Data Warehousing Project

Design decisions and assumptions for flight descriptive analytics data warehousing project

Jordi Corbalan Vilaplana, Ona Siscart Noguer, Team 12G

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

Subject: Advanced Databases, BDA-GCED
Professor: Albert Abello
Year: 2025-2026

1 Multidimensional Design

The followung schema supports all required query types (a–d), allowing analysis by aircraft, model, manufacturer, and airport at daily, monthly, and yearly granularity.

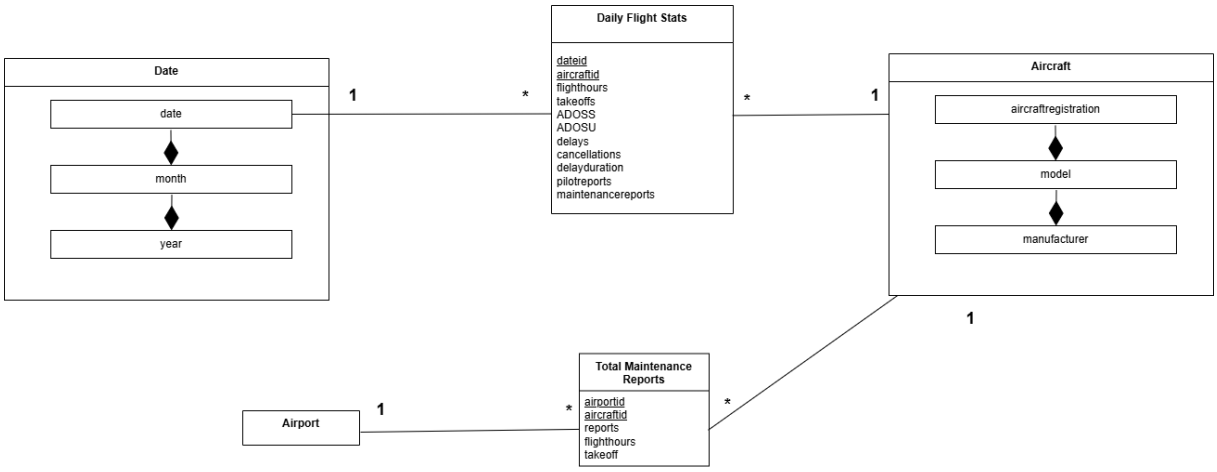

Figure 1: DW Multidimensional Schema

The relevant decisions made to achieve this design are summarized below:

- **Date**: We store most metrics at the date level even though some are only required at month level, as using a single time table results in flexibility and simpler queries. Splitting into separate date and month tables would require costly joins and slow response times. Since table growth is manageable, this design keeps the schema simple, scalable, and fast. Additionally, storing data at the daily level would allow to capture extra attributes such as day of the week, holidays, and other temporal characteristics, which can enhance future analytical queries.

- **Aircraft dimension:** we store all aggregation levels in one same table, as we do not expect the model or manufacturer of aircrafts to change and it saves costs in space (we only store existing combinations) and time (joins).

- **Airports:** As target queries are airport based and not worker based, we decided to store airports as a dimension. Although there are no atttributes to this dimension other than the code, we do not consider it a degenerate dimension and keep it as a table. This is because we expect that more information regarding the airport might be collected and stored in the future, thus keeping our data warehouse scalable. Even a new table about Airport - Maint personnel table may be created, if needed in future enhancements even it is not included in the scope of our queries.

- **Daily Flight Stats:** We decided to gather all statistics in a single fact table. The alternative of using separate fact tables according to the nature of the statistic (flight, maintenance) was considered, but ultimately discarded, as queries combining statistics from multiple sources would require a large number of unnecessary joins.

  The lowest granularity for flight data was chosen to be daily, as this matches the minimum granularity required by our queries (a). Although most queries are aggregated by month (b,c) or total(d), maintaining day-level data increases the scalability and flexibility of the data warehouse, allowing for a wider variety of future analyses, while still ensuring efficient execution for our four primary objective queries.

  We store only the minimum necessary aggregates to calculate the KPIs for query types a, b, and c. We do not store rates (KPIs) directly as it would complicate the updating of the data warehouse

and removes analysis flexibility, which contradicts the purpose of a data warehouse. We store number of reports in 2 attributes separated by role instead of adding a "role dimension", which would complicate the schema unnecessarily and add a degenerated dimension.

- **Total Maintenance Reports:** The fact table stores HISTORICAL TOTALS until the last update of maintenance reports, takeoffs and flight hours. This is in order to efficiently answer to query type d). We considered calculating the totals inside the queries, as aggregates in The Daily Aircraft stats table. But this would mean adding the "airport" dimension in the Daily flight stats fact table. This would increase the size of the table by a large factor. Moreover, is unnecessary for the current specified needs, as temporal data is never analyzed by airport. Table 1 presents a comparison between the chosen alternative (implementation 1, with Total Maintenance reports and Daily Flight Stats) and the alternative of adding airport as a dimension of our daily flight stats data

Table 1: Comparison of sizes and structure between implementations

Component	**Implementation 1 (two tables)**	**Implementation 2 (one table)**
Daily Flight Stats	6.02×10^6 (rows $= 5 \times 365 \times 300$, cols $= 11$)	1.45×10^9 (rows $= 5 \times 240 \times 300 \times 365$, cols $= 11$)
Total Maint. Reports	3.60×10^5 (rows $= 240 \times 300$, cols $= 5$)	-
Total size	6.38×10^6 ($6.02 \times 10^6 + 3.60 \times 10^5$)	1.45×10^9

Note: The numbers in the table are calculated assuming DW contains data over 5 years, $\approx$ 240 airports, $\approx$ 300 airplanes. The implementation with two tables occupies approximately 226 times less space. In the *Totals* table, we store again *Takeoffs* and *Flight Hours* (this time, total aggregates) to avoid having to perform a join for query (d); this redundancy is justified to optimize query performance.

