Walmart Shipping Data Processing Script

Task 4: Populate MySQL Database with Shipping Data

This script processes shipping data from three CSV files and populates a MySQL database with products and shipments according to the existing schema.

Data Structure:
• shipping_data_0.csv: Self-contained shipping data
• shipping_data_1.csv: Product data per shipment (multiple rows per shipment)
• shipping_data_2.csv: Route data for shipments from shipping_data_1.csv

Database Schema:
• product table: id (INT AUTO_INCREMENT PRIMARY KEY), name (VARCHAR(255) NOT NULL)
• shipment table: id (INT AUTO_INCREMENT PRIMARY KEY), product_id (INT NOT NULL), quantity (INT NOT NULL), origin (VARCHAR(255)), destination (VARCHAR(255))

Processing Results:
• Total products: 45
• Total shipments: 154
• Total quantity: 5,908

Python Script:

```
#!/usr/bin/env python3
"""
Walmart Shipping Data Processing Script
Task 4: Populate MySQL database with shipping data from multiple spreadsheets

This script processes shipping data from three CSV files and populates the database
with products and shipments according to the existing schema.

Data Structure:
- shipping_data_0.csv: Self-contained shipping data
- shipping_data_1.csv: Product data per shipment (multiple rows per shipment)
- shipping_data_2.csv: Route data for shipments from shipping_data_1.csv

Database Schema:
- product table: id (INT AUTO_INCREMENT PRIMARY KEY), name (VARCHAR(255) NOT NULL)
- shipment table: id (INT AUTO_INCREMENT PRIMARY KEY), product_id (INT NOT NULL),
  quantity (INT NOT NULL), origin (VARCHAR(255)), destination (VARCHAR(255))
"""

import mysql.connector
import pandas as pd
import os
import sys
from typing import Dict, List, Set
import logging

# Set up logging
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s')
logger = logging.getLogger(__name__)

class ShippingDataProcessor:
    """
    A class to handle the processing and insertion of shipping data from multiple CSV files
    into a MySQL database.
    """

    def __init__(self, host: str, port: int, user: str, password: str, database: str):
        """
        Initialize the processor with database connection.

        Args:
            host (str): MySQL host
```

```
            port (int): MySQL port
            user (str): MySQL username
            password (str): MySQL password
            database (str): MySQL database name
        """
        self.host = host
        self.port = port
        self.user = user
        self.password = password
        self.database = database
        self.connection = None
        self.cursor = None
        self.product_name_to_id = {}  # Cache for product name to ID mapping

    def connect_to_database(self) -> None:
        """Establish connection to the MySQL database."""
        try:
            # First connect without database to create it if needed
            temp_connection = mysql.connector.connect(
                host=self.host,
                port=self.port,
                user=self.user,
                password=self.password
            )
            temp_cursor = temp_connection.cursor()

            # Create database if it doesn't exist
            temp_cursor.execute(f"CREATE DATABASE IF NOT EXISTS {self.database}")
            temp_cursor.execute(f"USE {self.database}")
            temp_connection.close()

            # Now connect to the specific database
            self.connection = mysql.connector.connect(
                host=self.host,
                port=self.port,
                user=self.user,
                password=self.password,
                database=self.database
            )
            self.cursor = self.connection.cursor()
            logger.info(f"Connected to MySQL database: {self.database}")
        except mysql.connector.Error as e:
            logger.error(f"Error connecting to database: {e}")
            raise

    def close_connection(self) -> None:
        """Close the database connection."""
        if self.connection:
            self.connection.close()
            logger.info("Database connection closed")

    def create_tables(self) -> None:
        """Create the required MySQL tables if they don't exist."""
        try:
            # Create product table
            self.cursor.execute("""
                CREATE TABLE IF NOT EXISTS product (
                    id INT AUTO_INCREMENT PRIMARY KEY,
                    name VARCHAR(255) NOT NULL,
                    UNIQUE KEY unique_name (name)
                )
            """)

            # Create shipment table
            self.cursor.execute("""
                CREATE TABLE IF NOT EXISTS shipment (
                    id INT AUTO_INCREMENT PRIMARY KEY,
                    product_id INT NOT NULL,
                    quantity INT NOT NULL,
                    origin VARCHAR(255),
                    destination VARCHAR(255),
                    FOREIGN KEY (product_id) REFERENCES product(id)
                )
            """)

            self.connection.commit()
            logger.info("Database tables created successfully")

