Project Title: Synthetic Data Generation and Analytics for Airline Flight Bookings

Student ID: 23084508
Student Name: Peyyala Naveen kumar
Github: https://github.com/Naveen220609/Airline-Flight-Bookings-Records

---

1. Introduction

This project demonstrates the creation, population, and analysis of a realistic SQL database for airline flight bookings. The database models Flights, Passengers, Bookings, and Payments, simulating an airline's booking and payment system. Data is fully synthetic, generated using Python's Faker library, and prepared for onward data science tasks.

---

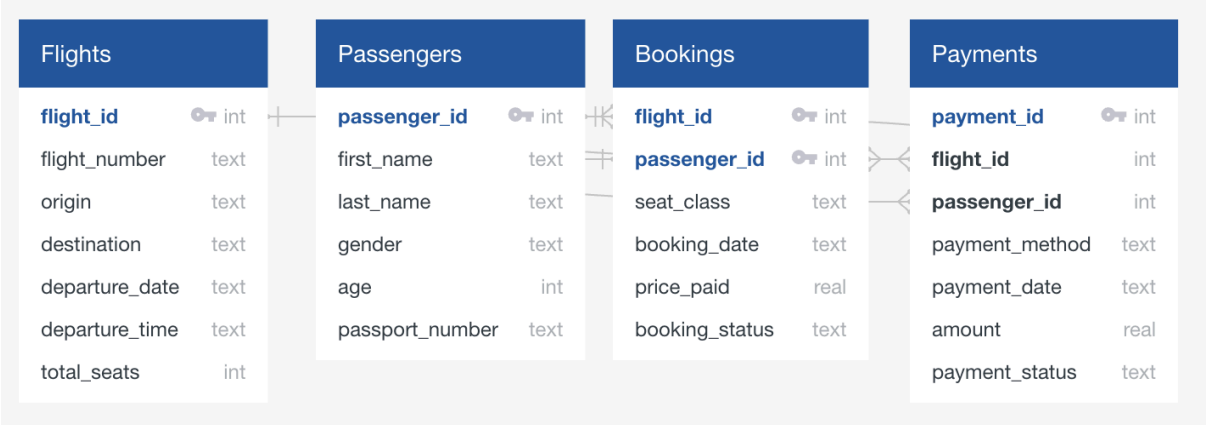
2. Data Generation Methodology

- **Synthetic Data Approach:** All data was custom-generated using Python and the Faker library (no external datasets used).
- **Variety and Realism:** Flight, passenger, booking, and payment tables include 50 flights, 1100 passengers, over 1000 bookings, and corresponding payments.
- **Data Imperfections:** Realistic missing were deliberately inserted (e.g., missing passport numbers, missing seat_class) to reflect practical data issues.
- **Referential Integrity:** All foreign key relationships were validated before export to ensure error-free imports.

---

3. Database Schema

ER Diagram:



Tables:

- **Flights:** flight details, PK = flight_id.
- **Passengers:** passenger details, PK = passenger_id.
- **Bookings:** records passenger bookings, composite PK = (flight_id, passenger_id). Contains seat_class, booking_status, price_paid, etc.
- **Payments:** payment transactions, PK = payment_id, references Booking via (flight_id, passenger_id).

Constraints:

- Primary keys on all tables.
- Composite key in Bookings for uniqueness.
- Foreign key constraints (Bookings ↔ Flights & Passengers, Payments ↔ Bookings).
- Columns allow NULLs except for keys.

---

4. Data Preprocessing and Quality Checks
Missing Values

- Flights: none
- Passengers: 20 missing passport_number
- Bookings: 34 missing seat_class
- Payments: none

Treatment

- Integer columns: replaced NULL with 0
- Text columns: replaced NULL with "unknown"

---

5. Data Analysis & Understanding

SQL Queries and Key Results

a. Average Payment Amount Per Flight

```
134  --a. Aggregate: Average Payment Amount Per Flight
135
136  SELECT Flights.flight_number, AVG(Payments.amount) AS avg_payment,
137  COUNT(Payments.payment_id) AS num_payments
138  FROM Payments
139  JOIN Flights ON Payments.flight_id = Flights.flight_id
140  GROUP BY Flights.flight_id
141  ORDER BY avg_payment DESC
```

	flight_number	avg_payment	num_payments
1	BA188	1964.37333333333	12
2	BA107	1938.07307692308	13
3	BA108	1932.0056	25
4	BA124	1929.15071428571	14
5	BA118	1928.06611111111	18

