REVATURE PROJECT 0 - DETAILED PROCESS OF ETL FOR THE GIVEN DATASETS

BY

ARAVINDAN.B

Introduction:

This documentation describes the ETL (Extract, Transform, Load) process implemented in Python for managing and processing data related to an e-commerce platform. The ETL process involves extracting data from JSON files, transforming it to fit the required format, and loading it into a MySQL database. The code provided handles the entire workflow from data extraction to data viewing.

Overview

The ETL script performs the following tasks:

Data Extraction:

Purpose: To read and load data from JSON files into Python.

Files:

1. customer.json: Contains customer details such as ID, name, country, and city.
2. transaction_logs.json: Contains transaction records, including product information, order details, and payment transactions.

Data Transformation:

Purpose: To clean and convert the extracted data into a format suitable for database insertion.

Operations:

1. Convert price values to floating-point numbers.
2. Ensure data consistency and readiness for database operations.

Data Loading:

Purpose: To insert the transformed data into a MySQL database.

Database Schema:

1. **Customers**: Stores customer information.
2. **Products**: Stores product details.
3. **Orders**: Contains order information, including customer and product references.
4. **Payments**: Manages payment transactions linked to orders.

Operations:

1. Create database tables if they do not exist.
2. Insert or update customer, product, order, and payment records.

Data Viewing:

Purpose: To query and display the contents of the database tables for verification and analysis.

Operations:

Fetch and print records from each table: Customers, Products, Orders, and Payments.

Functionality

The script is structured into four main functions:

- extract_data(customer_file, transactions_file): Reads data from the specified JSON files.
- transform_data(customers_data, transactions_data): Transforms the data for consistency and correctness.
- load_data(customers_data, transactions_data): Loads the transformed data into the MySQL database, managing table creation and data insertion.
- view_tables(): Queries and displays the data stored in the MySQL database tables.

This ETL process ensures that data from various sources is efficiently managed and maintained in a structured database, facilitating reliable data access and reporting for e-commerce operations.

