Data Mining and Discovery

SQL Assignment-1

NAME : PULIGILLA UDAY

STUDENT ID: 22033782

Git Link: 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 Queries

# Query 1: Get all bookings for a specific customer
query_example_1 = "SELECT * FROM MergedTable WHERE Customer_ID = 1;"
```

```
Example Query 1 Result:
   Customer_ID First_Name Last_Name              Email       Phone  Flight_ID  \
0            1    Jeffrey      Hill  amy33@example.com  2706529391        117
1            1    Jeffrey      Hill  amy33@example.com  2706529391        117
2            1    Jeffrey      Hill  amy33@example.com  2706529391        107
3            1    Jeffrey      Hill  amy33@example.com  2706529391        107

                   Airline    Departure_City    Destination_City  \
0  Smith, Burnett and Smith  West Christopher  East Adriennehaven
1  Smith, Burnett and Smith  West Christopher  East Adriennehaven
2            Whitehead Ltd     South Allison     Port Jamesville
3            Whitehead Ltd     South Allison     Port Jamesville

  Departure_Date  Price  Booking_ID  Customer_ID:1  Flight_ID:1 Booking_Date  \
0     2023-09-23    379          98              1          117   2023-02-06
1     2023-09-23    379          98              1          117   2023-02-06
2     2023-04-02    389         559              1          107   2023-07-10
3     2023-04-02    389         559              1          107   2023-07-10

   Payment_ID  Booking_ID:1      Amount Payment_Date
0         225            98  100.642017   2023-04-16
1         855            98  132.688084   2023-08-10
2          78           559  394.708415   2023-02-13
3         820           559  319.279337   2023-09-11
```

Example Query 2: Get total payments for each customer

```sql
# 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 2 Result:
     Customer_ID First_Name   Last_Name  Total_Payments
0              1    Jeffrey        Hill      947.317853
1              2  Gabrielle     Griffin             NaN
2              3      Kevin     Fuentes             NaN
3              4      Randy   Henderson             NaN
4              5       Eric  Cunningham             NaN
..           ...        ...         ...             ...
995          996    Allison    Marshall             NaN
996          997      Joyce   Hernandez             NaN
997          998    Rebecca        Wood      145.040610
998          999    Derrick       Watts             NaN
999         1000     Deanna       Brown             NaN

[1000 rows x 4 columns]
```

Example Query 3: Retrieve booking details with payment information

```
# 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;
'''
```

```
Example Query 3 Result:
      Booking_ID  Customer_ID  Flight_ID Booking_Date  Payment_ID  Booking_ID  \
0              1           36        687   2023-09-10       924.0         1.0
1              2           34        434   2023-09-03         NaN         NaN
2              3          662        402   2023-09-07         NaN         NaN
3              4          297        869   2023-03-08        63.0         4.0
4              4          297        869   2023-03-08       113.0         4.0
...          ...          ...        ...          ...         ...         ...
1362         996          664        818   2023-10-08         NaN         NaN
1363         997          928        446   2023-08-24         NaN         NaN
1364         998          570        877   2023-04-08         NaN         NaN
1365         999          896        338   2023-08-03       348.0       999.0
1366        1000          898        325   2023-08-26       321.0      1000.0

          Amount Payment_Date
0     384.509504   2023-02-02
1            NaN         None
2            NaN         None
3     354.616517   2023-07-16
4      89.267173   2023-11-10
...          ...          ...
1362         NaN         None
1363         NaN         None
1364         NaN         None
1365  210.634939   2023-01-14
1366  312.789964   2023-07-31

[1367 rows x 8 columns]
```

Example Query 4: Retrieve customer information with masked sensitive data

```
# 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;
'''
```

```
Example Query 4 Result:
     Customer_ID First_Name   Last_Name Masked_Email Masked_Phone
0              1     Jeffrey        Hill        *****        *****
1              2   Gabrielle     Griffin        *****        *****
2              3       Kevin     Fuentes        *****        *****
3              4       Randy   Henderson        *****        *****
4              5        Eric  Cunningham        *****        *****
..           ...         ...         ...          ...          ...
995          996     Allison    Marshall        *****        *****
996          997       Joyce   Hernandez        *****        *****
997          998     Rebecca        Wood        *****        *****
998          999     Derrick       Watts        *****        *****
999         1000      Deanna       Brown        *****        *****

[1000 rows x 5 columns]
```

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:

```
# 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')
```

Table:

DB Browser for SQLite - C:\Users\udayp\TravelBookingDB.db

File Edit View Tools Help

New Database | Open Database | Write Changes | Revert Changes | Open Project | Save Project

Database Structure | Browse Data | Edit Pragmas | Execute SQL

Table: Customers | Filter in any column

	Customer_ID	First_Name	Last_Name	Email	Phone
	Filter	Filter	Filter	Filter	Filter
1	1	Samuel	Schmidt	moodynorma@example.net	651-769-7123x7709
2	2	Ricardo	Johnson	ndean@example.org	341-608-8366x2117
3	3	Kelly	Sparks	gregory57@example.net	+1-583-303-2153x290
4	4	Jacob	Ramirez	williamdiaz@example.com	885-586-4525x50387
5	5	Edward	Diaz	phillipsgerald@example.org	3082791259
6	6	Angel	Conner	nathanclarke@example.net	001-569-653-3743x860
7	7	Justin	Jones	fjones@example.net	(370)640-6092
8	8	Daisy	Ramsey	christopherodonnell@example...	(973)508-4838x34736
9	9	Eric	Wilson	rhondasmith@example.org	(991)905-8538
10	10	Amanda	Patterson	jonathanwright@example.com	621.428.6795x52948
11	11	Michael	Ortiz	wattsdavid@example.net	302.744.1208x911
12	12	Cassandra	Hernandez	martinezbrenda@example.org	+1-780-751-9257x2501
13	13	Sarah	Lopez	opetersen@example.com	(406)416-5889x2138
14	14	Kaitlin	Woods	iestrada@example.com	260.383.9536
15	15	Kristen	Santana	vanessa41@example.com	(789)549-0558x724

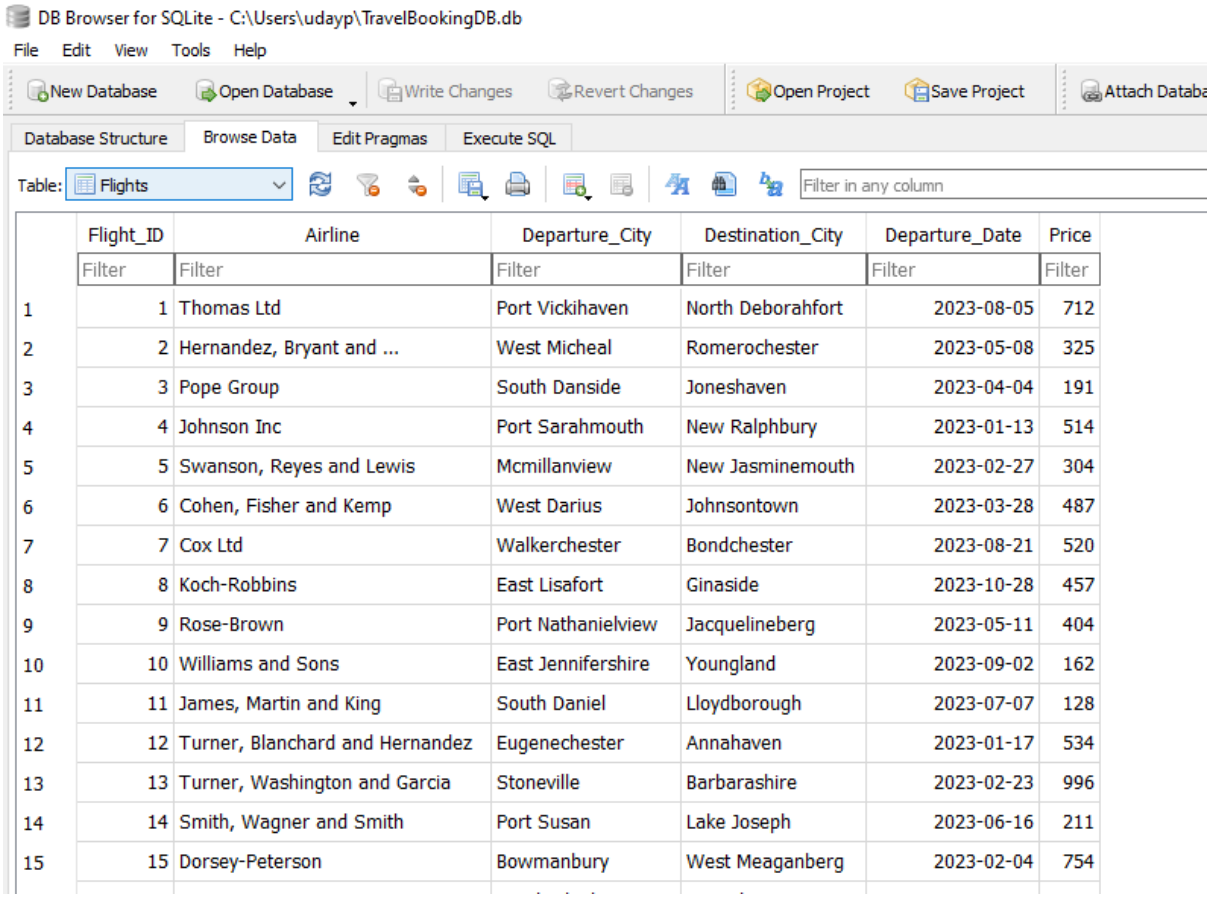
2. Flights Table:

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:

```
# 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')
```

Table:



	Flight_ID	Airline	Departure_City	Destination_City	Departure_Date	Price
	Filter	Filter	Filter	Filter	Filter	Filter
1	1	Thomas Ltd	Port Vickihaven	North Deborahfort	2023-08-05	712
2	2	Hernandez, Bryant and ...	West Micheal	Romerochester	2023-05-08	325
3	3	Pope Group	South Danside	Joneshaven	2023-04-04	191
4	4	Johnson Inc	Port Sarahmouth	New Ralphbury	2023-01-13	514
5	5	Swanson, Reyes and Lewis	Mcmillanview	New Jasminemouth	2023-02-27	304
6	6	Cohen, Fisher and Kemp	West Darius	Johnsontown	2023-03-28	487
7	7	Cox Ltd	Walkerchester	Bondchester	2023-08-21	520
8	8	Koch-Robbins	East Lisafort	Ginaside	2023-10-28	457
9	9	Rose-Brown	Port Nathanielview	Jacquelineberg	2023-05-11	404
10	10	Williams and Sons	East Jennifershire	Youngland	2023-09-02	162
11	11	James, Martin and King	South Daniel	Lloydborough	2023-07-07	128
12	12	Turner, Blanchard and Hernandez	Eugenechester	Annahaven	2023-01-17	534
13	13	Turner, Washington and Garcia	Stoneville	Barbarashire	2023-02-23	996
14	14	Smith, Wagner and Smith	Port Susan	Lake Joseph	2023-06-16	211
15	15	Dorsey-Peterson	Bowmanbury	West Meaganberg	2023-02-04	754

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:

```
# 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')
```

Table:

DB Browser for SQLite - C:\Users\udayp\TravelBookingDB.db

File Edit View Tools Help

New Database | Open Database | Write Changes | Revert

Database Structure | Browse Data | Edit Pragmas | Execute SQL

Table: Bookings

	Booking_ID	Customer_ID	Flight_ID	Booking_Date
	Filter	Filter	Filter	Filter
1	1	205	794	2023-08-07
2	2	347	352	2023-02-06
3	3	210	568	2023-10-12
4	4	264	605	2023-07-27
5	5	259	661	2023-09-02
6	6	972	183	2023-05-14
7	7	739	783	2023-10-09
8	8	812	767	2023-05-01
9	9	437	609	2023-05-29
10	10	806	91	2023-07-20
11	11	493	970	2023-08-10
12	12	862	318	2023-07-19
13	13	963	365	2023-10-15
14	14	462	954	2023-04-02
15	15	681	371	2023-03-24

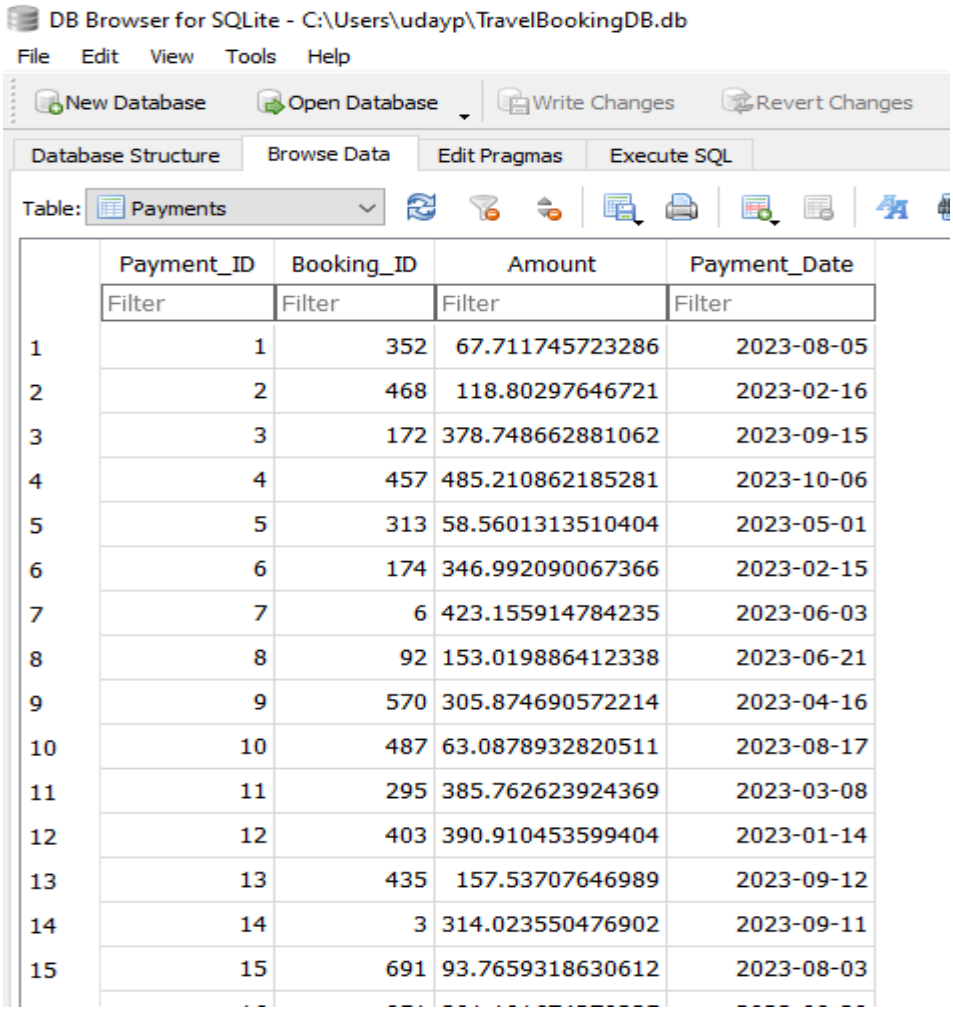
4. Payments Table:

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:

```
# 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')
```

Table:



	Payment_ID	Booking_ID	Amount	Payment_Date
	Filter	Filter	Filter	Filter
1	1	352	67.711745723286	2023-08-05
2	2	468	118.80297646721	2023-02-16
3	3	172	378.748662881062	2023-09-15
4	4	457	485.210862185281	2023-10-06
5	5	313	58.5601313510404	2023-05-01
6	6	174	346.992090067366	2023-02-15
7	7	6	423.155914784235	2023-06-03
8	8	92	153.019886412338	2023-06-21
9	9	570	305.874690572214	2023-04-16
10	10	487	63.0878932820511	2023-08-17
11	11	295	385.762623924369	2023-03-08
12	12	403	390.910453599404	2023-01-14
13	13	435	157.53707646989	2023-09-12
14	14	3	314.023550476902	2023-09-11
15	15	691	93.7659318630612	2023-08-03

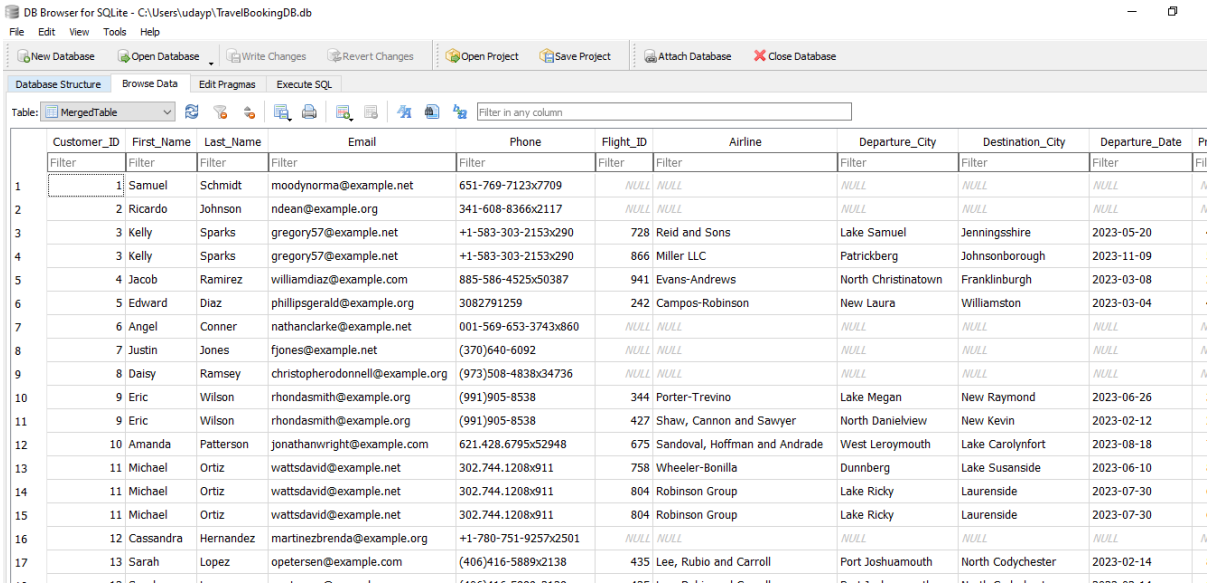
5. MergedTable:

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:

```
# 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;
''')
```

Table:



DB Browser for SQLite - C:\Users\udayp\TravelBookingDB.db

File Edit View Tools Help

New Database | Open Database | Write Changes | Revert Changes | Open Project | Save Project | Attach Database | Close Database

Database Structure | Browse Data | Edit Pragmas | Execute SQL

Table: MergedTable | Filter in any column

	Customer_ID	First_Name	Last_Name	Email	Phone	Flight_ID	Airline	Departure_City	Destination_City	Departure_Date
	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	1	Samuel	Schmidt	moodynorma@example.net	651-769-7123x7709	NULL	NULL	NULL	NULL	NULL
2	2	Ricardo	Johnson	ndean@example.org	341-608-8366x2117	NULL	NULL	NULL	NULL	NULL
3	3	Kelly	Sparks	gregory57@example.net	+1-583-303-2153x290	728	Reid and Sons	Lake Samuel	Jenningsshire	2023-05-20
4	3	Kelly	Sparks	gregory57@example.net	+1-583-303-2153x290	866	Miller LLC	Patrickberg	Johnsonborough	2023-11-09
5	4	Jacob	Ramirez	williamdiaz@example.com	885-586-4525x50387	941	Evans-Andrews	North Christinatown	Franklinburgh	2023-03-08
6	5	Edward	Diaz	phillipsgerald@example.org	3082791259	242	Campos-Robinson	New Laura	Williamston	2023-03-04
7	6	Angel	Conner	nathanclarke@example.net	001-569-653-3743x860	NULL	NULL	NULL	NULL	NULL
8	7	Justin	Jones	fjones@example.net	(370)640-6092	NULL	NULL	NULL	NULL	NULL
9	8	Daisy	Ramsey	christopherodonnell@example.org	(973)508-4838x34736	NULL	NULL	NULL	NULL	NULL
10	9	Eric	Wilson	rhondasmith@example.org	(991)905-8538	344	Porter-Trevino	Lake Megan	New Raymond	2023-06-26
11	9	Eric	Wilson	rhondasmith@example.org	(991)905-8538	427	Shaw, Cannon and Sawyer	North Danielview	New Kevin	2023-02-12
12	10	Amanda	Patterson	jonathanwright@example.com	621.428.6795x52948	675	Sandoval, Hoffman and Andrade	West Leroymouth	Lake Carolynfort	2023-08-18
13	11	Michael	Ortiz	wattsdavid@example.net	302.744.1208x911	758	Wheeler-Bonilla	Dunnberg	Lake Susanside	2023-06-10
14	11	Michael	Ortiz	wattsdavid@example.net	302.744.1208x911	804	Robinson Group	Lake Ricky	Laurenside	2023-07-30
15	11	Michael	Ortiz	wattsdavid@example.net	302.744.1208x911	804	Robinson Group	Lake Ricky	Laurenside	2023-07-30
16	12	Cassandra	Hernandez	martinezbrenda@example.org	+1-780-751-9257x2501	NULL	NULL	NULL	NULL	NULL
17	13	Sarah	Lopez	opetersen@example.com	(406)416-5889x2138	435	Lee, Rubio and Carroll	Port Joshuamouth	North Codychester	2023-02-14

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.