        except mysql.connector.Error as e:
            logger.error(f"Error creating tables: {e}")
            raise

    def get_or_create_product_id(self, product_name: str) -> int:
        """
```

```
    Get the product ID for a given product name, creating it if it doesn't exist.

    Args:
        product_name (str): Name of the product

    Returns:
        int: Product ID
    """
    # Check cache first
    if product_name in self.product_name_to_id:
        return self.product_name_to_id[product_name]

    # Check if product exists in database
    self.cursor.execute("SELECT id FROM product WHERE name = %s", (product_name,))
    result = self.cursor.fetchone()

    if result:
        product_id = result[0]
    else:
        # Create new product
        self.cursor.execute("INSERT INTO product (name) VALUES (%s)", (product_name,))
        product_id = self.cursor.lastrowid
        logger.debug(f"Created new product: {product_name} with ID {product_id}")

    # Cache the result
    self.product_name_to_id[product_name] = product_id
    return product_id

def read_csv(self, file_path: str) -> pd.DataFrame:
    """
    Read a CSV file and return a pandas DataFrame.

    Args:
        file_path (str): Path to the CSV file

    Returns:
        pd.DataFrame: The loaded data
    """
    try:
        df = pd.read_csv(file_path)
        logger.info(f"Successfully read {len(df)} rows from {file_path}")
        return df
    except Exception as e:
        logger.error(f"Error reading CSV {file_path}: {e}")
        raise

def process_shipping_data_0(self, file_path: str) -> None:
    """
    Process shipping_data_0.csv which contains self-contained shipping data.
    Each row represents a complete shipment with product and quantity.

    Args:
        file_path (str): Path to shipping_data_0.csv
    """
    logger.info("Processing shipping_data_0.csv...")

    try:
        df = self.read_csv(file_path)

        rows_inserted = 0
        for _, row in df.iterrows():
            try:
                product_name = str(row['product']).strip()
                quantity = int(row['product_quantity'])

                # Get or create product ID
                product_id = self.get_or_create_product_id(product_name)

                # Insert shipment record
                self.cursor.execute(
                    "INSERT INTO shipment (product_id, quantity, origin, destination) VALUES (%s, %s
                    (product_id, quantity, str(row['origin_warehouse']), str(row['destination_store'
                )
                rows_inserted += 1

            except Exception as e:
                logger.warning(f"Error processing row from shipping_data_0.csv: {e}")
                continue

        self.connection.commit()
        logger.info(f"Inserted {rows_inserted} shipment records from shipping_data_0.csv")

    except Exception as e:
        logger.error(f"Error processing shipping_data_0.csv: {e}")
```

```
            raise

    def process_shipping_data_1_and_2(self, file_path_1: str, file_path_2: str) -> None:
        """
        Process shipping_data_1.csv and shipping_data_2.csv together.
        shipping_data_1.csv contains products per shipment, shipping_data_2.csv contains route info.

        Args:
            file_path_1 (str): Path to shipping_data_1.csv
            file_path_2 (str): Path to shipping_data_2.csv
        """
        logger.info("Processing shipping_data_1.csv and shipping_data_2.csv...")

        try:
            # Read both CSV files
            df1 = self.read_csv(file_path_1)  # Products per shipment
            df2 = self.read_csv(file_path_2)  # Route info

            logger.info(f"shipping_data_1.csv contains {len(df1)} rows")
            logger.info(f"shipping_data_2.csv contains {len(df2)} rows")

            # Create a mapping from shipment_identifier to route info
            route_mapping = df2.set_index('shipment_identifier')[['origin_warehouse', 'destination_store

            # Group products by shipment_identifier and count quantities
            shipment_products = df1.groupby(['shipment_identifier', 'product']).size().reset_index(name=

            rows_inserted = 0
            for _, row in shipment_products.iterrows():
                try:
                    shipment_id = row['shipment_identifier']
                    product_name = str(row['product']).strip()
                    quantity = int(row['quantity'])

                    # Get route info for this shipment
                    if shipment_id in route_mapping:
                        origin = route_mapping[shipment_id]['origin_warehouse']
                        destination = route_mapping[shipment_id]['destination_store']
                    else:
                        origin = "Unknown Origin"
                        destination = "Unknown Destination"
                        logger.warning(f"No route info found for shipment {shipment_id}")

                    # Get or create product ID
                    product_id = self.get_or_create_product_id(product_name)

