SQL Analysis of Airline Database

Authors: Ayman EL ALASS & Abderaouf KHELFAOUI **Objective:** Execution of analytical queries ranging from simple data extraction to complex analysis and schema modification.

In [ ]:
```python
import sqlite3
import pandas as pd

# Connect to the database generated by main.py
db_name = "project_database.db"
conn = sqlite3.connect(db_name)
print(f"Connected to database: {db_name}")

# Helper function to execute SQL and return a readable DataFrame
def run_query(query):
    try:
        df = pd.read_sql(query, conn)
        return df
    except Exception as e:
        print(f"SQL Error: {e}")
```

1. Basic Data Retrieval (Sanity Check)

Scenario: A specific request from the Operations Center. We need to list details of flights operated by 'American Airlines Inc.' that faced extreme delays (> 4 hours) to identify patterns in specific airports.

In [ ]:
```python
query_1 = """
SELECT
    f.flight_date,
    f.flight_number,
    f.origin_airport,
    f.dest_airport,
    f.dep_delay || ' min' as delay_minutes -- Formatting for readability
FROM FLIGHTS f
JOIN AIRLINES a ON f.airline_code = a.airline_code
WHERE a.airline_name = 'American Airlines Inc.'
  AND f.dep_delay > 240
ORDER BY f.dep_delay DESC
LIMIT 10;
"""

print(">>> Extreme Delays Report (American Airlines):")
display(run_query(query_1))
```

2. Airline Performance Audit

Exemple Question: Which airlines are the most reliable? We calculate three key KPIs per airline:

1. **Volume:** Total flights.
2. **Punctuality:** Average departure delay.
3. **Reliability:** Cancellation Rate (%).

In [ ]:
```
query_2 = """
SELECT
    a.airline_name,
    COUNT(f.flight_id) as total_flights,
    ROUND(AVG(f.dep_delay), 2) as avg_delay_min,
    SUM(CASE WHEN f.cancelled = 1 THEN 1 ELSE 0 END) as cancelled_count,
    ROUND(
        (CAST(SUM(CASE WHEN f.cancelled = 1 THEN 1 ELSE 0 END) as FLOAT) / COUNT
    2) || '%' as cancellation_rate
FROM FLIGHTS f
JOIN AIRLINES a ON f.airline_code = a.airline_code
GROUP BY a.airline_name
HAVING total_flights > 500 -- Filter to keep only major airlines
ORDER BY avg_delay_min ASC;
"""

print(">>> Airline Performance Matrix (Ranked by Punctuality):")
display(run_query(query_2))
```

3. Route Analysis (Double Join)

Context: We want to identify the specific City-to-City connections that suffer from the worst delays.

Technique: We perform a **Double Join** on the `AIRPORTS` table (aliased as `origin` and `dest`) to retrieve readable city names instead of IATA codes.

In [ ]:
```
query_3 = """
SELECT
    origin.city || ' -> ' || dest.city AS Route,
    COUNT(*) as Flight_Count,
    ROUND(AVG(f.dep_delay), 2) as Avg_Delay_Min,
    MAX(f.dep_delay) as Max_Delay_Min
FROM FLIGHTS f
JOIN AIRPORTS origin ON f.origin_airport = origin.iata_code
JOIN AIRPORTS dest ON f.dest_airport = dest.iata_code
GROUP BY f.origin_airport, f.dest_airport
HAVING Flight_Count > 20 -- Ignore rare charter routes
ORDER BY Avg_Delay_Min DESC
LIMIT 10;
"""

print(">>> Top 10 Routes with Highest Average Delays:")
display(run_query(query_3))
```

4. Exemple of Weather Impact Querying

Context: For instance we can assume that to avoid heavy processing times, we analyze the correlation between wind and delays for a **single specific day** (January 1st, 2015).

Technique: We aggregate the average wind speed and average delay per airport for that day.

In [ ]:
```
query_4 = """
WITH DailyWeather AS (
    SELECT
        airport_code,
        AVG(wind_speed) as avg_wind_speed
    FROM WEATHER
    WHERE date(reading_time) = '2015-01-01'
    GROUP BY airport_code
),
DailyFlights AS (
    SELECT
        origin_airport,
        AVG(dep_delay) as avg_dep_delay
    FROM FLIGHTS
    WHERE date(flight_date) = '2015-01-01'
    GROUP BY origin_airport
)
SELECT
    f.origin_airport,
    f.avg_dep_delay,
    w.avg_wind_speed
FROM DailyFlights f
JOIN DailyWeather w ON f.origin_airport = w.airport_code
ORDER BY f.avg_dep_delay DESC;
"""

df_result = run_query(query_4)
display(df_result)
```

5. Database Evolution

Requirement: To optimize future reporting, we need to persist the "Delay Category" directly in the database table, rather than calculating it every time.

