ETL Pipeline for Automated Excel File Processing

Install necessary packages:

In [1]:
```
pip install pandas sqlalchemy pymysql watchdog openpyxl
```

```
Requirement already satisfied: pandas in c:\users\dell\anaconda3\lib\site-packages (2.2.2)
Requirement already satisfied: sqlalchemy in c:\users\dell\anaconda3\lib\site-packages (2.0.30)
Requirement already satisfied: pymysql in c:\users\dell\anaconda3\lib\site-packages (1.1.1)
Requirement already satisfied: watchdog in c:\users\dell\anaconda3\lib\site-packages (4.0.1)
Requirement already satisfied: openpyxl in c:\users\dell\anaconda3\lib\site-packages (3.1.2)
Requirement already satisfied: numpy>=1.26.0 in c:\users\dell\anaconda3\lib\site-packages (from pandas) (1.26.4)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\dell\anaconda3\lib\site-packages (from pandas) (2.
9.0.post0)
Requirement already satisfied: pytz>=2020.1 in c:\users\dell\anaconda3\lib\site-packages (from pandas) (2024.1)
Requirement already satisfied: tzdata>=2022.7 in c:\users\dell\anaconda3\lib\site-packages (from pandas) (2023.3)
Requirement already satisfied: typing-extensions>=4.6.0 in c:\users\dell\anaconda3\lib\site-packages (from sqlalchem
y) (4.12.2)
Requirement already satisfied: greenlet!=0.4.17 in c:\users\dell\anaconda3\lib\site-packages (from sqlalchemy) (3.0.
1)
Requirement already satisfied: et-xmlfile in c:\users\dell\anaconda3\lib\site-packages (from openpyxl) (1.1.0)
Requirement already satisfied: six>=1.5 in c:\users\dell\anaconda3\lib\site-packages (from python-dateutil>=2.8.2->pa
ndas) (1.16.0)
Note: you may need to restart the kernel to use updated packages.
```

Automated ETL Script

In [2]:
```python
import pandas as pd
import os
from sqlalchemy import create_engine
import pymysql  # Ensure pymysql is imported
from watchdog.observers import Observer
from watchdog.events import FileSystemEventHandler
import time
```

In [3]:
```python

# --- Configuration ---
watch_folder = r"C:\Users\DELL\Documents\Pipeline_Automation"  # folder to watch for new Excel files
db_config = {
    "username": "root",
    "password": "CHINyere9$",  # Ensure this is correct
    "host": "127.0.0.1",
    "port": 3306,
    "database": "order_sales_data",

}
table_name = "order_data"
```

In [4]:
```
# --- Database Connection ---
def get_engine():
    try:
        db_url = f"mysql+pymysql://{db_config['username']}:{db_config['password']}@{db_config['host']}:{db_config['pc
        engine = create_engine(db_url)
        print(f"✅ Database engine created successfully: {engine}")
        return engine
    except Exception as e:
        print(f"❌ Error creating database engine: {e}")
        return None
```

In [5]:
```
# --- Data Transformation ---

def process_file(file_path):
    try:
        print(f"📂 Processing file: {file_path}")
        df = pd.read_excel(file_path)

        # Ensure 'Customer_ID' is treated as a string
        df['Customer_ID'] = df['Customer_ID'].astype(str)
        df[['Customer_ID', 'Customer_Name']] = df['Customer_ID'].str.split(' - ', expand=True)

        df = df[df["Customer_Name"] != 'Promo']
        df['Cookies_Shipped'] = df['Cookies_Shipped'].replace('[\\$,]', '', regex=True).astype(float)

        engine = get_engine()
        if engine is None:
            print("❌ Database connection failed. Skipping file processing.")
            return

        # Fetch existing records and ensure type consistency
        existing_df = pd.read_sql(f"SELECT Customer_ID, Order_ID FROM {table_name}", con=engine)
        # **Ensure Customer_ID is string in both DataFrames**
        existing_df['Customer_ID'] = existing_df['Customer_ID'].astype(str)

        # Debugging: Print data types before merging
        print(f"🔎 Data types in database:\n{existing_df.dtypes}")
        print(f"🔎 Data types in new DataFrame:\n{df.dtypes}")

        # Prevent duplicates before inserting
        df = df.merge(existing_df, on=['Customer_ID', 'Order_ID'], how='left', indicator=True)
        df = df[df['_merge'] == 'left_only'].drop(columns=['_merge'])  # Keep only new records

        print(f"✅ Final DataFrame shape before inserting: {df.shape}")

        # Load into MySQL
        df.to_sql(table_name, con=engine, if_exists="append", index=False)
        print("✅ ETL completed and data loaded into MySQL without duplicates.")

    except Exception as e:
        print(f"❌ Error processing file: {e}")
```

In [6]:
```
# --- Watcher Class ---
class ExcelHandler(FileSystemEventHandler):
    def on_created(self, event):
        if event.is_directory:
            return
        if event.src_path.endswith(".xlsx"):
            print(f"📁 New file detected: {event.src_path}")  # Debugging output
            process_file(event.src_path)
```

In [ ]:
```
# --- Start Observer ---
if __name__ == "__main__":
    event_handler = ExcelHandler()
    observer = Observer()
    observer.schedule(event_handler, path=watch_folder, recursive=False)
    observer.start()
    print(f"👀 Watching folder: {watch_folder} for new Excel files...")

    try:
        while True:
            time.sleep(1)
    except KeyboardInterrupt:
        observer.stop()
    observer.join()
```

```
👀 Watching folder: C:\Users\DELL\Documents\Pipeline_Automation for new Excel files...
📁 New file detected: C:\Users\DELL\Documents\Pipeline_Automation\2020 Order Data.xlsx
📁 Processing file: C:\Users\DELL\Documents\Pipeline_Automation\2020 Order Data.xlsx
✅ Database engine created successfully: Engine(mysql+pymysql://root:***@127.0.0.1:3306/order_sales_data)
🔎 Data types in database:
Customer_ID    object
Order_ID        int64
dtype: object
🔎 Data types in new DataFrame:
Order_ID                    object
Customer_ID                 object
Cookies_Shipped            float64
Revenue                      int64
Cost                       float64
Orde_Date           datetime64[ns]
Ship_Date           datetime64[ns]
Order_Status                object
Customer_Name               object
dtype: object
✅ Final DataFrame shape before inserting: (0, 9)
✅ ETL completed and data loaded into MySQL without duplicates.
```

In [ ]: