Walmart Internship Task_4

Amit jakhar
amitjakhar@iitbhilai.ac.in

The purpose of this task was to perform data munging operations on CSV files and populate the data into an SQLite database. The objective was to extract relevant information from multiple spreadsheets, transform it as required, and load it into a new database for further analysis and querying.

First I made a database and then two files task_4.py and queries.sql ran in the terminal for successful completion.
I also added a file that contains data from queries on GitHub named task_4_datafromqueries.pdf

a. Creation of Database:

An SQLite database named "shipment_database.db" was created. (after running the Python script it was automatically created of size 24KB.
The "shipping_data" table was designed with appropriate columns to store the extracted data.

b. Reading and Processing CSV Files:

CSV files, named "shipping_data_0.csv," "shipping_data_1.csv," and "shipping_data_2.csv," were used as input data sources.
The Python csv module was utilized to read the CSV files.
Each CSV file was processed separately, and the relevant data was extracted.

c. Data Transformation and Loading:

The extracted data was transformed and loaded into the "shipping_data" table in the database.
The CSV data was mapped to the corresponding columns in the table.
In cases where certain columns were missing in the CSV files, default values were used to maintain consistency.

Explanation of Python script file

```
# Connect to the database
conn = sqlite3.connect('shipment_database.db')
cursor = conn.cursor()

# Drop the existing shipping_data table if it exists
cursor.execute('''DROP TABLE IF EXISTS shipping_data''')

# Create the shipping_data table with the correct columns
cursor.execute('''CREATE TABLE shipping_data (
                    id INTEGER PRIMARY KEY,
                    origin_warehouse TEXT,
                    destination_store TEXT,
                    product TEXT,
                    on_time TEXT,
                    product_quantity INTEGER,
                    driver_identifier TEXT
                )''')
```

conn = sqlite3.connect('shipment_database.db'): This line establishes a connection to the SQLite database named 'shipment_database.db'. If the database file does not exist, it will be created. The conn variable holds the connection object.

cursor = conn.cursor(): This line creates a cursor object from the connection. The cursor is used to execute SQL statements and interact with the database.

cursor.execute('''DROP TABLE IF EXISTS shipping_data'''): This SQL statement drops the 'shipping_data' table if it already exists in the database. The IF EXISTS clause ensures that the statement does not throw an error if the table doesn't exist.

cursor.execute('''CREATE TABLE shipping_data (id INTEGER PRIMARY KEY, origin_warehouse TEXT, destination_store TEXT, product TEXT, on_time TEXT, product_quantity INTEGER, driver_identifier TEXT)'''): This SQL statement creates a new table named ***'shipping_data'*** in the database. The table has the following columns:

id: An INTEGER column that serves as the primary key of the table.
origin_warehouse: A TEXT column to store the origin warehouse of the shipment.
destination_store: A TEXT column to store the destination store of the shipment.

product: A TEXT column to store the product being shipped.
on_time: A TEXT column to indicate whether the shipment arrived on time.
product_quantity: An INTEGER column to store the quantity of the product being shipped.
driver_identifier: A TEXT column to store the identifier of the driver responsible for the shipment.
This statement defines the structure and schema of the 'shipping_data' table in the database.

After executing these statements, the database connection and cursor can be used to interact with the 'shipment_database.db' database and the 'shipping_data' table.

```
# Read and insert data from Spreadsheet 0
with open('shipping_data_0.csv', 'r') as file:
    reader = csv.reader(file)
    next(reader)  # Skip header row
    for row in reader:
        origin_warehouse, destination_store, product, on_time, product_quantity, driver_identifier = row
        cursor.execute('''INSERT INTO shipping_data (origin_warehouse, destination_store, product, on_time, product_quantity, driver_identifier)
                          VALUES (?, ?, ?, ?, ?, ?)''', (origin_warehouse, destination_store, product, on_time, int(product_quantity), driver_identifier))
```

This code snippet reads data from a CSV file named 'shipping_data_0.csv' and inserts it into the 'shipping_data' table in the database.

with open('shipping_data_0.csv', 'r') as file:: This line opens the CSV file 'shipping_data_0.csv' in read mode and assigns it to the file variable. The with statement ensures that the file is properly closed after reading.

reader = csv.reader(file):r This line creates a CSV reader object reader using the csv.reader() function. The reader object allows us to iterate over the rows of the CSV file.

next(reader): This line skips the header row of the CSV file using the next() function. It moves the reader object to the next row, effectively skipping the first row of column headers.

for row in reader:: This line starts a loop that iterates over each row in the CSV file.

origin_warehouse, destination_store, product, on_time, product_quantity, driver_identifier = row: This line unpacks the values in the current row of the CSV file into separate variables. Each variable corresponds to a column in the CSV file.

cursor.execute('''INSERT INTO shipping_data (origin_warehouse, destination_store, product, on_time, product_quantity, driver_identifier) VALUES (?, ?, ?, ?, ?, ?)''',

(origin_warehouse, destination_store, product, on_time, int(product_quantity), driver_identifier)):
This line executes an SQL INSERT statement to insert the values from the CSV row into the 'shipping_data' table. The? placeholders in the SQL statement are replaced with the corresponding values from the CSV row using the second argument of the execute() method.

This code snippet is repeated for each CSV file to read and insert data into the 'shipping_data' table, allowing us to populate the table with data from multiple sources.

```
# Read and insert data from Spreadsheet 1
with open('shipping_data_1.csv', 'r') as file:
    reader = csv.reader(file)
    next(reader)  # Skip header row
    for row in reader:
        shipment_identifier, product, on_time = row
        cursor.execute('''INSERT INTO shipping_data (origin_warehouse, destination_store, product, on_time, product_quantity, driver_identifier)
                          VALUES (?, ?, ?, ?, ?, ?)''', (None, None, product, on_time, 1, None))
```

This code snippet reads data from a different CSV file named **'shipping_data_1.csv'** and inserts it into the 'shipping_data' table in the database.

with open('shipping_data_1.csv', 'r') as file:: This line opens the CSV file 'shipping_data_1.csv' in read mode and assigns it to the file variable. The with statement ensures that the file is properly closed after reading.

reader = csv.reader(file): This line creates a CSV reader object reader using the csv.reader() function to iterate over the rows of the CSV file.

next(reader): This line skips the header row of the CSV file by calling the next() function on the reader object. It moves the reader object to the next row, effectively skipping the first row of column headers.

for row in reader:: This line starts a loop that iterates over each row in the CSV file.

shipment_identifier, product, on_time = row: This line unpacks the values in the current row of the CSV file into separate variables. The variables correspond to specific columns in the CSV file.

cursor.execute('''INSERT INTO shipping_data (origin_warehouse, destination_store, product, on_time, product_quantity, driver_identifier) VALUES (?, ?, ?, ?, ?, ?)''', (None, None, product, on_time, 1, None)):
This line executes an SQL INSERT statement to insert the values from the CSV row into the 'shipping_data' table. The placeholders in the SQL statement are replaced with the corresponding values from the CSV row. In this case, origin_warehouse and destination_store

are set to None, while product and on_time are taken from the CSV row. product_quantity is set to 1, and driver_identifier is set to None.

By using this code snippet, we can read and insert data from the 'shipping_data_1.csv' file into the 'shipping_data' table.

```
# Read and insert data from Spreadsheet 2
with open('shipping_data_2.csv', 'r') as file:
    reader = csv.reader(file)
    next(reader)  # Skip header row
    for row in reader:
        shipment_identifier, origin_warehouse, destination_store, driver_identifier = row
        cursor.execute('''INSERT INTO shipping_data (origin_warehouse, destination_store, product, on_time, product_quantity, driver_identifier)
                          VALUES (?, ?, ?, ?, ?, ?)''', (origin_warehouse, destination_store, None, None, 1, driver_identifier))
```

This code snippet reads data from another CSV file named 'shipping_data_2.csv' and inserts it into the 'shipping_data' table in the database.

with open('shipping_data_2.csv', 'r') as file:: This line opens the CSV file 'shipping_data_2.csv' in read mode and assigns it to the file variable. The with statement ensures that the file is properly closed after reading.

reader = csv.reader(file): This line creates a CSV reader object reader to iterate over the rows of the CSV file.

next(reader): This line skips the header row of the CSV file by calling the next() function on the reader object. It moves the reader object to the next row, effectively skipping the first row of column headers.

for row in reader:: This line starts a loop that iterates over each row in the CSV file.

shipment_identifier, origin_warehouse, destination_store, driver_identifier = row: This line unpacks the values in the current row of the CSV file into separate variables. The variables correspond to specific columns in the CSV file.

cursor.execute('''INSERT INTO shipping_data (origin_warehouse, destination_store, product, on_time, product_quantity, driver_identifier) VALUES (?, ?, ?, ?, ?, ?)''', (origin_warehouse, destination_store, None, None, 1, driver_identifier)):
This line executes an SQL INSERT statement to insert the values from the CSV row into the 'shipping_data' table. The placeholders in the SQL statement are replaced with the corresponding values from the CSV row. In this case, origin_warehouse, destination_store, and driver_identifier are taken from the CSV row, while product and on_time are set to None. product_quantity is set to 1.

By using this code snippet, we can read and insert data from the 'shipping_data_2.csv' file into the 'shipping_data' table.

```
# Commit changes and close the connection
conn.commit()
conn.close()
```

After inserting the data into the 'shipping_data' table, the code snippet executes the following lines to commit the changes made to the database and close the connection:

conn.commit(): This line commits the changes made to the database. It ensures that all the data inserted into the 'shipping_data' table is saved permanently.

conn.close(): This line closes the connection to the database. It's important to close the connection after you're done working with the database to free up system resources and ensure proper handling of the database file.

By calling these two functions, the code snippet ensures that the changes are committed and the connection to the database is closed, completing the data insertion process.

Now queries.sql file (here we use queries)

```
-- Query 1: Get all the shipping data
SELECT * FROM shipping_data;

-- Query 2: Get the total quantity of each product
SELECT product, SUM(product_quantity) AS total_quantity
FROM shipping_data
GROUP BY product;

-- Query 3: Get the count of shipments for each origin warehouse
SELECT origin_warehouse, COUNT(*) AS shipment_count
FROM shipping_data
GROUP BY origin_warehouse;

-- Query 4: Get the count of shipments for each destination store
SELECT destination_store, COUNT(*) AS shipment_count
FROM shipping_data
GROUP BY destination_store;

-- Query 5: Get the count of on-time shipments
SELECT COUNT(*) AS on_time_shipments
FROM shipping_data
WHERE on_time = 'Yes';
```

Query 1: Get all the shipping data

```sql
-- Query 1: Get all the shipping data
SELECT * FROM shipping_data;
```

This query retrieves all the columns and rows from the ‘shipping_data’ table. It will return a result set with all the data stored in the table, including the columns ‘id’, ‘origin_warehouse’, ‘destination_store’, ‘ product’, ‘ on_time’, ‘product_quantity’, and ‘driver_identifier’.

Query_2: Get the total quantity of each product

```sql
-- Query 2: Get the total quantity of each product
SELECT product, SUM(product_quantity) AS total_quantity
FROM shipping_data
GROUP BY product;
```

This query calculates the total quantity of each product by summing up the ‘product_quantity’ values for each unique product. The result set will include two columns ‘product’ and ‘total_quantity’. Each row represents a unique product and its corresponding total quantity.

Query_3: Get the count of shipments for each origin warehouse

```sql
-- Query 3: Get the count of shipments for each origin warehouse
SELECT origin_warehouse, COUNT(*) AS shipment_count
FROM shipping_data
GROUP BY origin_warehouse;
```

Explorer

This query counts the number of shipments for each warehouse. It groups the data by the ‘orgin_warehouse’ column and uses the ‘COUNT(*)’ function to count the number of rows in each group. The result set will include two columns. ‘Origin_warehouse’ and ‘shipment_count’, where each row represents an origin warehouse and the count of shipments associated with it.

Query_4: Get the count of shipments for each destination store

```sql
-- Query 4: Get the count of shipments for each destination store
SELECT destination_store, COUNT(*) AS shipment_count
FROM shipping_data
GROUP BY destination_store;
```

This query counts the number of shipments for each destination store. It groups the data by the ‘destination_store’ column and uses the ‘COUNT(*)’ function to count the number of rows in each group. The result set will include two columns ‘destination_store’ and ‘shipment_count’, where each row represents a destination store and the count of shipments associated with it.

Query_5 Get the count of on-time shipments;

```sql
-- Query 5: Get the count of on-time shipments
SELECT COUNT(*) AS on_time_shipments
FROM shipping_data
WHERE on_time = 'Yes';
```

This query counts the number of on-time shipments. It uses the 'COUNT(*)' function to count the number of rows where the value of the 'on_time' column is 'Yes'. The result set will include a single column 'on_time_shipments' with the count of on_time shipments.

By these queries, we can retrieve specific information from the 'shipping_data' table and analyze the data based on different criteria such as product quantity,origin_warehouse, destination store, and on-time shipments.

First, run the python(task_4.py) file, and then in the terminal of queries.sql run two commands **Sqlite3 shipment_database.db** (this file was created when you ran the python file **.read queries.sql** This will show all data which we are getting by queries in **queries.sql** file.

Upon executing the Python script, the data munging process was completed successfully. The "shipment_database.db" SQLite database was created in the directory, and the "shipping_data" table was populated with the extracted data from the CSV files.

The data munging task, as outlined in Task 4, was successfully accomplished. The Python script effectively read the CSV files, transformed the data as required, and loaded it into a new SQLite database. This database can now be used for further analysis, querying, and generating insights related to shipping data.

For more understanding please contact me by amitjakhar@iitbhilai.ac.in