Code for the Data Extraction, Transformation and Load:

```
import json
import mysql.connector
from mysql.connector import Error

# Function to extract data from JSON files
def extract_data(customer_file, transactions_file):
    print("Extracting data from JSON files...")
    with open(customer_file, 'r') as file:
        customers_data = json.load(file)
    with open(transactions_file, 'r') as file:
        transactions_data = json.load(file)
    print("Data extraction completed.")
    return customers_data, transactions_data

# Function to transform data if needed
def transform_data(customers_data, transactions_data):
    print("Transforming data...")
    # Example transformation: converting price from integer to float
    for transaction in transactions_data:
        transaction['price'] = float(transaction['price'])
    print("Data transformation completed.")
    return customers_data, transactions_data

# Function to load data into MySQL with normalization
def load_data(customers_data, transactions_data):
    connection = None
    try:
        print("Connecting to the MySQL database...")
        # Connect to MySQL database
        connection = mysql.connector.connect(
            host='localhost',
            database='ecommerce_db',
            user='your_username',
            password='your_password'
        )

        if connection.is_connected():
            print("Connection established.")
            cursor = connection.cursor()

            # Create normalized tables if not exists
            print("Creating tables if they do not exist...")
            cursor.execute('''CREATE TABLE IF NOT EXISTS customers (
                                customer_id INT PRIMARY KEY,
                                customer_name VARCHAR(255),
                                country VARCHAR(255),
                                city VARCHAR(255)
                              )''')

            cursor.execute('''CREATE TABLE IF NOT EXISTS products (
                                product_id INT PRIMARY KEY,
                                product_name VARCHAR(255),
                                product_category VARCHAR(255)
                              )''')

            cursor.execute('''CREATE TABLE IF NOT EXISTS orders (
                                order_id INT PRIMARY KEY,
                                customer_id INT,
                                product_id INT,
                                datetime DATETIME,
                                qty INT,
                                price FLOAT,
                                ecommerce_website_name VARCHAR(255),
                                FOREIGN KEY (customer_id) REFERENCES customers(customer_id),
                                FOREIGN KEY (product_id) REFERENCES products(product_id)
                              )''')

            cursor.execute('''CREATE TABLE IF NOT EXISTS payments (
                                payment_txn_id VARCHAR(255) PRIMARY KEY,
                                order_id INT,
                                payment_type VARCHAR(50),
                                payment_txn_success CHAR(1),
                    failure_reason VARCHAR(255),
                    FOREIGN KEY (order_id) REFERENCES orders(order_id)
                  )''')
            print("Tables created or confirmed.")

            # Load customers data
            print("Inserting customer data...")
            for customer in customers_data:
                cursor.execute('''INSERT INTO customers (customer_id, customer_name, country, city)
                      VALUES (%s, %s, %s, %s)
                      ON DUPLICATE KEY UPDATE
                      customer_name=VALUES(customer_name),
                      country=VALUES(country),
                      city=VALUES(city)''',
                   (customer['customer_id'], customer['customer_name'],
                    customer['country'], customer['city']))
            print("Customer data inserted.")

            # Load transactions data
            print("Inserting transactions data...")
            for transaction in transactions_data:
                # Insert product data
                cursor.execute('''INSERT INTO products (product_id, product_name, product_category)
                      VALUES (%s, %s, %s)
                      ON DUPLICATE KEY UPDATE
                      product_name=VALUES(product_name),
                      product_category=VALUES(product_category)''',
                   (transaction['product_id'], transaction['product_name'],
                    transaction['product_category']))

                # Insert order data
                cursor.execute('''INSERT INTO orders (order_id, customer_id, product_id, datetime, qty,
price, ecommerce_website_name)
                      VALUES (%s, %s, %s, %s, %s, %s, %s)
                      ON DUPLICATE KEY UPDATE
                      datetime=VALUES(datetime),
                      qty=VALUES(qty),
                      price=VALUES(price),
                      ecommerce_website_name=VALUES(ecommerce_website_name)''',
                   (transaction['order_id'], transaction['customer_id'],
                    transaction['product_id'], transaction['datetime'],
                    transaction['qty'], transaction['price'],
                    transaction['ecommerce_website_name']))

                # Insert payment data
                cursor.execute('''INSERT INTO payments (payment_txn_id, order_id, payment_type,
payment_txn_success, failure_reason)
                      VALUES (%s, %s, %s, %s, %s)
                      ON DUPLICATE KEY UPDATE
                      payment_type=VALUES(payment_type),
                      payment_txn_success=VALUES(payment_txn_success),
                      failure_reason=VALUES(failure_reason)''',
                   (transaction['payment_txn_id'], transaction['order_id'],
                    transaction['payment_type'], transaction['payment_txn_success'],
                    transaction['failure_reason']))

            connection.commit()
            print("Transactions data inserted.")
            print("Data loading completed.")

    except Error as e:
        print(f"Error: {e}")
    finally:
        if connection and connection.is_connected():
            cursor.close()
            connection.close()
            print("Connection closed.")

if __name__ == '__main__':
    # File paths for JSON files
    customer_file = 'customer.json'
    transactions_file = 'transaction_logs.json'

    # ETL Process
    customers_data, transactions_data = extract_data(customer_file, transactions_file)
    customers_data, transactions_data = transform_data(customers_data, transactions_data)
    load_data(customers_data, transactions_data)
```

ETL Architecture Overview

Detailed Architecture

1. Data Extraction

Component: File Reader

Input: JSON Files

Function: Extract data from customer.json and transaction_logs.json.

Code: def extract_data(customer_file, transactions_file):

2. Data Transformation

Component: Data Transformer

Input: Raw JSON data (customers and transactions)

Operation: Transform data for consistency and prepare it for loading.

Code: def transform_data(customers_data, transactions_data):

3. Data Loading

Component: Data Loader

Input: Transformed data

Operation: Load data into MySQL database.

Sub-components:

Database Connection Manager: Manages database connections.

Table Creator: Creates tables if they do not exist.

Data Inserter: Inserts or updates data into tables.

