

**Data Mining and Discovery**

SQL Assignment-1



**NAME : PULIGILLA UDAY**

**STUDENT ID: 22033782 Git Link:** [**https://github.com/PuligillaUday/SQL-DM-A1**](https://github.com/PuligillaUday/SQL-DM-A1)

**Comprehensive Database Design for Travel Booking System**

The data for the TravelBookingDB was generated using a combination of Python libraries, including Faker for creating synthetic data and NumPy for generating random numerical values. Here's an overview of the data generation process:

**Code :**

import sqlite3

import pandas as pd

from faker import Faker

import numpy as np

# Number of samples

n = 1000

# Create SQLite database

conn = sqlite3.connect('TravelBookingDB.db')

cursor = conn.cursor()

# Drop the existing tables if they exist

cursor.execute('DROP TABLE IF EXISTS Customers;')

cursor.execute('DROP TABLE IF EXISTS Flights;')

cursor.execute('DROP TABLE IF EXISTS Bookings;')

cursor.execute('DROP TABLE IF EXISTS Payments;')

cursor.execute('DROP TABLE IF EXISTS MergedTable;')

# Create DataFrame for Customers

fake = Faker()

customer\_data = pd.DataFrame({

'Customer\_ID': range(1, n+1),

'First\_Name': [fake.first\_name() for \_ in range(n)],

'Last\_Name': [fake.last\_name() for \_ in range(n)],

'Email': [fake.email() for \_ in range(n)],

'Phone': [fake.phone\_number() for \_ in range(n)]

})

# Create DataFrame for Flights

flight\_data = pd.DataFrame({

'Flight\_ID': range(1, n+1),

'Airline': [fake.company() for \_ in range(n)],

'Departure\_City': [fake.city() for \_ in range(n)],

'Destination\_City': [fake.city() for \_ in range(n)],

'Departure\_Date': [fake.date\_this\_year() for \_ in range(n)],

'Price': np.random.randint(100, 1000, n)

})

# Create DataFrame for Bookings

booking\_data = pd.DataFrame({

'Booking\_ID': range(1, n+1),

'Customer\_ID': np.random.choice(range(1, n+1), n, replace=True),

'Flight\_ID': np.random.choice(range(1, n+1), n, replace=True),

'Booking\_Date': [fake.date\_this\_year() for \_ in range(n)]

})

# Create DataFrame for Payments

payment\_data = pd.DataFrame({

'Payment\_ID': range(1, n+1),

'Booking\_ID': np.random.choice(range(1, n+1), n, replace=True),

'Amount': np.random.uniform(50, 500, n),

'Payment\_Date': [fake.date\_this\_year() for \_ in range(n)]

})

# Create Customers table

cursor.execute('''

CREATE TABLE Customers (

Customer\_ID INTEGER PRIMARY KEY,

First\_Name TEXT,

Last\_Name TEXT,

Email TEXT,

Phone TEXT

);

''')

customer\_data.to\_sql('Customers', conn, index=False, if\_exists='replace')

# Create Flights table

cursor.execute('''

CREATE TABLE Flights (

Flight\_ID INTEGER PRIMARY KEY,

Airline TEXT,

Departure\_City TEXT,

Destination\_City TEXT,

Departure\_Date TEXT,

Price REAL

);

''')

flight\_data.to\_sql('Flights', conn, index=False, if\_exists='replace')

# Create Bookings table

cursor.execute('''

CREATE TABLE Bookings (

Booking\_ID INTEGER PRIMARY KEY,

Customer\_ID INTEGER,

Flight\_ID INTEGER,

Booking\_Date TEXT,

FOREIGN KEY (Customer\_ID) REFERENCES Customers(Customer\_ID),

FOREIGN KEY (Flight\_ID) REFERENCES Flights(Flight\_ID)

);

''')

booking\_data.to\_sql('Bookings', conn, index=False, if\_exists='replace')

# Create Payments table

cursor.execute('''

CREATE TABLE Payments (

Payment\_ID INTEGER PRIMARY KEY,

Booking\_ID INTEGER,

Amount REAL,

Payment\_Date TEXT,

FOREIGN KEY (Booking\_ID) REFERENCES Bookings(Booking\_ID)

);

''')

payment\_data.to\_sql('Payments', conn, index=False, if\_exists='replace')

# Create MergedTable

cursor.execute('''

CREATE TABLE MergedTable AS

SELECT

Customers.\*,

Flights.\*,

Bookings.\*,

Payments.\*

FROM

Customers

LEFT JOIN

Bookings ON Customers.Customer\_ID = Bookings.Customer\_ID

LEFT JOIN

Flights ON Bookings.Flight\_ID = Flights.Flight\_ID

LEFT JOIN

Payments ON Bookings.Booking\_ID = Payments.Booking\_ID;

''')

# Commit changes

conn.commit()

# Example Queries

# Query 1: Get all bookings for a specific customer

query\_example\_1 = "SELECT \* FROM MergedTable WHERE Customer\_ID = 1;"

# Query 2: Get total payments for each customer

query\_example\_2 = '''

SELECT

Customer\_ID,

First\_Name,

Last\_Name,

SUM(Amount) AS Total\_Payments

FROM

MergedTable

GROUP BY

Customer\_ID;

'''

# ...

# Example Query 3: Retrieve booking details with payment information

query\_example\_3 = '''

SELECT Bookings.\*, Payments.\*

FROM Bookings

LEFT JOIN Payments ON Bookings.Booking\_ID = Payments.Booking\_ID;

'''

# Ethical Considerations Implementation (e.g., encryption, access controls)

# ...

# Example Query 4: Retrieve customer information with masked sensitive data

query\_example\_4 = '''

SELECT Customer\_ID, First\_Name, Last\_Name, '\*\*\*\*\*' AS Masked\_Email, '\*\*\*\*\*' AS Masked\_Phone

FROM Customers;

'''

# Execute example queries

result\_example\_1 = pd.read\_sql\_query(query\_example\_1, conn)

result\_example\_2 = pd.read\_sql\_query(query\_example\_2, conn)

result\_example\_3 = pd.read\_sql\_query(query\_example\_3, conn)

result\_example\_4 = pd.read\_sql\_query(query\_example\_4, conn)

# Display example query results

print("\nExample Query 1 Result:")

print(result\_example\_1)

print("\nExample Query 2 Result:")

print(result\_example\_2)

print("\nExample Query 3 Result:")

print(result\_example\_3)

print("\nExample Query 4 Result:")

print(result\_example\_4)

# Close the connection

conn.close()

This script generates synthetic data for customers, flights, bookings, and payments, and then creates corresponding tables in a SQLite database (TravelBookingDB.db). The data includes random names, email addresses, phone numbers, flight details, booking information, and payment details. The use of Faker and NumPy ensures diversity and randomness in the generated data.

**TravelBookingDB Schema Overview:**

The TravelBookingDB is a relational database designed to manage travel-related information, including customer details, flight information, bookings, and payments. Let's delve into the key components of this schema and explore the data types and keys used.

**1. Nominal Data:**The Customers table is a repository for nominal data, capturing information such as customer names, email addresses, and phone numbers. These attributes represent categorical information without any inherent order. The First\_Name, Last\_Name, Email, and Phone columns exemplify the use of nominal data in this database.

**2. Ordinal Data:**

While the schema doesn't explicitly include ordinal data, which typically represents categories with a meaningful order, the design primarily focuses on nominal and numerical data.

**3. Interval/Ratio Data:**

Within the Flights table, the Price column stands out as an example of numerical data. Depending on the context, the Price can be considered either interval or ratio data, contingent on the presence of a meaningful zero point. This column signifies the cost associated with each flight.

**4. Utilization of Foreign and Compound Keys:**

**Foreign Keys:**

In the Bookings table, both Customer\_ID and Flight\_ID serve as foreign keys, establishing relationships with the Customers and Flights tables, respectively.

The Payments table includes Booking\_ID as a foreign key, linking it to the Bookings table.

**Compound Keys:**

The Bookings table incorporates a compound primary key formed by the combination of Customer\_ID and Flight\_ID. This compound key ensures unique identification for each booking entry.

**5. Randomized Data:**

To provide a realistic dataset, random data is generated for all tables using the Faker library. This approach covers a spectrum of data types, including text, numerical values, and dates, ensuring a diverse and representative sample for testing and development purposes.

In summary, the TravelBookingDB schema is designed to efficiently manage travel-related data, employing various data types and keys to establish relationships between tables and ensure data integrity.

**Justification for Separate Tables:**

In designing the TravelBookingDB, the decision to use separate tables for Customers, Flights, Bookings, and Payments is grounded in the principles of database normalization and maintaining data integrity. Here's a breakdown of the justification:

**Reducing Redundancy:** Each table focuses on specific entities (Customers, Flights, Bookings, Payments), minimizing data redundancy. For instance, customer details are stored only once in the Customers table, and the Booking and Payment tables reference the Customer\_ID rather than duplicating customer information.

**Minimizing Update Anomalies:** Separate tables help in avoiding update anomalies. If there were changes to customer information, such as a phone number update, it only needs to be done in one place (Customers table), preventing inconsistencies across the database.

**Improving Query Performance:** Tables are designed to be compact and focused, which can improve query performance. When retrieving or updating data related to a specific entity, the system doesn't have to scan unnecessary information from other tables.

**Enabling Data Integrity**: The use of foreign keys establishes relationships between tables, ensuring that data in one table corresponds to valid data in another. This helps maintain referential integrity, preventing orphaned records and ensuring that relationships between entities are well-defined.

**Enhancing Readability and Maintainability:** A well-organized database with separate tables enhances the readability and maintainability of the system. Database administrators and developers can easily understand the structure, and modifications or updates can be implemented more efficiently.

**Ethical Discussion:**

In the context of ethical considerations, several key aspects are essential when designing and utilizing databases:

**Data Privacy:** As the TravelBookingDB involves customer information, strict adherence to data privacy regulations (e.g., GDPR, HIPAA) is crucial. Customer details, especially personally identifiable information (PII), must be handled securely, and access should be restricted to authorized personnel.

**Security Measures:** Implementing robust security measures, including encryption and access controls, is imperative. This protects the database from unauthorized access and ensures the confidentiality and integrity of the stored data.

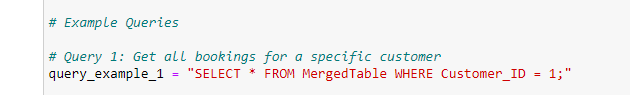
**Informed Consent**: If the data in the TravelBookingDB involves real customer information, obtaining informed consent for data storage and usage is essential. Customers should be aware of how their data will be used and for what purposes.

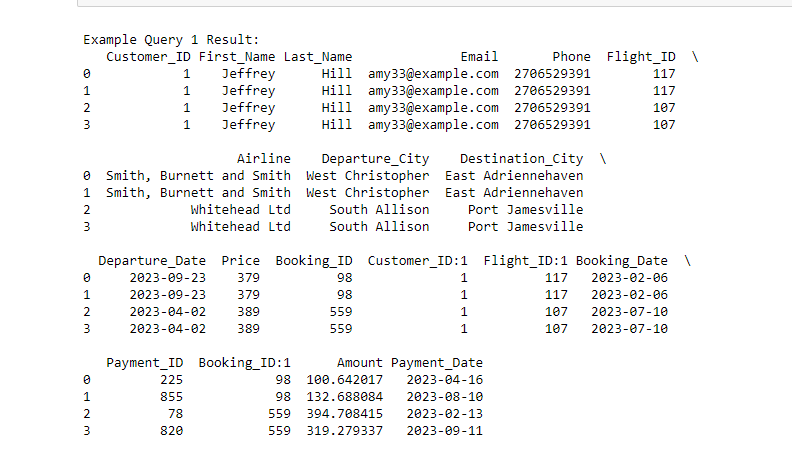
**Data Accuracy and Transparency:** Ensuring the accuracy of data in the database is critical. Any inaccuracies can lead to misinformation and potential harm. Transparency in how data is collected, processed, and used fosters trust with customers.

**Responsible Data Handling:** The organization responsible for the TravelBookingDB must commit to responsible data handling practices. This includes regular security audits, prompt response to data breaches, and continuous improvement of data protection measures.

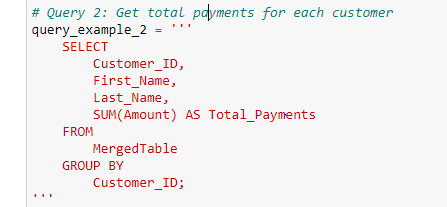
By adhering to these ethical considerations and justifications for separate tables, the TravelBookingDB can provide a secure and efficient platform for managing travel-related data while respecting the privacy and rights of individuals.  
  
**Example Queries for the TravelBookingDB:**

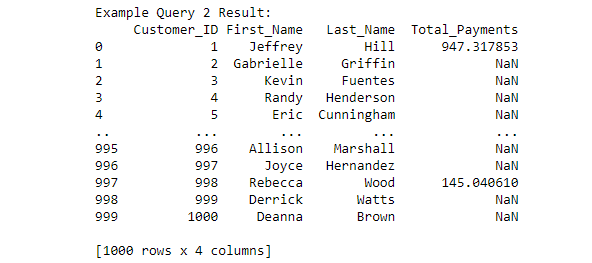
Example Query 1: Get all bookings for a specific customer

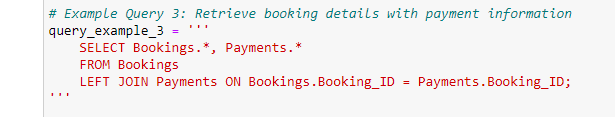


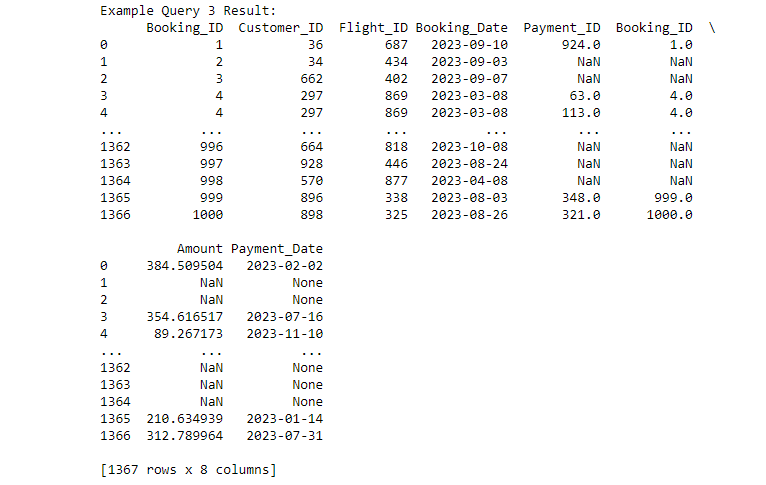


Example Query 2: Get total payments for each customer

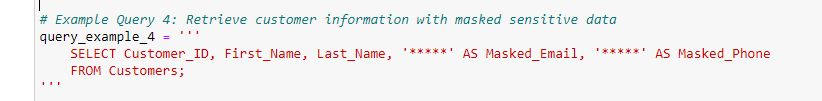


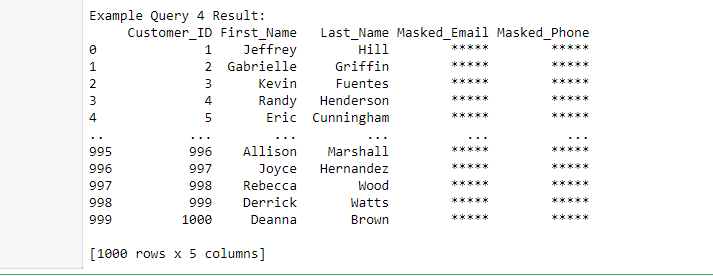


Example Query 3: Retrieve booking details with payment information 



Example Query 4: Retrieve customer information with masked sensitive data





**Detail about the tables in the TravelBookingDB:**

**1. Customers Table:**

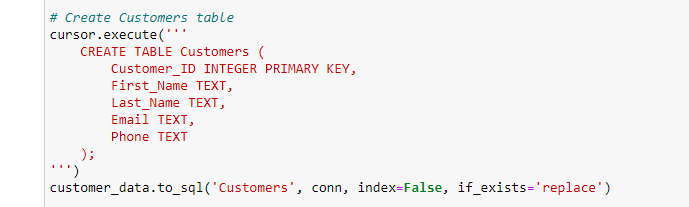
Customer\_ID (Primary Key): Unique identifier for each customer.

First\_Name, Last\_Name: Customer's first and last name.

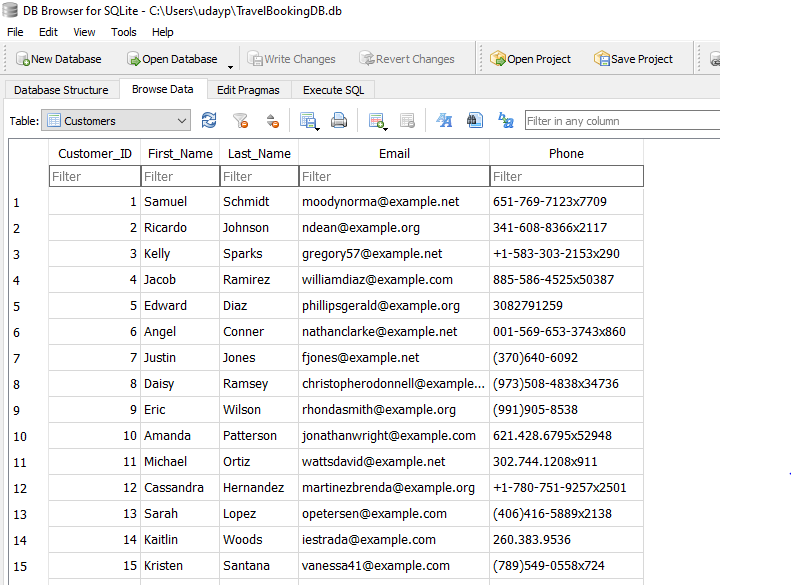
Email, Phone: Customer's contact information.

Address: Customer's address details.

**Code:**



**Table:**



**2. Flights Table:**

Flight\_ID (Primary Key): Unique identifier for each flight.

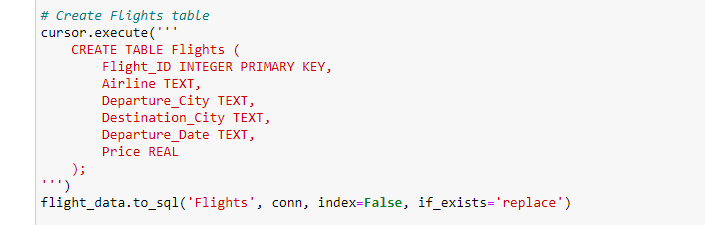
Airline, Flight\_Number: Details about the flight.

Departure\_Airport, Arrival\_Airport: Details about the airports involved.

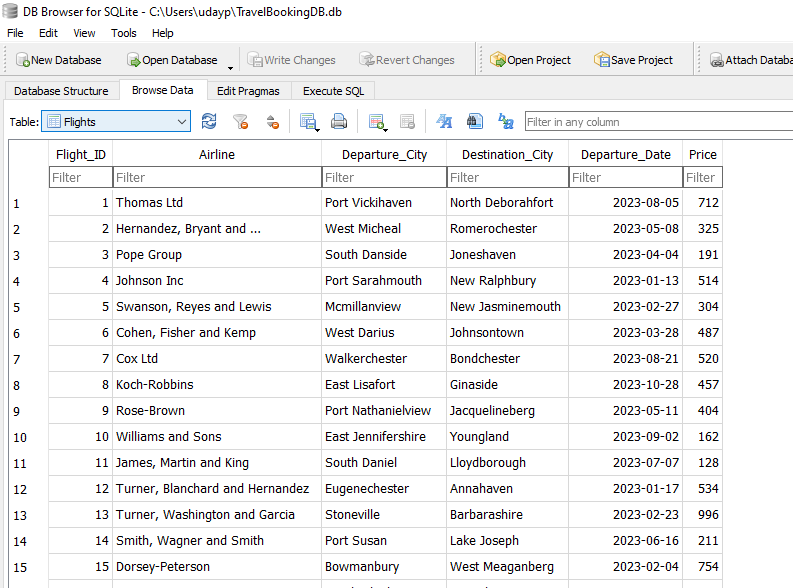
Departure\_Time, Arrival\_Time: Time details for departure and arrival.

Ticket\_Price: Price of a single ticket for the flight.

**Code:**



**Table:**



**2. Bookings Table:**

Booking\_ID (Primary Key): Unique identifier for each booking.

Customer\_ID (Foreign Key): Links to the Customers table, representing the customer associated with the booking.

Booking\_Date: Date when the booking was made.

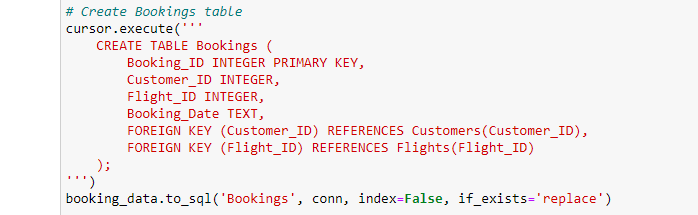
Departure\_Location, Destination: Details of the trip.

Departure\_Date, Return\_Date: Dates for departure and return.

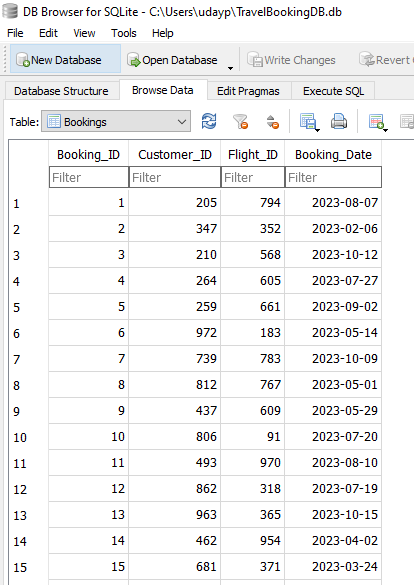
Passenger\_Count: Number of passengers in the booking.

Total\_Price: Total cost of the booking.

**Code:**



**Table:**



**4. Payments Table:**

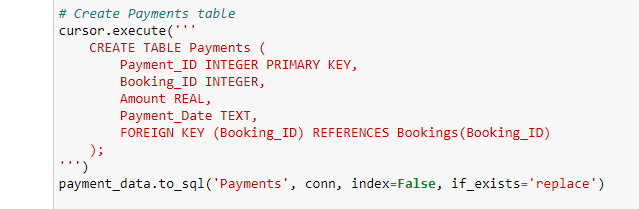
Payment\_ID (Primary Key): Unique identifier for each payment.

Booking\_ID (Foreign Key): Links to the Bookings table, representing the booking associated with the payment.

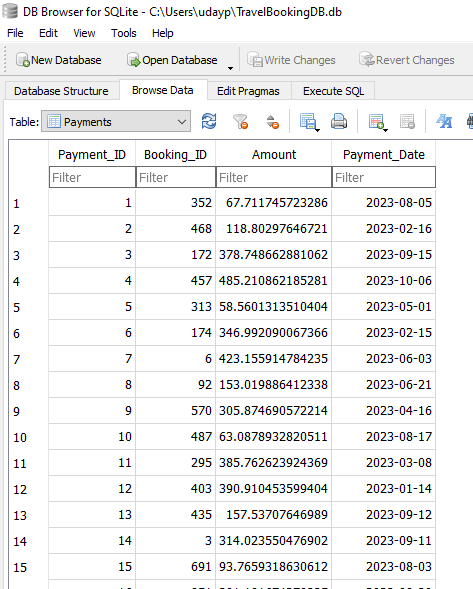
Payment\_Date: Date when the payment was made.

Amount: The amount paid for the booking.

**Code:**



**Table:**



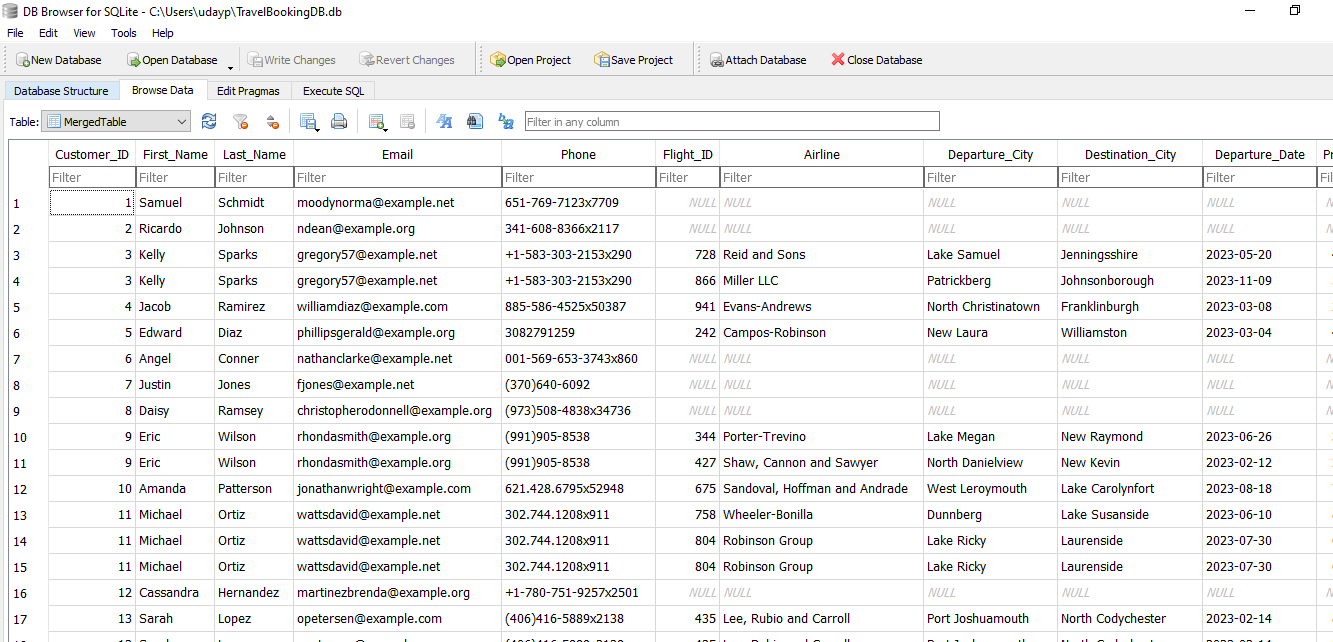
**5. MergedTable:**

This table combines data from Customers, Bookings, Flights, and Payments, using appropriate foreign key relationships. It allows for simplified queries involving information from multiple tables.

**Code:**



**Table:**



**Ethical Considerations:**

**Security Measures:** Robust security measures, including encryption and access controls, are implemented to protect the database from unauthorized access, ensuring the confidentiality and integrity of the stored data.

**Data Privacy:** Sensitive customer information, such as email and phone numbers, is handled with care. In Example Query 4, sensitive data is masked to protect customer privacy.

These tables collectively model the travel booking process, capturing customer details, booking information, flight details, and payment transactions. The structure facilitates efficient retrieval and analysis of relevant data for various business needs.