Ministère de l'Education Nationale,
De l'Enseignement Supérieur et de la Recherche
Université de Lille
Département d'informatique

DataBases Project

Done by
Abderaouf KHELFAOUI
Ayman EL ALASS
Supervisor
Sara RIVA

2025-12-08

Content

Table des figures

List of Algorithms

1 Introduction

For the "Databases 1" course project, we selected a dataset that allows for complex relational modeling : flight delays. This topic immediately caught our attention because of its real-world relevance and the potential for interesting analysis. Aviation data is inherently complex, involving multiple interconnected entities like airlines, airports, routes, and schedules, which makes it an excellent candidate for demonstrating relational database principles.

To create a more interesting and challenging database, we combined this flight data with a second dataset containing historical weather information. This decision was motivated by the hypothesis that weather conditions play a significant role in flight operations. By integrating these two datasets, we could explore relationships that wouldn't be apparent from flight data alone, such as whether high wind speeds correlate with increased cancellations or whether temperature extremes affect departure delays.

The objective was to identify raw data, structure it into a relational database using SQL, and then run realistic queries to answer meaningful questions. We wanted our database to serve a practical purpose, not just demonstrate technical skills. The questions we aimed to answer included :

Which airlines have the best punctuality ? This analysis could help travelers make informed decisions when booking flights and could provide insights into operational efficiency differences between carriers.

Which routes suffer the most from delays ? Understanding problematic routes could reveal systemic issues, whether they're related to air traffic congestion, airport infrastructure limitations, or geographic challenges.

Does weather impact flight cancellations and delays ? By correlating meteorological data with flight performance, we hoped to quantify how much weather conditions—such as wind speed, temperature, and precipitation—actually affect airline operations. This type of analysis has practical applications for both airlines planning their schedules and passengers trying to avoid weather-related disruptions.

Throughout the project, we focused on creating a database that wasn't just technically sound but also capable of answering these real-world questions effectively.

2 Data Sourcing

We used two primary data sources downloaded from Kaggle, both offering substantial datasets that required careful processing and integration :

We used two primary data sources downloaded from Kaggle :

(a) **Flight Delays** [1] **:** A large dataset containing flight schedules, delays, and cancellations for the year 2015. This dataset includes information about departure times, delay durations, cancellation status, airlines, and airports. The data comes from the US Department of Transportation and provides a comprehensive view of domestic flight operations.

(b) **Historical Hourly Weather Data** [2] **:** A meteorological dataset containing hourly measurements of temperature and wind speed for major US cities throughout 2015. This data allows us to examine potential correlations between weather conditions and flight performance.

As per the project guidelines, we had to manage the merging of these different sources, handling redundant information and ensuring data consistency. The main challenge was connecting the two datasets meaningfully. The flight data uses airport codes while the weather data uses city names, so we needed to establish the correct correspondences. Additionally, both datasets required cleaning to remove missing or invalid entries before they could be properly integrated into our relational database structure.

1. https://www.kaggle.com/datasets/usdot/flight-delays
2. https://www.kaggle.com/datasets/selfishgene/historical-hourly-weather-data

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.