Code: def load_data(customers_data, transactions_data):

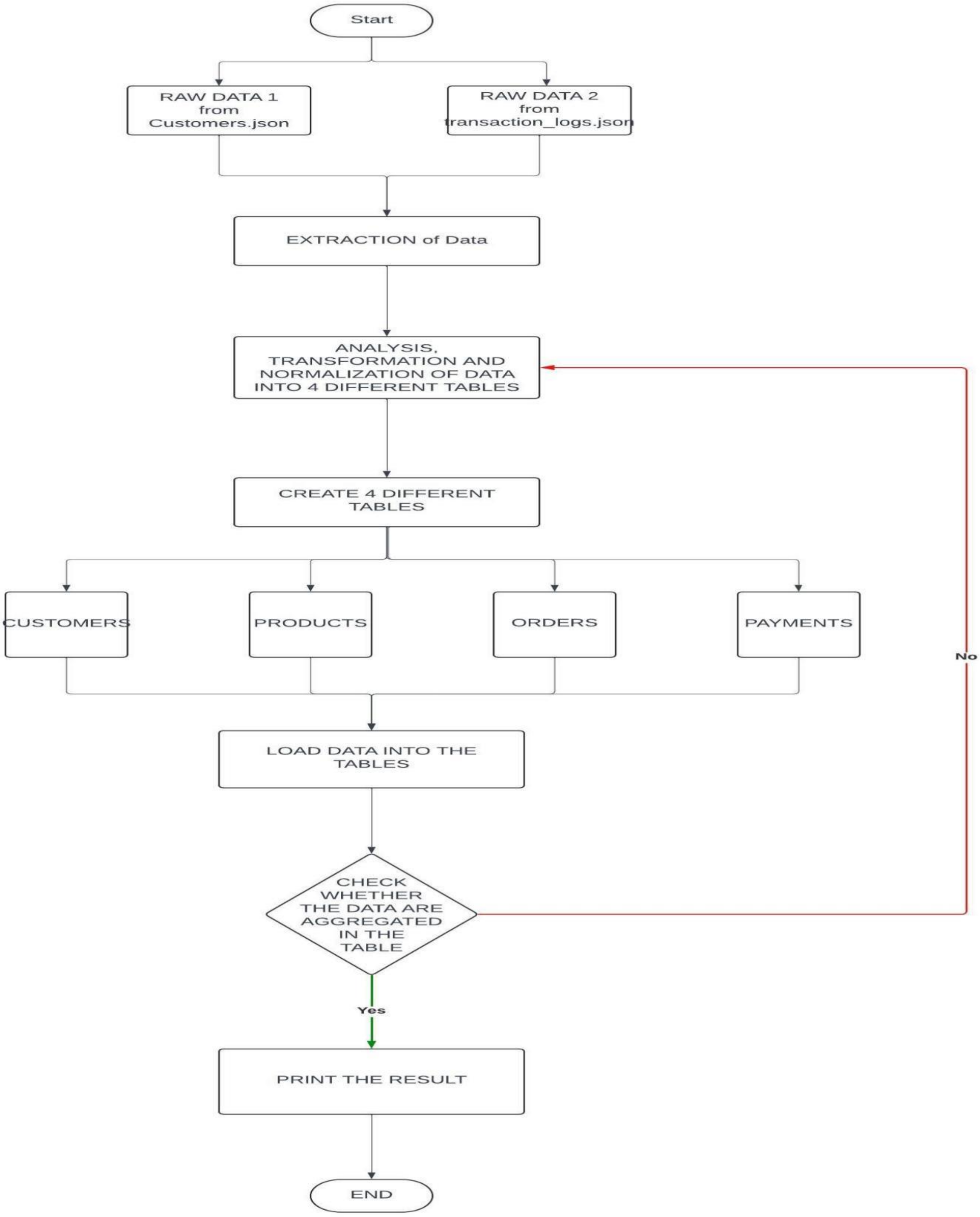
4. Data Viewing

Component: Data Viewer

Operation: Fetch and display data from MySQL tables.

Code: def view_tables():

ARCHITECHTURAL DIAGRAM:


Start
RAW DATA 1 from Customers.json
RAW DATA 2 from transaction_logs.json
EXTRACTION of Data
ANALYSIS, TRANSFORMATION AND NORMALIZATION OF DATA INTO 4 DIFFERENT TABLES
CREATE 4 DIFFERENT TABLES
CUSTOMERS
PRODUCTS
ORDERS
PAYMENTS
No
LOAD DATA INTO THE TABLES
CHECK WHETHER THE DATA ARE AGGREGATED IN THE TABLE
Yes
PRINT THE RESULT
END

Schema

1. Customers Table

Table Name: customers

Description: Stores customer information.

