Project Report: Flight Delay & Weather Database

Databases 1 – Master 1

Ayman EL ALASS
Abderaouf KHELFAOUI

December 2025

University of Lille

Abstract

This project focuses on creating a relational database from raw CSV datasets concerning US flight traffic and historical weather conditions. As required by the course guidelines, we modeled the database structure, processed and cleaned the raw data, populated the tables using Python, and performed significant SQL queries to analyze the data. This report details our modeling choices, the strategies used to handle data inconsistencies, and the results of our analysis.

Contents

1 Introduction

For the "Databases 1" course project, we selected a dataset that allows for complex relational modeling: **flight delays**. To create a more interesting and challenging database, we combined this flight data with a second dataset containing **historical weather information**.

The objective was to identify raw data, structure it into a relational database (SQLite), and then run realistic queries to answer questions such as:

- Which airlines have the best punctuality?
- Which routes suffer the most from delays?
- Does weather (e.g., wind speed) impact flight cancellations?

2 Data Sourcing

We used two primary data sources downloaded from Kaggle:

1. **Flight Delays:** A large dataset containing flight schedules, delays, and cancellations for the year 2015.
2. **Historical Hourly Weather:** Containing temperature and wind speed for major US cities.

As per the project guidelines, we had to manage the merging of these different sources, handling redundant information and ensuring data consistency.

3 Database Modeling

We designed a relational schema to structure this data efficiently, justifying our choices for tables, attributes, and constraints.

3.1 Conceptual Data Model (MCD)

Our model is centered on the **FLIGHTS** entity.

- A flight is operated by one **AIRLINE**.
- A flight has an origin **AIRPORT** and a destination **AIRPORT**.
- **WEATHER** conditions are recorded at specific airports.

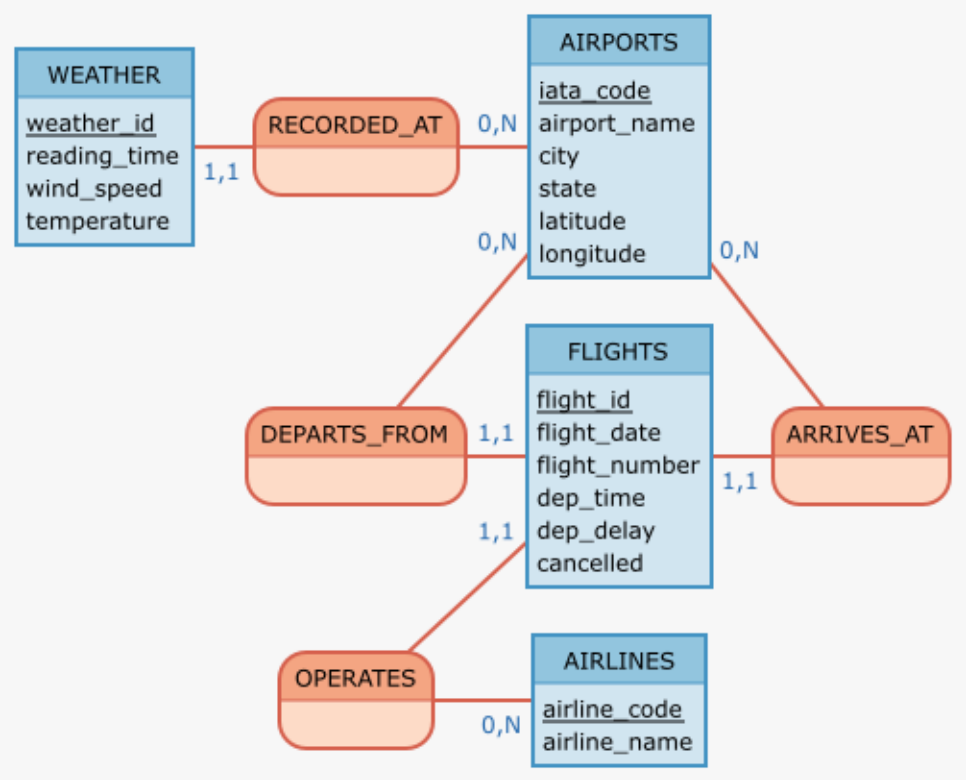

Figure 1: Conceptual Data Model (MCD)

3.2 Logical Data Model (MLD) & Implementation

We translated the conceptual model into a logical schema (MLD) and then into SQL. We made specific choices regarding data types for SQLite:

- `CHAR(2)` and `CHAR(3)` for standard codes (IATA) to ensure consistency.
- `TEXT` for time columns, as SQLite does not have a specific Time type.

