Data Engineer Take-Home Challenge: ETL Pipeline Development

Objective:
Design and implement an ETL pipeline that extracts data from multiple CSV files, applies transformations based on business rules, and loads the data into a PostgreSQL database. The pipeline should also include robust error handling and logging.

You should create a private GitHub repository and share it with the GitHub usernames (RahmatYousufi and andyphamberkeley) upon completion. There is a 72-hour timeline from the time you receive this assignment. We estimate it will take 4-6 hours to complete.

Challenge Overview

You will be provided with multiple CSV files containing sales and customer data. Your task is to build a dynamic ETL pipeline that processes this data and handles various potential issues along the way.

Sample Data

sales_data.csv

order_id	customer_id	product	quantity	price	order_date
1	1001	Widget A	2	19.99	2024-01-15 10:00:00
2	1002	Widget B	1	29.99	2024-01-16 11:00:00
3	1003	Widget A	1	19.99	2024-01-16 12:00:00

4	1004	Widget C	0	39.99	2024-01-17 14:00:00
5	1005	Widget B	3	29.99	invalid_date

customer_data.csv

customer_id	customer_name	email	signup_date
1001	John Doe	john.doe@example.com	2023-06-10 09:00:00
1002	Jane Smith	jane.smith@example.com	invalid_date
1003	Alice Johnson	alice.j@example.com	2022-12-01 14:00:00
1004	Bob Brown	bob.brown@example.com	2023-01-17 10:00:00
1005	Charlie White	charlie.w@example.com	2023-11-15 11:00:00

You can create these files for the challenge.

Requirements

Extraction

1. **File Reading:**

    - Read multiple CSV files (`sales_data.csv` and `customer_data.csv`) and load them into suitable data structures (e.g., Pandas DataFrames).
    - Dynamically detect and handle missing files.
2. **Validation:**
    - Validate that the required columns exist in each file. Log errors if columns are missing.

Transformation

1. **Data Cleaning:**
    - Convert `order_date` and `signup_date` to a standard format (YYYY-MM-DD). Log errors for invalid dates and skip affected records.
    - Remove any rows with missing or invalid `customer_id` or `order_id`.
2. **Data Enrichment:**
    - Calculate the total value for each order (`total_value = quantity * price`).
    - Join the sales data with customer data on `customer_id` to enrich the sales table with `customer_name` and `email`.
3. **Business Rules:**
    - Exclude orders with a `quantity` of 0 or less.
    - Mark orders with total values exceeding $1000 as "High-Value Orders" in a new column `order_type`.
    - Add a column `customer_tenure` representing the number of days since the customer's signup date.
4. **Data Aggregation:**
    - Create a summary table with total sales (`SUM(total_value)`) per product and order counts (`COUNT(order_id)`).

Loading

1. **Database Schema:**
    - Create the following tables in PostgreSQL:
        - **sales**: Contains all transformed sales data with fields: `order_id`, `customer_id`, `product`, `quantity`, `price`, `order_date`, `total_value`, `customer_name`, `email`, `order_type`, `customer_tenure`.
        - **sales_summary**: Aggregates product-level metrics with fields: `product`, `total_sales`, `order_count`.
2. **Constraints:**
    - Enforce primary and foreign key constraints (e.g., `customer_id` in `sales` references `customer_data`).
3. **Data Ingestion:**

    - Batch load data into the database. Implement transaction management to ensure that partial failures do not corrupt data.
4. **Logging:**
    - Log the number of records processed, records loaded, and any errors encountered.

Advanced Requirements

1. **Performance Optimization:**
    - Optimize the pipeline for handling large files (e.g., processing data in chunks).
    - Use indexes in the PostgreSQL database for faster queries.
2. **Scalability:**
    - Design the ETL to handle additional files and tables in the future by parameterizing the pipeline.
3. **Configuration Management:**
    - Store configurable parameters (e.g., file paths, database connection strings) in a `config.json` file or environment variables.

Deliverables

1. **Source Code:**
    - Include clear and well-organized Python scripts or notebooks for the ETL pipeline.
2. **Database Script:**
    - SQL script to create the required PostgreSQL tables with constraints.
3. **README:**
    - Explain the approach, any challenges faced, and how to:
        - Set up the database.
        - Run the ETL pipeline.
4. **Sample CSV Files:**
    - Provide `sales_data.csv` and `customer_data.csv`.

Evaluation Criteria

1. **Code Quality:**
    - Modularity, readability, and adherence to best practices.
2. **Error Handling:**
    - Robust handling of edge cases and invalid data.
3. **Performance:**

    - Ability to handle large datasets efficiently.
4. **Scalability:**
    - Flexibility of the pipeline to adapt to future requirements.
5. **Documentation:**
    - Completeness and clarity of instructions.

Deliverables

1. **Code**: Your implementation and database migration files.
2. **Documentation**: A README file with setup instructions.
3. **Git Workflow**: Ensure your commits reflect logical progress, and commit as you implement.