Column Name	Data Type	Constraints	Description
Customer_id	INT	PRIMARY KEY	Unique identifier for each customer.
Customer_name	VARCHAR(255)	NOT NULL	Name of the customer.
country	VARCHAR(255)		Country where the customer resides.
city	VARCHAR(255)		City where the customer resides.

2. Products Table

Table Name: products

Description: Stores product information.

Column Name	Data Type	Constraints	Description
Product_id	INT	PRIMARY KEY	Unique identifier of the Product
Product_name	VARCHAR(255)	NOT NULL	Name of the Product
Product_category	VARCHAR(255)		Category of the Product.

3. Orders Table

Table Name: orders

Description: Stores order details.

Column Name	Data Type	Constraints	Description
Order_id	INT	PRIMARY KEY	Unique Identifier of the order.
Customer_id	INT	FOREIGN KEY REFERENCES customers(customer_id)	Reference to the customer who made the order.
Product_id	INT	FOREIGN KEY REFERENCES products(product_id)	Reference to the product in the order.
datetime	DATETIME	NOT NULL	Date and time when the order was placed.
qty	INT	NOT NULL	Quantity of the product ordered.
price	FLOAT	NOT NULL	Price of the product in the order.
Ecommerce_website_name	VARCHAR(255)		Name of the e-commerce website where the order was placed.

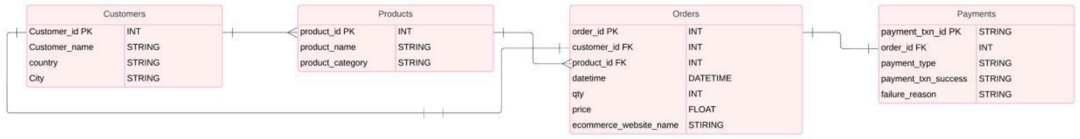
4. Payments Table

Table Name: payments

Description: Stores payment information.

Column Name	Data Type	Constraints	Description
payment_txn_id	VARCHAR(255)	PRIMARY KEY	Unique identifier for the payment transaction.
order_id	INT	FOREIGN KEY REFERENCES orders(order_id)	Reference to the associated order.
payment_type	VARCHAR(50)	NOT NULL	Type of payment (e.g., credit card, PayPal).
payment_txn_success	CHAR(1)	NOT NULL	Indicates if the payment transaction was successful (Y or N).
failure_reason	VARCHAR(255)		Reason for payment failure if applicable.

ENTITY RELATIONSHIP DIAGRAM:



Relationships:

Customers to Orders: One-to-Many (One customer can have many orders).

Products to Orders: One-to-Many (One product can be part of many orders).

Orders to Payments: One-to-One (Each order has one payment record).

PROBLEMS/CHALLENGES FACED:

1. Data Transformation Complexity

Problem: Transforming data, especially when the transformations are complex or require aggregations and calculations, can be challenging.

Solution: I Clearly defined the transformation rules and logic.I tested transformations on a small subset of data before applying them to the entire data set.

2. Handling Schema Changes

Problem: Changes in the source schema can disrupt the ETL process.

Solution: I designed the ETL processes to be adaptable to schema changes and Implemented versioning and backward compatibility checks.

3. Error Handling and Logging

Problem: Identifying and resolving errors during extraction, transformation, or loading can be challenging without proper logging and error-handling mechanisms.

Solution: I implemented the error handling method to overcome this process and deliver a smoother content.

Conclusion:

Key things to be considered for the ETL process are:

- **Data Quality**: Data validation and cleaning are crucial to ensure that the data loaded into the database is accurate and consistent.
- **Schema Design**: A well-designed schema is essential for efficient data storage and retrieval, as well as for maintaining data integrity and reducing redundancy.
- **Performance**: Handling large volumes of data requires careful attention to performance optimization, including indexing and batch processing techniques.
- **Error Handling**: Comprehensive logging and error-handling mechanisms are implemented to manage and resolve issues that arise during the ETL process.
- **Security and Compliance**: Data security and compliance with regulations are prioritized to protect sensitive information and meet legal requirements.