2 Extract-Transform-Load (ETL) Process Design

The ETL was designed using the pygrametl Python library, ensuring modular and parallelizable data extraction and transformation.

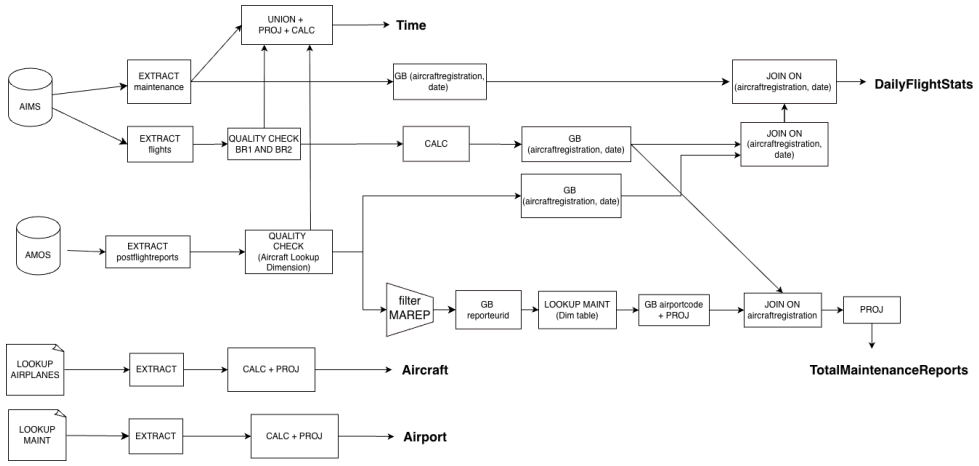

Figure 2: ETL Data Flow

- **Extract**: No transformations were applied upon extraction, raw data was collected and passed to respective transform functions. All sources were extracted as their respective stream (Csv lookups as CSVSource, SQL Queries as SQLSource). We do this in order to make operations as non-blocking as possible. In sources that accepted it (SQL queries), we made projections upon extraction, keeping

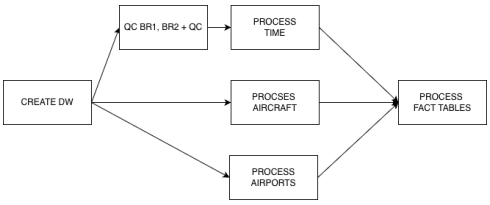

Figure 3: ETL Control Flow

only necessary attributes. As error handling, the ETL process is stopped if any extraction fails, as we would not be able to build the database without data.

- **Transform**: Quality checks are done as soon as they are possible, as we do not want to carry data that we will eventually discard. During the flights quality check, flights data is transformed to a Dataframe, as BR-21 (slots overlaps) includes a blocking operation. However, BR-22(swapped times) and QualityCheck regarding the existence of airplanes of postflightreports in our lookup are both non-blocking.

  It is important to remark that the dates included in the *Time* dimension are the unique date values included in the union of dates in all used tables. However, only the ones belonging to the years that appear in the flights table (as this is how they were selected in the baseline example query)

  After cleaning when needed, we derive attributes, perform filters and change bases when needed. We aggregate data to avoid having too many tuples at an unnecessary granularity. We perform joins after reducing size of tables to the maximum and aggregating as much as possible to reduce the number of comparisons as much as possible. Joins to obtain *Dailyaircraft* stats are performed to join the tables with least number of rows first, and use only years contained in "AIMS.flights" in order to match the behavior of baseline queries. Joins to reach *TotalMaintenanceReports* are used to first obtain the airport for each reporteurID and then to obtain total aggregates for "takeoffs" and "flighthours".

- **Load**: Once again, we stop the entire process in case of any error when loading data into the warehouse.

- **General Dependencies (Control flow):** Aircraft and airports dimensions have no dependencies and their are loaded as soon as possible. The time dimension can be created just after the data is cleaned. The data cleaning process is done only once, as it precedes both Time dimension creation and facts table creation. Once all dimensions are inserted, the fact tables can then be created.

3 Comparison of queries over Data Warehouse and sources

Query	**DW (s)**	**Baseline (s)**
Utilization	0.0188	0.9689
Reporting	0.0104	0.4253
Reporting per role	0.0100	0.3818

Table 2: Execution time for each query (Baseline vs DW).

As shown in Table 2, the queries executed on the Data Warehouse are roughly 40 - 50 times faster than the corresponding queries on the source systems. This significant improvement highlights how the DW design, with optimized fact and dimension tables, surrogate indexes, and pre-aggregated data, enables much more efficient querying and rapid analytical insights.

In addition, all query results were identical between the DW and the raw sources (before implementing BRs and QualityChecks), confirming the correctness of the ETL process and the suitability of our design.