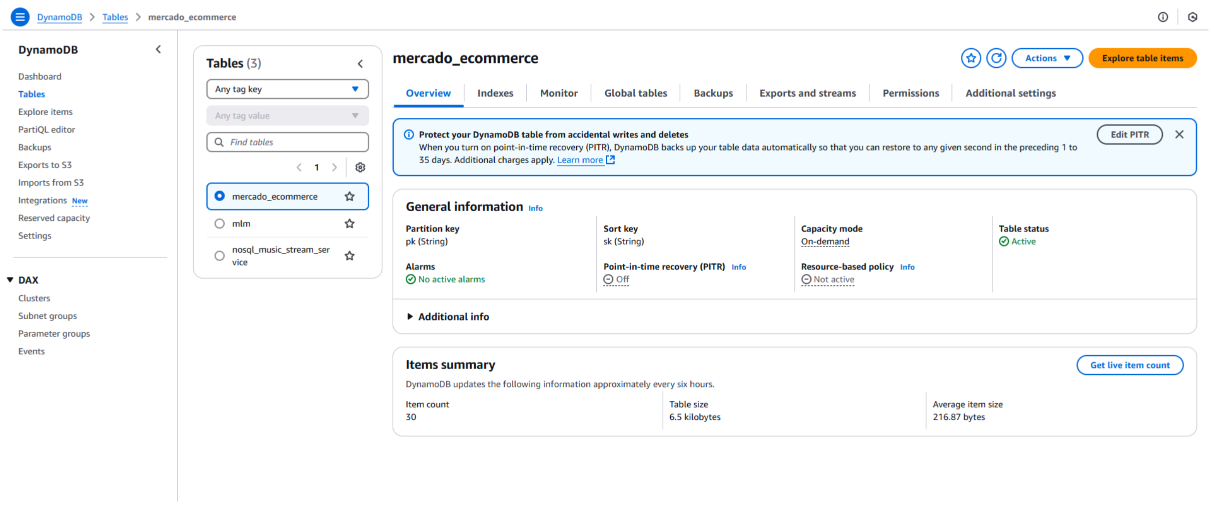
50 rows × 6 columns

```
vendor
```

	id	vendor_name	region	joined_at
0	1	Moyer-Vance	BZ	2025-03-17 02:33:40.466291
1	2	Todd and Sons	MN	2025-03-17 02:33:40.475478
2	3	Adams PLC	SZ	2025-03-17 02:33:40.482745
3	4	Mendoza Ltd	BZ	2025-03-17 02:33:40.490691
4	5	Williams Inc	PS	2025-03-17 02:33:40.498885

DynamoDB side

ERD/Entity Containment in NoSQL:



mercado_ecommerce

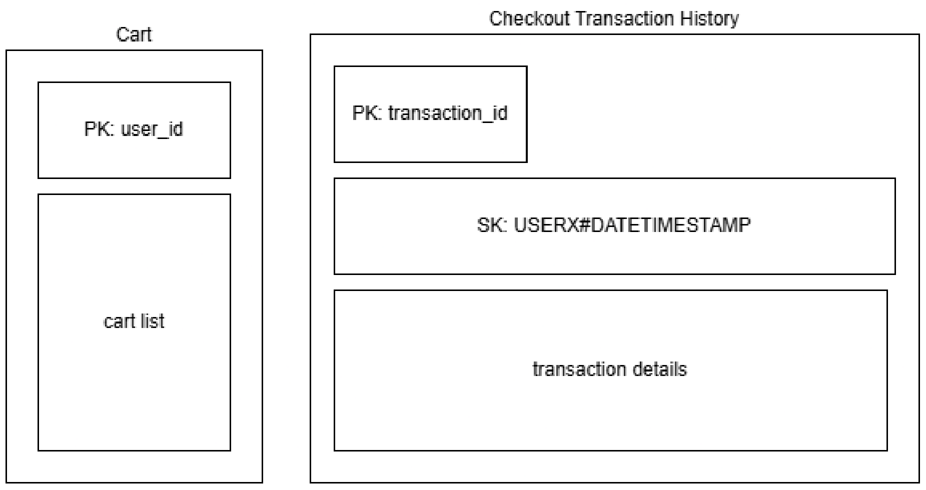
Protect your DynamoDB table from accidental writes and deletes
When you turn on point-in-time recovery (PITR), DynamoDB backs up your table data automatically so that you can restore to any given second in the preceding 1 to 35 days. Additional charges apply. Learn more

General information

Partition key	Sort key	Capacity mode	Table status
pk (String)	sk (String)	On-demand	Active
Alarms	**Point-in-time recovery (PITR)**	**Resource-based policy**	
No active alarms	Off	Not active	

Items summary
DynamoDB updates the following information approximately every six hours.

Item count	Table size	Average item size
30	6.5 kilobytes	216.87 bytes



DynamoDB is chosen for handling **frequently changing data** such as shopping carts and transactions due to its **schema flexibility, high-performance read/write capabilities, and scalability**.

- The **Cart table** uses `user_id` as the **Partition Key (PK)**, allowing it to store a **nested list of items**, enabling **real-time updates** as users modify their carts.
- The **Transaction table** is structured with `transaction_id` as the **Partition Key (PK)** and `user_id#timestamp` as the **Sort Key (SK)**, ensuring **efficient lookups and chronological ordering** of transactions per user.

DynamoDB's **scalability and low-latency access** make it an ideal choice for **real-time cart updates and seamless transaction history management**, ensuring a smooth and responsive shopping experience in a dynamic e-commerce environment.

```
In [12]: dynamodb = boto3.resource("dynamodb", "us-east-1")
         table = dynamodb.Table("mercado_ecommerce")
         table.scan()['Items'][:3]
```

```
Out[12]: [{'updated_at': '2025-03-17T02:33:41.460999',
           'sk': 'CART',
           'pk': 'USER#8',
           'cart': [{'unit_price': Decimal('20'),
             'total_price': Decimal('9540'),
             'item_id': '2',
             'qty': Decimal('477'),
             'vendor_id': '2'},
            {'unit_price': Decimal('20'),
             'total_price': Decimal('9540'),
             'item_id': '10',
             'qty': Decimal('477'),
             'vendor_id': '5'}]},
          {'sk': 'USER6#2025-03-17T02:33:41.331368',
           'created_at': '2025-03-17T02:33:41.331368',
           'pk': '170fdaaa-f2e6-4809-b42c-23357be591c3',
           'cart': [{'unit_price': Decimal('150'),
             'total_price': Decimal('56100'),
             'item_id': '1',
             'qty': Decimal('374'),
             'vendor_id': '5'},
            {'unit_price': Decimal('381'),
             'total_price': Decimal('68580'),
             'item_id': '7',
             'qty': Decimal('180'),
             'vendor_id': '3'}]},
          {'updated_at': '2025-03-17T02:33:41.099113',
           'sk': 'CART',
           'pk': 'USER#2',
           'cart': [{'unit_price': Decimal('151'),
             'total_price': Decimal('67950'),
             'item_id': '1',
             'qty': Decimal('450'),
             'vendor_id': '1'},
            {'unit_price': Decimal('151'),
             'total_price': Decimal('67950'),
             'item_id': '9',
             'qty': Decimal('450'),
             'vendor_id': '5'}]}]
```

Worflow Management and Data Pipeline

This section outlines the **workflow deployed in Apache Airflow**.