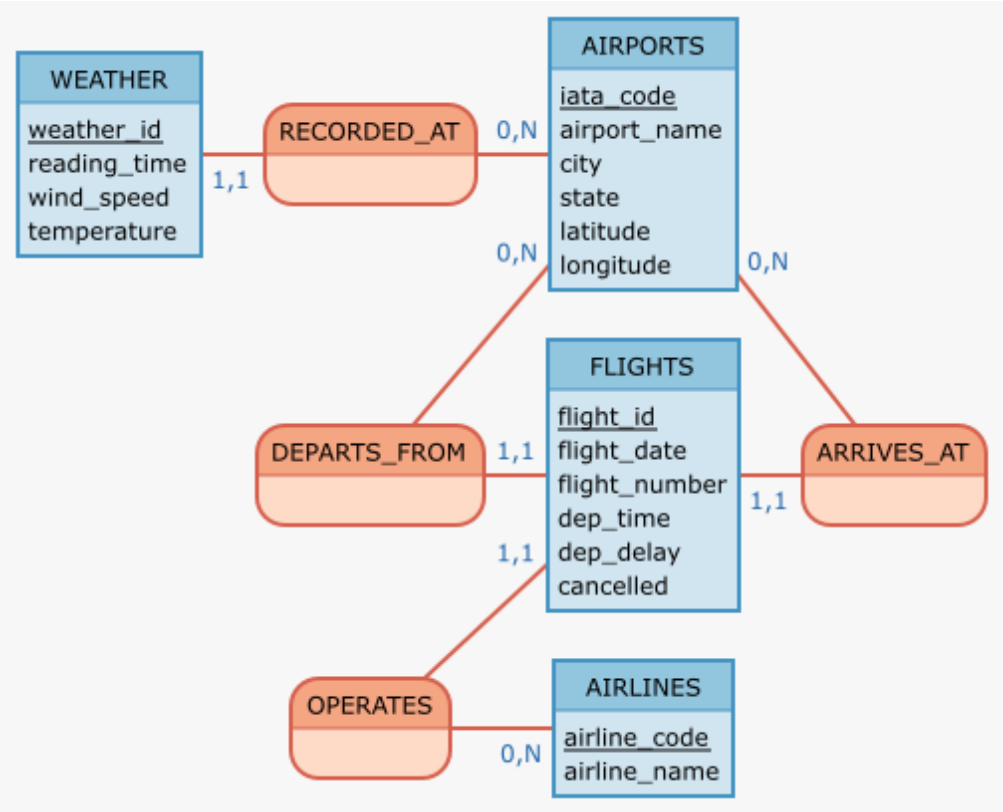

FIGURE 1 – Conceptual Data Model (MCD)

Explanation of the Conceptual Model

Our database is centered around the **FLIGHTS** table, which contains the core information about each flight operation. Each flight record includes details such as the flight date, flight number, departure time, delay duration, and whether the flight was cancelled.

A flight is connected to other entities through several relationships :

Airlines : Each flight is operated by exactly one airline (relationship "OPERATES"). An airline can operate many flights (0,N cardinality), but each individual flight belongs to only one airline (1,1 cardinality). The airline information includes the airline code and name.

Airports : Each flight has two connections to airports. It departs from one origin airport (relationship "DEPARTS_FROM") and arrives at one destination airport (relationship "ARRIVES_AT"). An airport can serve as the origin or destination for many flights (0,N cardinality), while each flight has exactly one origin and one destination (1,1 cardinality). The airport entity stores information like the IATA code, airport name, city, state, and geographic coordinates (latitude and longitude).

Weather : Weather conditions are recorded at specific airports (relationship "RECORDED_AT"). Each weather reading is associated with one airport (1,1 cardinality), and an airport can have multiple weather readings over time (0,N cardinality). The weather data includes the reading time, wind speed, and temperature.

This structure allows us to answer questions like What was the weather like when a flight departed ?" by linking flights to their origin airport and then to the weather conditions recorded at that airport around the departure time.

3.2 Logical Data Model (MLD)

Based on the conceptual framework defined previously, we derived the Logical Data Model (MLD). This step translates the abstract entities into a concrete relational schema, defining tables, Primary Keys (PK), and Foreign Keys (FK).