Actions:

1. **ALTER TABLE:** Add a new column `delay_category`.
2. **UPDATE:** Populate this column based on the `dep_delay` value.

In [ ]:
```
# 1. Add the column structure
try:
    conn.execute("ALTER TABLE FLIGHTS ADD COLUMN delay_category VARCHAR(20)")
    print("Schema Altered: Column 'delay_category' added.")
except sqlite3.OperationalError:
    print("Column 'delay_category' already exists.")

# 2. Populate the data
update_query = """
UPDATE FLIGHTS
SET delay_category = CASE
    WHEN dep_delay <= 0 THEN 'On Time / Early'
    WHEN dep_delay > 0 AND dep_delay <= 15 THEN 'Small Delay'
    WHEN dep_delay > 15 AND dep_delay <= 45 THEN 'Medium Delay'
```

```
    ELSE 'Major Delay (>45m)'
END;
"""
conn.execute(update_query)
conn.commit()
print("Data Updated: Categories populated.")
```

In [ ]:
```
# 3. Verification Query
query_check = """
SELECT
    delay_category,
    COUNT(*) as flight_count,
    ROUND((CAST(COUNT(*) as FLOAT) / (SELECT COUNT(*) FROM FLIGHTS)) * 100, 1) |
FROM FLIGHTS
GROUP BY delay_category
ORDER BY flight_count DESC;
"""

print(">>> Verification: Distribution of new categories:")
display(run_query(query_check))
```

6. Global Statistics

In [ ]:
```
# Total flights in 2015
query_total = "SELECT COUNT(flight_id) as 'Total Flights 2015' FROM flights;"
display(run_query(query_total))

# Busiest day of the year
query_busiest_day = """
SELECT flight_date, COUNT(flight_id) as number_of_flights
FROM flights
WHERE strftime('%Y', flight_date) = '2015'
GROUP BY flight_date
ORDER BY number_of_flights DESC
LIMIT 1;
"""
print(">>> Busiest day of 2015:")
display(run_query(query_busiest_day))
```

7. Airport & Location Analysis

In [ ]:
```
# Busiest Airport (Origin)
query_busiest_airport = """
SELECT f.origin_airport, a.airport_name, COUNT(f.flight_id) AS number_of_flights
FROM flights f
JOIN airports a ON f.origin_airport = a.iata_code
WHERE strftime('%Y', f.flight_date) = '2015'
GROUP BY f.origin_airport
ORDER BY number_of_flights DESC
LIMIT 1;
"""
print(">>> Busiest Airport:")
display(run_query(query_busiest_airport))
```

In [ ]:
```
# Peak traffic day per State (Complex Sub-query)
query_state_peak = """
```

```
WITH DailyStats AS (
    SELECT
        a.state,
        f.origin_airport,
        a.airport_name,
        f.flight_date,
        COUNT(f.flight_id) AS number_of_flights,
        RANK() OVER (
            PARTITION BY a.state
            ORDER BY COUNT(f.flight_id) DESC
        ) as rang
    FROM flights f
    JOIN airports a ON f.origin_airport = a.iata_code
    WHERE f.flight_date BETWEEN '2015-01-01' AND '2015-12-31'
    GROUP BY a.state, f.origin_airport, a.airport_name, f.flight_date
)
SELECT
    state,
    origin_airport,
    airport_name,
    flight_date,
    number_of_flights
FROM DailyStats
WHERE rang = 1
ORDER BY state;
"""

print(">>> Peak Traffic Day per State (Optimized):")
display(run_query(query_state_peak))
```

8. Delays and Cancellations

In [ ]:

```
# Average Delay per Airline per Airport
query_delay_airline_airport = """
SELECT ar.airline_name, f.origin_airport, ROUND(AVG(f.dep_delay), 2) AS avg_dep_
FROM flights f
JOIN airports ap ON f.origin_airport = ap.iata_code
JOIN airlines ar ON f.airline_code = ar.airline_code
WHERE strftime('%Y', f.flight_date) = '2015'
GROUP BY ar.airline_name, f.origin_airport
ORDER BY avg_dep_delay DESC
LIMIT 10;
"""
print(">>> Top 10 Highest Average Delays (Airline/Airport):")
display(run_query(query_delay_airline_airport))
```

In [ ]:
```
# Top 3 Airports with most cancellations
query_cancelled_airports = """
SELECT origin_airport, airport_name, COUNT(cancelled) AS cancelled_count
FROM airports ap
JOIN flights f ON ap.iata_code = f.origin_airport
WHERE cancelled = 1
GROUP BY origin_airport, airport_name
ORDER BY cancelled_count DESC
LIMIT 3;
"""
print(">>> Top 3 Airports for Cancellations:")
display(run_query(query_cancelled_airports))
```