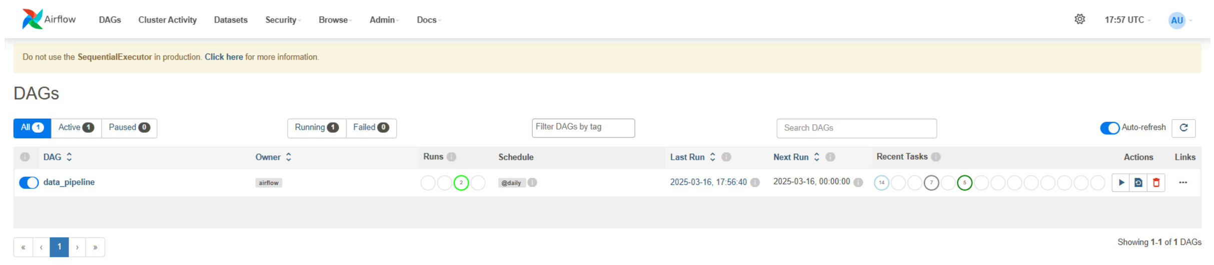
Airflow DAG Overview The **Directed Acyclic Graph (DAG)** automates the following key tasks:

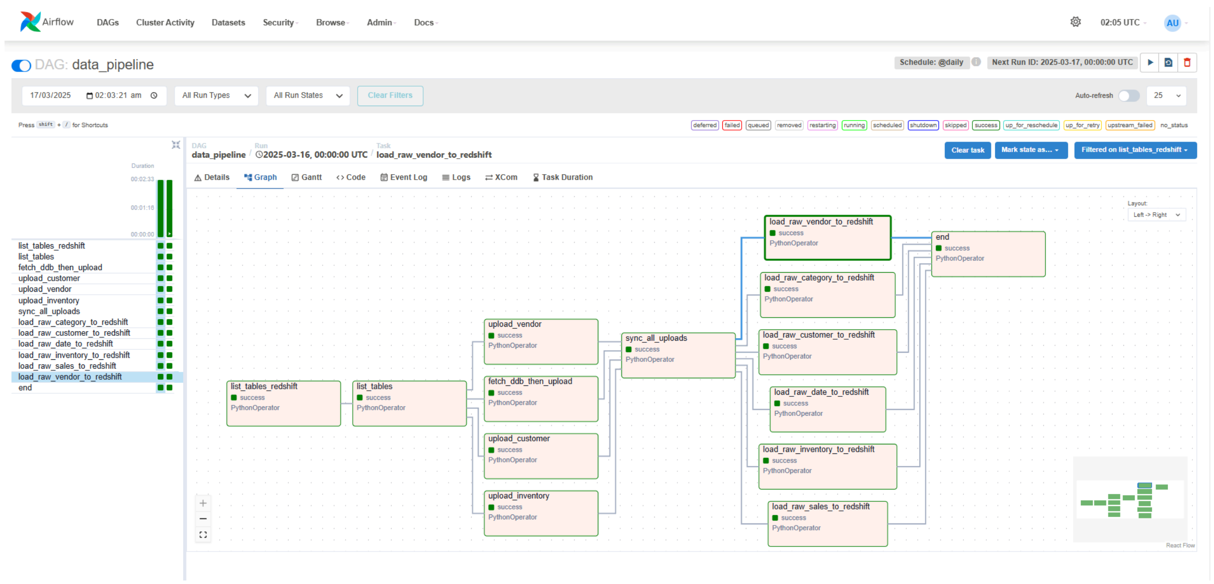
1. **Fetch Data from OLTP Sources**
   - Extracts transactional data from **PostgreSQL** and **DynamoDB**.
2. **Store Data in S3 Zones**
   - Loads extracted data into Amazon S3, categorized into different zones:
     - `raw` – Unprocessed data as extracted from OLTP sources.
     - `cleaned` – Data after initial cleaning and transformation.
     - `work` – Intermediate datasets used for analytics and processing.
     - `gold` – Final, structured datasets ready for analysis.
3. **Ingest Data into Redshift**
   - Transfers data from the `gold` **zone** in S3 to **Amazon Redshift**, the data warehouse, for analytics and reporting.

Processing Mode

- The DAG operates in **batch processing mode**, running **daily** to ensure timely updates and maintain data consistency across the pipeline.