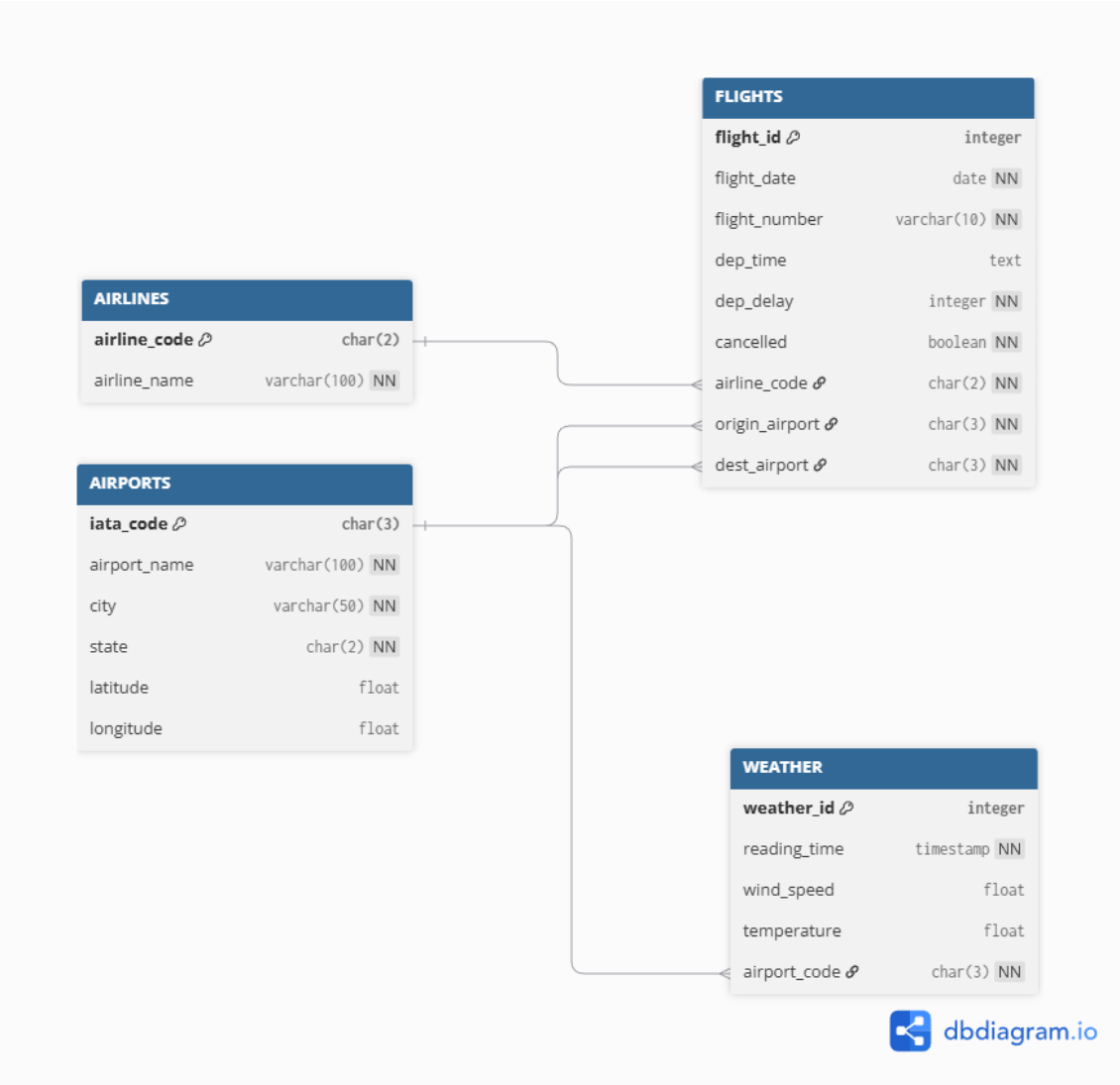

FIGURE 2 – Logical Data Model (MLD)

Relational Schema Translation

As illustrated in Figure 2, the transformation involves converting entities into tables and relationships into keys. The resulting structure is as follows :

— **FLIGHTS** (`flight_id`, `flight_date`, `flight_number`, ..., #`airline_code`, #`origin_airport`, #`dest_airport`)

This table acts as the central fact table. Notably, it contains two foreign keys pointing to the **AIRPORTS** table to distinguish between the origin and the destination of the flight.

— **AIRPORTS** (iata_code, airport_name, city, state, latitude, longitude)
The primary key is the standard 3-letter IATA code (e.g., 'JFK').

— **AIRLINES** (airline_code, airline_name)
The primary key is the standard 2-letter IATA carrier code.

— **WEATHER** (weather_id, reading_time, temperature, wind_speed, #iata_code)
Weather data is linked to airports via the IATA code, maintaining 3rd Normal Form.

Implementation Choices

Specific design decisions were made regarding data types to ensure storage efficiency and standard compliance :

— **Standardized Codes :** We selected `CHAR(2)` and `CHAR(3)` for airline and airport codes respectively. Since these codes are fixed-length international standards, using `CHAR` is more performance-efficient than variable-length strings.

— **Temporal Data :** We utilize standard date and time formats to ensure temporal queries (e.g., calculating delays) can be performed accurately.

— **Boolean Indicators :** The `cancelled` attribute is modeled as a boolean flag, allowing for straightforward filtering of cancelled flights.