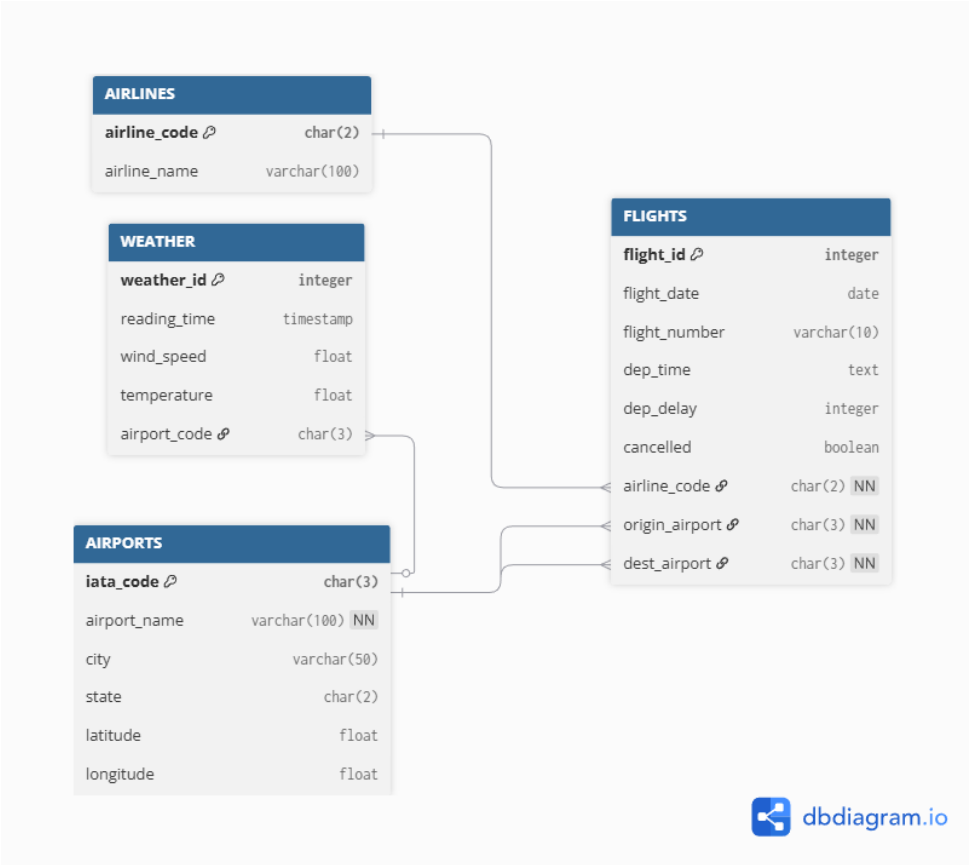

Figure 2: Logical Data Model (MLD)

3.3 Schema Evolution

During the project, we refined our model. For instance, we realized the `AIRPORTS` table needed an `airport_name` field to be more useful. We updated our schema (SQL script) to reflect this, ensuring the database accurately represents the real-world entities.

```sql
CREATE TABLE FLIGHTS (
    flight_id INTEGER PRIMARY KEY,
    flight_date DATE,
    flight_number VARCHAR(10),
    dep_time TEXT,
    dep_delay INTEGER,
    cancelled BOOLEAN,
    airline_code CHAR(2),
    origin_airport CHAR(3),
    dest_airport CHAR(3),
    FOREIGN KEY (airline_code) REFERENCES AIRLINES(airline_code),
    FOREIGN KEY (origin_airport) REFERENCES AIRPORTS(iata_code),
    FOREIGN KEY (dest_airport) REFERENCES AIRPORTS(iata_code)
);
```

Listing 1: Final SQL Schema for Flights

4 Population of Tables & Data Cleaning

Populating the database required handling several data quality issues. We used a Python script (`main.py`) to read the CSV files, clean the data, and insert it into the SQLite database.

4.1 Handling Inconsistencies

We analyzed the raw data (detailed in `dataprocessing.ipynb`) and applied rules to fix problems before insertion:

Dataset	Issue Identified	Solution Applied
Flights	Time format '2400'	Converted to '23:59:00' for SQL compatibility
Flights	Missing Delays	Imputed with 0 (Assumed on time)
Flights	Unknown Airports	Removed flights referencing missing airports
Weather	Units in Kelvin	Converted to Celsius for readability
Weather	Sensor Errors	Filtered out unrealistic temperatures ($< -60°C$)
Weather	Location Key	Mapped City Names to IATA Airport Codes

Table 1: Summary of Data Cleaning Strategy

4.2 Code Implementation

The script checks for the existence of CSV files and processes them in chunks to manage memory usage efficiently.

```
# Cleaning weather data before insertion
df_weather['temperature'] = df_weather['temperature_k'] - 273.15
df_weather = df_weather[
    (df_weather['temperature'] > -60) &
    (df_weather['temperature'] < 60)
]
```

Listing 2: Example of Data Cleaning in Python

5 Querying the Database

With the database populated (approx. 500k flights in our sample), we performed various SQL queries to extract insights, as demonstrated in the `query.ipynb` notebook.

5.1 Complex Analysis: Peak Traffic

We aimed to find the busiest day of the year for *each* US State. To do this efficiently, we used a window function (`RANK()`) which allows us to rank days by flight count within each state group in a single query.

```
WITH DailyStats AS (
    SELECT a.state, 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.flight_date
)
SELECT * FROM DailyStats WHERE rang = 1;
```

Listing 3: Optimized Query for State Traffic

5.2 Modifying the Database

As suggested in the guidelines, we also proposed a realistic modification to the database structure. We added a new column `delay_category` to the `FLIGHTS` table and populated it using an SQL `UPDATE` statement to categorize delays (e.g., "Small Delay", "Major Delay"). This optimization helps in future reporting.

6 Conclusion

This project allowed us to apply the concepts learned in class: identifying relevant data sources, modeling a relational database, handling data population issues (cleaning and formatting), and performing meaningful SQL queries. The final result is a functional database that links flight operations with weather conditions, capable of supporting complex analysis.