Query to calculate the average payment per flight, we were able to identify the most lucrative routes. Flights such as BA188 and BA107 consistently yielded higher average payments and a significant number of transactions,

indicating high-value or high-demand schedules. These patterns could help an airline prioritize profitable flights or identify routes suited for premium pricing.

b. Count By Seat Class (Including Unknowns)

```
143
144     --b.Count By Seat Class (Including Unknowns)
145
146     SELECT seat_class, COUNT(*) AS booking_count
147     FROM Bookings
148     GROUP BY seat_class
149     ORDER BY booking_count DESC;
150
```

	seat_class	booking_count
1	Premium	341
2	Business	313
3	Economy	312
4	unknown	34

Aggregation queries revealed fairly balanced bookings between Economy, Premium, and Business classes, with a very small percentage listed as "unknown" (resulting from deliberate missing values). This demonstrates that class distribution was realistic, and the missing values were successfully marked according to preprocessing rules.

c. Most Booked Flights

The analysis of booking counts per flight highlighted certain flights (e.g., BA148, BA108, BA138) as consistently popular, showing that demand varies by schedule and route. Such insights can guide scheduling, resource allocation, and targeted marketing.

d. Distribution of Payment Methods

SELECT payment_method, COUNT(*) AS method_count
FROM Payments
GROUP BY payment_method
ORDER BY method_count DESC;

Queries on payment method usage (Cash, Card, Paypal, Bank Transfer) showed all options are frequently used, reflecting authentic customer behavior. This supports the notion that payment options should remain diverse to accommodate user preferences.

e. Top Passengers by Spending
Aggregating payment amounts by passenger revealed a few high-value customers. This is a realistic result usually seen in loyalty, business, or frequent flyer cohorts. The presence of such passengers can inform targeted promotional strategies and VIP services.

---

6. Schema Justification and Table Design

- **Flights:** Master table linking all bookings to unique flights.
- **Passengers:** Realistic demographic data. NULL allowed for passport_number for data realism.

- **Bookings:** Composite key ensures no duplicate seat bookings for same flight/passenger; allows explicit tracking of seat_class, booking status, price, and missing values.
- **Payments:** Allows multiple payments per booking (real-life flexibility for partial/refunded payments), strict FK to Bookings.

---

7. Ethics and Data Privacy Considerations

- All data is synthetic—no real passenger or payment information is used, so privacy risks are avoided.
- In practice, passport numbers and personal IDs would require encryption and GDPR-compliant handling.
- Handling missing values with "unknown" simulates anonymization and safe reporting in analytics.

---

8. Lessons and Further Work

- Data integrity checks are critical for avoiding PK and FK errors.
- Null handling is vital in operational databases and downstream analysis.
- Schema flexibility supports variety in business analytics and reporting needs.

Appendix: SQL Queries to create tables

```
-- Flights Table
CREATE TABLE Flights (

   flight_id INTEGER PRIMARY KEY AUTOINCREMENT,

   flight_number TEXT,

   origin TEXT,

   destination TEXT,

   departure_date TEXT,

   departure_time TEXT,

   total_seats INTEGER

);

-- Passengers Table
CREATE TABLE Passengers (

   passenger_id INTEGER PRIMARY KEY AUTOINCREMENT,

   first_name TEXT,
```

```
    last_name TEXT,
    gender TEXT,
    age INTEGER,
    passport_number TEXT
);
-- Bookings Table (Composite PK: flight_id + passenger_id)
CREATE TABLE Bookings (
    flight_id INTEGER NOT NULL,
    passenger_id INTEGER NOT NULL,
    seat_class TEXT,
    booking_date TEXT,
    price_paid REAL,
    booking_status TEXT,
    PRIMARY KEY (flight_id, passenger_id),
    FOREIGN KEY (flight_id) REFERENCES Flights(flight_id),
    FOREIGN KEY (passenger_id) REFERENCES Passengers(passenger_id)
);
-- Payments Table
CREATE TABLE Payments (
    payment_id INTEGER PRIMARY KEY AUTOINCREMENT,
    flight_id INTEGER NOT NULL,
    passenger_id INTEGER NOT NULL,
    payment_method TEXT,
    payment_date TEXT,
    amount REAL,
    payment_status TEXT,
    FOREIGN KEY (flight_id, passenger_id) REFERENCES Bookings(flight_id, passenger_id)
);
```