Integrity Constraints

The schema enforces strict referential integrity. The **Not Null (NN)** constraints visible in the diagram ensure that every flight is strictly associated with an existing airline and valid origin/destination airports, preventing data inconsistencies.

3.3 Design Choices and Constraints

When designing our SQL schema, we had to make careful decisions about how to structure each table. Here's the reasoning behind our choices for primary keys, data types, and constraints.

Choosing Primary Keys

For the **AIRLINES** table, we used `airline_code` as the primary key. This made sense because IATA airline codes (like "AA" for American Airlines or "DL" for Delta) are already standardized and unique in the aviation industry. Using these natural codes makes our queries more readable than arbitrary numeric IDs.

The **AIRPORTS** table follows the same logic with `iata_code` as the primary key. Airport codes like "JFK" or "LAX" are universally recognized and appear in both of our source datasets, making them the obvious choice for uniquely identifying airports.

For the **FLIGHTS** table, we decided on a surrogate key called `flight_id`. Unlike airlines and airports, there isn't a single natural attribute that uniquely identifies a flight. The same flight number (like "AA100") operates multiple times throughout the year, so we needed a simple auto-incrementing integer to distinguish each individual flight record.

Similarly, the **WEATHER** table uses `weather_id` as a surrogate key. Weather observations are actually identified by both their timestamp and location, which would require a composite key. A single integer key keeps things simpler for joins and queries.

Data Types and Constraints Explained

Let's walk through each table and explain why we chose specific data types and added certain constraints.

For the AIRLINES table :

We used `CHAR(2)` for `airline_code` because IATA codes are always exactly two letters. The airline name is `VARCHAR(100)` with a `NOT NULL` constraint—every airline needs a name, and we allow up to 100 characters for longer company names.

For the AIRPORTS table :

The `iata_code` is `CHAR(3)` since airport codes are always three letters. Airport names and cities are `VARCHAR` fields marked as `NOT NULL` because this information is essential for identifying the airport. The state code is `CHAR(2)` to match US state abbreviations.

For coordinates, we chose `FLOAT` to store `latitude` and `longitude` with decimal precision. We added a `CHECK` constraint to ensure these values stay within valid geographic ranges : latitude must be between -90 and 90 degrees, and longitude between -180 and 180 degrees. This prevents obvious data entry errors.

For the FLIGHTS table :

The `flight_date` is a `DATE` type marked `NOT NULL`—every flight must have a date. The `flight_number` is `VARCHAR(10)` to handle different airline numbering formats.

We stored `dep_time` as `TEXT` because SQLite doesn't have a native TIME type. To ensure the time format is consistent, we added a `CHECK` constraint using pattern matching (`GLOB`) to enforce the "HH :MM :SS" format. The pattern `[0-2][0-9]:[0-5][0-9]:[0-5][0-9]` ensures valid hours, minutes, and seconds.

The `dep_delay` is an `INTEGER NOT NULL` representing delay in minutes. We made it required (even if zero) so we always know the delay status. Negative values indicate the flight left early.

The `cancelled` field is a `BOOLEAN NOT NULL`—we need to explicitly know whether each flight was cancelled or not.

Finally, we have three foreign key constraints : `airline_code`, `origin_airport`, and `dest_airport` must all reference valid entries in their respective tables. This ensures we can't have flights from non-existent airlines or airports.

For the WEATHER table :

The `reading_time` is a `TIMESTAMP NOT NULL` that records exactly when the weather was measured. This is crucial for matching weather conditions with flight times.

Both `wind_speed` and `temperature` are `FLOAT` types. We left these nullable because weather stations occasionally have incomplete readings due to sensor issues.

The `airport_code` is `CHAR(3) NOT NULL` and includes a foreign key constraint to the AIRPORTS table, ensuring every weather reading is linked to a valid airport in our database.

Overall, these design choices help maintain data quality and consistency. The constraints prevent invalid data from being inserted, while the foreign keys ensure that relationships between tables remain intact.

3.4 Schema Evolution

During the project, we refined our model. For instance, we realized that the `AIRPORTS` table needed an `airport_name` field to be more useful. We updated our SQL schema to reflect this.

```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 issues related to data quality, data volume, and format inconsistencies. To automate this process, we implemented two Python functions : `populate_flights_and_others()` for airlines, airports, and flights, and `process_weather_data()` for the meteorological dataset. These scripts ensured a clean, consistent, and memory-efficient insertion into SQL.

4.1 Sampling Strategy & Memory Management

The original flights dataset contains more than 5.8 million rows. To prevent memory exhaustion and maintain a representative sample, we randomly selected **10% of the rows** :

- Sampling is performed over the **entire index range**, ensuring coverage across all dates in 2015.
- This guarantees that our sample includes flights from every season, avoiding bias in time-dependent analysis.
- Only the sampled rows are loaded into memory, which drastically reduces RAM usage.

This strategy produces a dataset that is still large enough to extract meaningful insights while remaining practical on limited hardware.

4.2 Handling Inconsistencies in Flight Data

Several cleaning operations were applied before inserting rows into the `FLIGHTS` table :

- **Time normalization :** The dataset contains invalid values such as `2400` ; these were converted to `23:59:00`.
- **Date reconstruction :** Separate `YEAR`, `MONTH`, and `DAY` columns were merged into a single `flight_date`.
- **Missing delays :** Null `DEPARTURE_DELAY` entries were replaced with 0, assuming the flight departed on time.
- **Referential integrity :** Flights referencing airports not present in the `AIRPORTS` table were removed to avoid foreign key violations.
- **Column filtering :** Only relevant attributes were extracted to reduce memory usage.

Airlines and airports were loaded beforehand, with redundant or unused columns dropped to streamline the schema.

4.3 Cleaning and Structuring the Weather Data

The weather dataset required substantially more transformation, as it was originally provided in a **wide format** (each city in its own column). The following processing steps were applied :

- **Wide-to-long transformation :** Using `pandas.melt()`, wind and temperature tables were converted to long format, enabling relational storage.
- **City-to-airport mapping :** Weather records contained city names rather than airport codes. We created a mapping dictionary assigning each city to its main airport (e.g., `New York → JFK`).
- **Temperature correction :** Temperatures were converted from Kelvin to Celsius.
- **Outlier removal :** Unrealistic sensor readings (temperatures below $-60°C$ or above $60°C$) were removed.
- **Wind speed validation :** Negative wind speeds were discarded.
- **Duplicate filtering :** Rows with the same city and timestamp were removed.
- **Referential integrity :** Only weather rows matching existing IATA airport codes were inserted.

After processing, the cleaned weather dataset was loaded into the `WEATHER` table.

4.4 Summary of Data Cleaning

Dataset	Issue Identified	Solution Applied
Flights	Dataset too large	Sampled 10% uniformly across all dates
Flights	Time format "2400"	Converted to "23 :59 :00"
Flights	Missing delay values	Replaced with "0"
Flights	Unknown airports	Invalid rows removed
Weather	Wide format tables	Reshaped to long format using melt()
Weather	Temperature in Kelvin	Converted to Celsius
Weather	Extreme sensor values	Removed values $< -60°C$ or $> 60°C$
Weather	Wind speed errors	Negative values removed
Weather	City names instead of IATA codes	Mapped to airport codes using a dictionary
Weather	Duplicate rows	Removed based on city + timestamp

TABLE 1 – Summary of Data Cleaning and Preprocessing Steps

Overall, these operations ensured a coherent and reliable dataset, aligned with our relational schema and suitable for meaningful SQL analysis.

5 SQL-Based Analysis of the Airline Database

Once the database was fully populated and cleaned, we conducted a series of analytical SQL queries to evaluate the quality and richness of the dataset. These queries were executed through a Python script that connects to our SQLite database and retrieves the results as dataframes for inspection. The objective was to explore the data from different angles : airline performance, route delays, weather impact, airport traffic, and temporal trends across the year 2015.

5.1 Sanity Checks and First Insights

We began the analysis by performing simple retrieval queries to verify the integrity of the imported data. For example, we extracted the most extreme delays for flights operated by *American Airlines Inc.*, limited to delays exceeding 240 minutes. This served two purposes : (1) validating that delays were stored correctly after cleaning, and (2) ensuring that joins between the `FLIGHTS` and `AIRLINES` tables were functioning as expected. These initial tests confirmed that our transformation rules—for example the normalization of time fields, and the replacement of missing delay values—were correctly applied.

5.2 Airline Performance Assessment

A more elaborate set of queries was then designed to evaluate airline reliability. Three key performance indicators (KPIs) were computed for each major airline (more than 500 recorded flights) :

— **Volume** : total number of flights.
— **Punctuality** : average departure delay.
— **Reliability** : proportion of cancelled flights.

This query required a group-by aggregation over the `FLIGHTS` table joined with the `AIRLINES` table. The results allowed us to rank airlines based on punctuality and identify carriers with a significantly higher cancellation rate. This analysis also helped confirm that our cleaning step for cancellation fields (`0/1` boolean normalization) was correctly applied.

5.3 Route Delay Analysis Using Double Joins

To investigate delays at the route level, we performed a double join on the `AIRPORTS` table—one join for the origin airport and one for the destination. This enabled us to translate IATA codes into human-readable city names and compute metrics such as :

— average delay per city-to-city route,

— maximum observed delay on each route,
— number of flights per route (to filter out rare or anomalous cases).

We restricted the output to routes with at least 20 flights to avoid statistical noise. This query revealed the connections most affected by systematic delays and highlighted geographical patterns in the dataset.

5.4 Weather–Delay Correlation Study

To integrate the `WEATHER` table into our analysis, we computed a simple correlation example focusing on a specific day : January 1st, 2015. Using two Common Table Expressions (CTEs), we computed :

(a) the average wind speed at each airport on that day,
(b) the average departure delay of all flights departing from that same airport.

By joining these two CTEs, we produced a per-airport summary showing potential associations between wind conditions and flight delays. This step also served to validate that our earlier weather cleaning rules (e.g., Kelvin-to-Celsius conversion, removal of sensor errors, and mapping of city names to IATA codes) were correctly applied.

5.5 Schema Evolution Through SQL

To prepare for further analysis, we modified the schema of the `FLIGHTS` table by adding a new column `delay_category`. Instead of recalculating delay categories during every query, we now persist them directly in the table. After altering the schema, we ran an update statement classifying each flight into four categories : “On Time / Early”, “Small Delay”, “Medium Delay”, and “Major Delay”. A verification query showed the distribution of these categories across the entire year, confirming that the update was successfully applied.

5.6 Global Statistics and Temporal Patterns

We then computed several global indicators :

— total number of recorded flights in 2015,
— the busiest day of the year,
— the most active origin airport,
— for each U.S. state, the peak-traffic day and airport.

The last query relied on a window function (`RANK() OVER (...)`) to select the maximum per state without subqueries inside subqueries. This allowed us to extract the most significant traffic day for each state in an efficient and readable way.

5.7 Delay and Cancellation Hotspots

Finally, we performed a deeper look at the relationship between airlines, airports, and delays. We measured the highest average departure delays per (airline, airport) pair, followed by identifying the three airports with the most cancellations. These analyses enabled us to highlight operational bottlenecks and airports with recurrent disruption patterns.

Overall, this set of SQL queries provided a rich and multi-dimensional analysis of the U.S. flight operations in 2015, demonstrating both the quality of our cleaned datasets and the analytical strengths of a well-structured relational schema.

6 Conclusion

Going through this project was not only a hands-on exercise with real-world data ; it also taught us how to think like true data practitioners. Working with raw, imperfect datasets forced us to understand the importance of exploring the data we are given, questioning its quality, and choosing the right strategies to handle inconsistencies. Every issue whether missing values, inconsistent formats, or structural errors became an opportunity to strengthen our ability to reason about data rather than just process it. This project also gave us the chance to experience the complete lifecycle of relational database development. From selecting and inspecting datasets, to building a conceptual schema, translating it into a logical model, and finally implementing it in SQL, each phase deepened our understanding of how databases are actually built and used. We learned how to integrate heterogeneous data sources, enforce constraints, and structure our tables in a way that supports efficient querying. More importantly, writing and executing advanced SQL queries allowed us to connect the abstract notions of relational algebra to practical analytical tasks. Whether examining delays, identifying performance patterns, or building summary statistics, we saw how a well-designed schema enables powerful insights with relatively simple queries.

By the end, we produced a functional, clean, and scalable relational database capable of supporting complex analysis across flights, airlines, airports, and weather conditions. Beyond the technical implementation, the project helped us appreciate the mindset required when interacting with the world's data : be curious, be critical, and always understand why the data looks the way it does.