                    # Insert shipment record
                    self.cursor.execute(
                        "INSERT INTO shipment (product_id, quantity, origin, destination) VALUES (%s, %s
                        (product_id, quantity, origin, destination)
                    )
                    rows_inserted += 1

                except Exception as e:
                    logger.warning(f"Error processing shipment product: {e}")
                    continue

            self.connection.commit()
            logger.info(f"Inserted {rows_inserted} shipment records from shipping_data_1.csv and shippin

        except Exception as e:
            logger.error(f"Error processing shipping_data_1.csv and shipping_data_2.csv: {e}")
            raise

    def validate_data_insertion(self) -> None:
        """Validate that data was inserted correctly by running some basic checks."""
        try:
            # Count total products
            self.cursor.execute("SELECT COUNT(*) FROM product")
            total_products = self.cursor.fetchone()[0]

            # Count total shipments
            self.cursor.execute("SELECT COUNT(*) FROM shipment")
            total_shipments = self.cursor.fetchone()[0]

            # Get total quantity
            self.cursor.execute("SELECT SUM(quantity) FROM shipment")
            total_quantity = self.cursor.fetchone()[0] or 0

            logger.info(f"Validation Results:")
            logger.info(f"  Total products: {total_products}")
            logger.info(f"  Total shipments: {total_shipments}")
            logger.info(f"  Total quantity: {total_quantity}")
```

```
            # Show some sample data
            self.cursor.execute("""
                SELECT p.name, s.quantity
                FROM shipment s
                JOIN product p ON s.product_id = p.id
                ORDER BY s.id
                LIMIT 10
            """)
            sample_data = self.cursor.fetchall()
            logger.info(f"Sample data (first 10 records): {sample_data}")

            # Show product distribution
            self.cursor.execute("""
                SELECT p.name, COUNT(s.id) as shipment_count, SUM(s.quantity) as total_quantity
                FROM product p
                LEFT JOIN shipment s ON p.id = s.product_id
                GROUP BY p.id, p.name
                ORDER BY total_quantity DESC
                LIMIT 10
            """)
            product_stats = self.cursor.fetchall()
            logger.info(f"Top 10 products by total quantity: {product_stats}")

        except mysql.connector.Error as e:
            logger.error(f"Error during validation: {e}")


def main():
    """
    Main function to orchestrate the entire data processing pipeline.
    """
    # MySQL Configuration - Update these with your MySQL credentials
    MYSQL_HOST = 'localhost'
    MYSQL_PORT = 3306
    MYSQL_USER = 'root'
    MYSQL_PASSWORD = 'mmm1204'  # Update with your MySQL password
    MYSQL_DATABASE = 'shipping_db'  # Update with your database name

    # File paths
    SHIPPING_DATA_0_PATH = "data/shipping_data_0.csv"
    SHIPPING_DATA_1_PATH = "data/shipping_data_1.csv"
    SHIPPING_DATA_2_PATH = "data/shipping_data_2.csv"

    # Check if required files exist
    required_files = [SHIPPING_DATA_0_PATH, SHIPPING_DATA_1_PATH, SHIPPING_DATA_2_PATH]
    missing_files = [f for f in required_files if not os.path.exists(f)]

    if missing_files:
        logger.error(f"Missing required files: {missing_files}")
        logger.info("Please ensure all CSV files are in the data/ directory")
        return

    # Initialize processor with MySQL connection parameters
    processor = ShippingDataProcessor(
        host=MYSQL_HOST,
        port=MYSQL_PORT,
        user=MYSQL_USER,
        password=MYSQL_PASSWORD,
        database=MYSQL_DATABASE
    )

    try:
        # Connect to database
        processor.connect_to_database()

        # Create tables if they don't exist
        logger.info("Creating database tables...")
        processor.create_tables()

        # Process shipping_data_0.csv (self-contained data)
        logger.info("Starting processing of shipping_data_0.csv...")
        processor.process_shipping_data_0(SHIPPING_DATA_0_PATH)

        # Process shipping_data_1.csv and shipping_data_2.csv (dependent data)
        logger.info("Starting processing of shipping_data_1.csv and shipping_data_2.csv...")
        processor.process_shipping_data_1_and_2(SHIPPING_DATA_1_PATH, SHIPPING_DATA_2_PATH)

        # Validate the results
        logger.info("Validating data insertion...")
        processor.validate_data_insertion()

        logger.info("Data processing completed successfully!")
        logger.info(f"All shipping data has been inserted into MySQL database: {MYSQL_DATABASE}")
```

```
    except Exception as e:
        logger.error(f"Fatal error during processing: {e}")
        sys.exit(1)

    finally:
        # Clean up
        processor.close_connection()


if __name__ == "__main__":
    main()
```