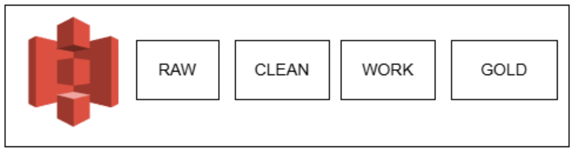
*DAG technical code definition found in `src\airflow\dags\*` directory





Data Lake

Storage Layer



Storage Layer

The **data lake storage layer** is structured into four key zones—**raw, cleaned, work, and gold**—each serving a distinct role in the data processing pipeline.

1. **Raw Zone**
   - Acts as the **landing area** for unprocessed data.
   - Stores data **exactly as ingested** from various sources, preserving **data fidelity and traceability**.
2. **Cleaned Zone**

- Contains data that has undergone **basic transformations**, including:
    - **Deduplication** to remove redundant entries.
    - **Format standardization** for consistency across datasets.
    - **Schema validation** to ensure structural integrity.
- Prepares data for **further transformation and analysis**.

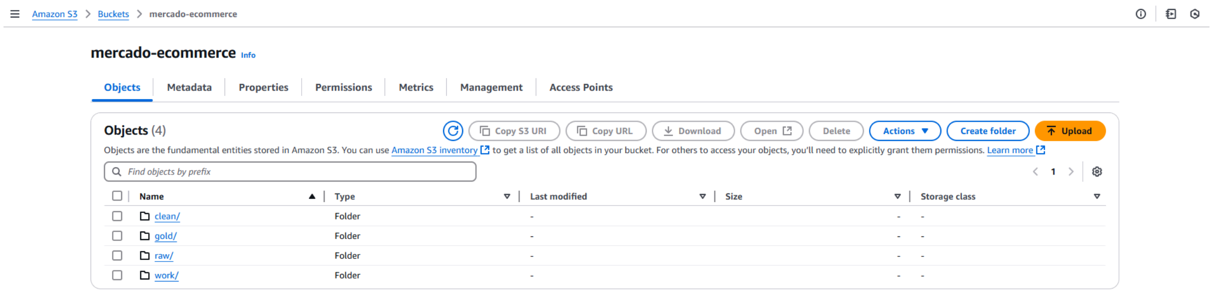
3. **Work Zone**

- Serves as a **sandbox environment** for **analysts and data engineers**.
- Used for **data transformations, enrichment, and exploratory analysis** before finalizing datasets.

4. **Gold Zone**

- Stores **highly curated, business-ready datasets** optimized for:
    - **Analytics and reporting** for decision-making.
    - **Machine learning applications** requiring high data quality.
- Ensures **data governance, compliance, and usability**.

S3 contents:



Amazon S3 > Buckets > mercado-ecommerce

mercado-ecommerce Info

Objects | Metadata | Properties | Permissions | Metrics | Management | Access Points

Objects (4)

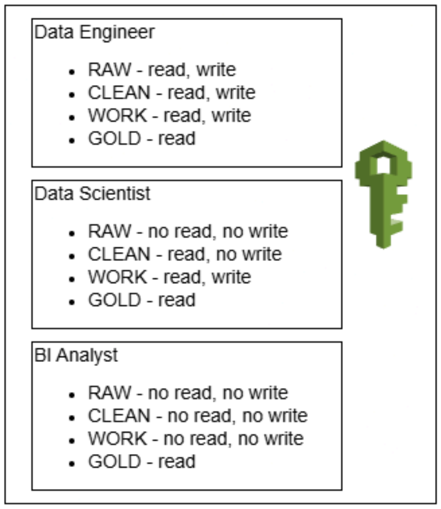
Copy S3 URI | Copy URL | Download | Open | Delete | Actions | Create folder | Upload

Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

Find objects by prefix

Name	Type	Last modified	Size	Storage class
clean/	Folder	-	-	-
gold/	Folder	-	-	-
raw/	Folder	-	-	-
work/	Folder	-	-	-

RBAC - Role-Based Access Management



Data Engineer

- RAW - read, write
- CLEAN - read, write
- WORK - read, write
- GOLD - read

Data Scientist

- RAW - no read, no write
- CLEAN - read, no write
- WORK - read, write
- GOLD - read

BI Analyst

- RAW - no read, no write
- CLEAN - no read, no write
- WORK - no read, no write
- GOLD - read

RBAC

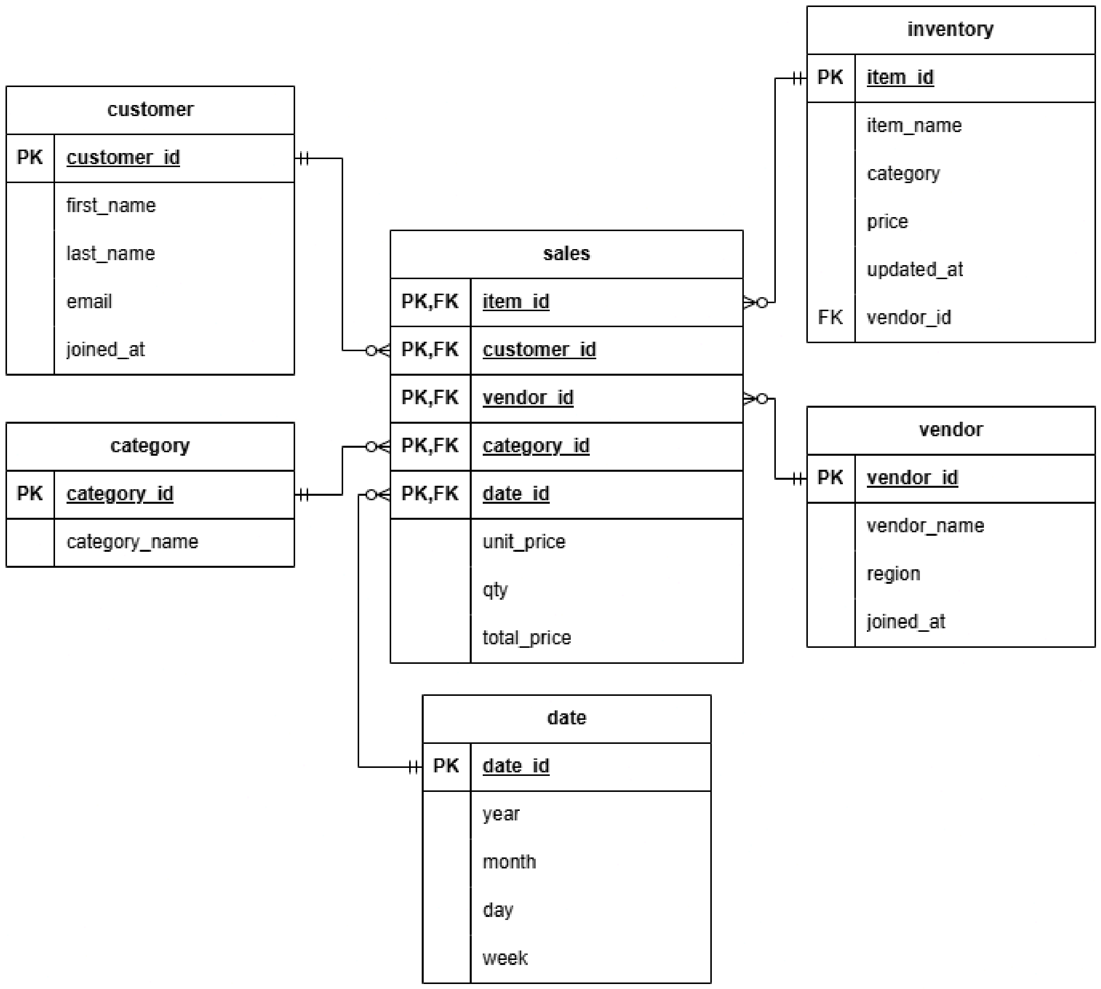
The **Role-Based Access Control (RBAC)** model is designed to enforce **data security, governance, and efficient collaboration** by granting appropriate permissions based on user roles within the data lake.

- **Data Engineers**
    - Have **full access** to ingest, clean, and prepare data.

    - Ensure **data quality and usability** across all zones.
- **Data Scientists**
    - Access **curated datasets** for analysis and modeling.
    - Can experiment in the **work zone** but **cannot modify raw or cleaned data**, preserving data integrity.
- **BI Analysts**
    - Restricted to **business-ready data** in the **gold zone**.
    - Ensures **data consistency in reporting** while preventing accidental modifications to upstream datasets.

This structure minimizes the risk of accidental data corruption while ensuring each role has the appropriate level of access for their tasks.

Data Warehousing - Redshift



The **Entity-Relationship Diagram (ERD)** represents an **e-commerce sales fact table**, with each transaction line item serving as the **granularity** of the dataset.

At the center of the schema is the `sales` **fact table**, which records key transactional data, including:

- `product_id` – Links to product details.
- `customer_id` – Associates sales with specific customers.
- `date_id` – Enables time-based analysis.
- `quantity_sold` – Tracks the number of units sold per transaction.

Supporting Dimension Tables
The fact table is connected to multiple **dimension tables**, providing contextual details for analytics:

- `customer` **dimension**
  - Stores customer-related data, enabling insights into **purchasing behavior and segmentation**.
- `category` **dimension**
  - Classifies products, supporting **sales performance analysis** across different product groups.
- `date` **dimension**
  - Facilitates **time-based reporting**, enabling trend analysis, seasonality insights, and forecasting.
- `inventory` **dimension**
  - Tracks stock levels, aiding in **supply chain efficiency** and inventory optimization.
- `vendor` **dimension**
  - Contains supplier details, helping assess **procurement efficiency and vendor performance**.

Star Schema for Optimized Analytics
This **star schema design** enhances **query performance and scalability**, making it efficient for **business intelligence, reporting, and sales analysis** in an e-commerce environment.

In [14]:
```
connection_string = f"postgresql://{OLAP_USER}:{OLAP_PASS}@{OLAP_HOST}:5439/{OLAP_DB_NAME}"
engine = redshift_connector.connect(
    host=OLAP_HOST,
    port=5439,
    database=OLAP_DB_NAME,
    user=OLAP_USER,
    password=OLAP_PASS
)
engine = create_engine(connection_string)
get_ipython().run_line_magic('sql', connection_string)
```

Connecting and switching to connection 'postgresql://vincent:***@samplecluster.cq68pg38qszs.us-east-1.redshift.amazonaws.com:5439/mercado_ecommerce'

*Create table definition found in `src\sql\*` directory

In [21]:
```
%sql \dt
```

Running query in 'postgresql://vincent:***@samplecluster.cq68pg38qszs.us-east-1.redshift.amazonaws.com:5439/mercado_ecommerce'

Out[21]:

schema	name	type	owner
public	category	table	vincent
public	customer	table	vincent
public	date	table	vincent
public	inventory	table	vincent
public	sales	table	vincent
public	vendor	table	vincent

In [22]:
```
table_list = [
    "category", "customer", "date", "inventory", "vendor", "sales"
]
```

In [23]:
```
for table in table_list:
    print(f"{table}")
    schema_query = f"columns --table {table}"
    display(get_ipython().run_line_magic('sqlcmd', schema_query))
```

category

name	type	nullable	default	autoincrement	comment
category_id	INTEGER	False	None	False	None
category_name	VARCHAR(100)	True	None	False	None

customer

name	type	nullable	default	autoincrement	comment
customer_id	INTEGER	False	None	False	None
first_name	VARCHAR(50)	True	None	False	None
last_name	VARCHAR(50)	True	None	False	None
email	VARCHAR(100)	True	None	False	None
joined_at	TIMESTAMP	True	None	False	None

date

name	type	nullable	default	autoincrement	comment
date_id	INTEGER	False	None	False	None
year	INTEGER	True	None	False	None
month	INTEGER	True	None	False	None
day	INTEGER	True	None	False	None

inventory

name	type	nullable	default	autoincrement	comment
item_id	INTEGER	False	None	False	None
item_name	VARCHAR(100)	True	None	False	None
category	INTEGER	True	None	False	None
price	NUMERIC(10, 2)	True	None	False	None
updated_at	TIMESTAMP	True	None	False	None
vendor_id	INTEGER	True	None	False	None

vendor

name	type	nullable	default	autoincrement	comment
vendor_id	INTEGER	False	None	False	None
vendor_name	VARCHAR(100)	True	None	False	None
region	VARCHAR(100)	True	None	False	None
joined_at	TIMESTAMP	True	None	False	None

sales

name	type	nullable	default	autoincrement	comment
item_id	INTEGER	True	None	False	None
customer_id	INTEGER	True	None	False	None
vendor_id	INTEGER	True	None	False	None
category_id	INTEGER	True	None	False	None
date_id	INTEGER	True	None	False	None
unit_price	NUMERIC(10, 2)	True	None	False	None
qty	INTEGER	True	None	False	None
total_price	NUMERIC(10, 2)	True	None	False	None

In [24]:
```
for table in table_list:
    print(f"{table}")
    display(pd.read_sql(f"select * from {table} limit 100", engine.raw_connection()))
```

```
category
```

	category_id	category_name
0	1	General

```
customer
```

	customer_id	first_name	last_name	email	joined_at
0	1	Brandon	Sims	adrian49@example.com	2025-03-17 02:33:40.378808
1	2	Ronald	Johnson	igarcia@example.org	2025-03-17 02:33:40.388027
2	3	William	Bryant	bobby28@example.org	2025-03-17 02:33:40.396208
3	4	Sarah	Roberts	wlopez@example.org	2025-03-17 02:33:40.403323
4	5	Stacy	Riley	heather29@example.com	2025-03-17 02:33:40.411846
5	6	Maria	Case	michaelrasmussen@example.net	2025-03-17 02:33:40.421316
6	7	Barbara	James	scott38@example.com	2025-03-17 02:33:40.428812
7	8	Joseph	Miller	nicolepreston@example.com	2025-03-17 02:33:40.437615
8	9	Melissa	Thomas	carol19@example.com	2025-03-17 02:33:40.446074
9	10	Roberta	Brown	lambertbryan@example.org	2025-03-17 02:33:40.456996

```
date
```

	date_id	year	month	day
0	20250317	2025	3	17

```
inventory
```

	item_id	item_name	category	price	updated_at	vendor_id
0	1	necessary	1	314.0	2025-03-17 02:33:40.508823	1
1	2	consumer	1	79.0	2025-03-17 02:33:40.517968	1
2	3	begin	1	334.0	2025-03-17 02:33:40.525865	1
...	...	...	...	...	...	...
47	48	section	1	237.0	2025-03-17 02:33:40.912562	5
48	49	popular	1	124.0	2025-03-17 02:33:40.919968	5
49	50	theory	1	227.0	2025-03-17 02:33:40.926696	5

50 rows × 6 columns

```
vendor
```

	vendor_id	vendor_name	region	joined_at
0	1	Moyer-Vance	BZ	2025-03-17 02:33:40.466291
1	2	Todd and Sons	MN	2025-03-17 02:33:40.475478
2	3	Adams PLC	SZ	2025-03-17 02:33:40.482745
3	4	Mendoza Ltd	BZ	2025-03-17 02:33:40.490691
4	5	Williams Inc	PS	2025-03-17 02:33:40.498885

```
sales
```

	item_id	customer_id	vendor_id	category_id	date_id	unit_price	qty	total_price
0	1	6	5	1	20250317	150.0	374	56100.0
1	7	6	3	1	20250317	381.0	180	68580.0
2	4	9	2	1	20250317	306.0	29	8874.0
...	...	...	...	...	...	...	...	...
37	9	4	2	1	20250317	372.0	65	24180.0
38	1	10	1	1	20250317	129.0	53	6837.0
39	2	10	4	1	20250317	290.0	428	124120.0

40 rows × 8 columns

Consumption Layer

Below we explore some options in consuming the data on our created Data Lake and Warehoure

Read from Data Lake

We can read files from the data lake in this very notebook

```
In [25]: pd.read_csv("s3://mercado-ecommerce/raw/sales/20250317/sales.csv").head()
```

Out[25]:

	item_id	customer_id	vendor_id	category_id	date_id	unit_price	qty	total_price
0	1	6	5	1	20250317	150.0	374.0	56100.0
1	7	6	3	1	20250317	381.0	180.0	68580.0
2	4	9	2	1	20250317	306.0	29.0	8874.0
3	2	9	3	1	20250317	52.0	21.0	1092.0
4	3	5	4	1	20250317	247.0	291.0	71877.0

Read from Data Warehouse

We can also consume the data warehouse tables

```
In [27]: %%sql
SELECT
    i.item_name,
    c.first_name,
    c.last_name,
    v.vendor_name,
    cat.category_name,
    d.year,
    d.month,
    d.day,
    s.unit_price,
    s.qty,
    s.total_price
FROM sales s
INNER JOIN inventory i
    ON i.item_id = s.item_id
INNER JOIN customer c
    ON c.customer_id = s.customer_id
INNER JOIN vendor v
    ON v.vendor_id = s.vendor_id
INNER JOIN category cat
    ON cat.category_id = s.category_id
```

```
INNER JOIN date d
    ON d.date_id = s.date_id
```

```
Running query in 'postgresql://vincent:***@samplecluster.cq68pg38qszs.us-east-1.redshift.amazonaws.com:5439/mercado_ecommerce'
40 rows affected.
```

Out[27]:

item_name	first_name	last_name	vendor_name	category_name	year	month	day	unit_price	qty	total_pri
necessary	Maria	Case	Williams Inc	General	2025	3	17	150.00	374	56100.
board	Maria	Case	Adams PLC	General	2025	3	17	381.00	180	68580.
already	Melissa	Thomas	Todd and Sons	General	2025	3	17	306.00	29	8874.
consumer	Melissa	Thomas	Adams PLC	General	2025	3	17	52.00	21	1092.
begin	Stacy	Riley	Mendoza Ltd	General	2025	3	17	247.00	291	71877.
window	Stacy	Riley	Adams PLC	General	2025	3	17	98.00	363	35574.
service	Brandon	Sims	Adams PLC	General	2025	3	17	12.00	182	2184.
charge	Brandon	Sims	Moyer-Vance	General	2025	3	17	392.00	162	63504.
board	Melissa	Thomas	Adams PLC	General	2025	3	17	413.00	340	140420.
agent	Melissa	Thomas	Williams Inc	General	2025	3	17	392.00	310	121520.

Truncated to displaylimit of 10.

Summary: End-to-End E-Commerce System Implementation

This implementation of **Samn's Mercado** showcases a complete **end-to-end e-commerce architecture**, covering essential components:

- **API Layer** – Handles communication between frontend, backend, and external services.
- **OLTP Layer (SQL & NoSQL)** – Manages transactional data using **PostgreSQL** and **DynamoDB** for structured and flexible storage.
- **Workflow Management & Data Pipelines** – Automates data extraction, transformation, and loading (ETL) using **Apache Airflow**.
- **Data Lake Zones & Role-Based Access Control (RBAC)** – Organizes data into **raw, cleaned, work, and gold zones** while enforcing security and governance.
- **Data Warehousing & Dimensional Modeling** – Structures data efficiently in **Amazon Redshift**, enabling analytics and business intelligence.

Scalability, Flexibility, and Business Insights
By integrating **Docker, AWS services, and Apache Airflow**, **Samn's Mercado** ensures:
✅ **Scalability** – Adapts to growing business demands.
✅ **Flexibility** – Supports a hybrid SQL & NoSQL architecture.
✅ **Security** – Implements robust access controls and governance policies.

With this architecture, the platform **empowers vendors** with **advanced analytics and data-driven decision-making**, optimizing business operations for **enhanced performance and growth**.