✈️ Airline Business Intelligence Database

Final Project Overview Document

Grace Polito – MSDS Capstone Project

Eastern University • December 1, 2025

---

1. Introduction & Problem Context

The Airline Business Intelligence Database project was designed to simulate the end-to-end data ecosystem of a mid-size airline, from raw operational data capture through analytical reporting layers used for decision-making. Real airline organizations rely on large, fragmented datasets—airports, routes, flight schedules, delays, bookings, payments, passengers, loyalty programs, and revenue streams—each often maintained in separate operational systems. As a result, critical business questions involving performance, reliability, and customer value typically require integration of multiple sources, extensive cleansing, and development of analytics-ready structures.

This capstone project developed a complete, end-to-end Business Intelligence environment for airline operations. I built the system from scratch using PostgreSQL, Python ETL pipelines, custom SQL analytics, and a Python-based BI layer.

The project integrates real-world reference data (OpenFlights, U.S. BTS On-Time Performance) with several synthetic datasets I generated to model flights, passengers, bookings, payments, loyalty accounts, and revenue.

My goal was to simulate what a real airline BI team would produce:

- a clean, constrained database
- validated and standardized data
- analytical queries for operations, network, revenue, and loyalty
- visual insights generated through Python
- documentation and reproducibility across all phases.

This document summarizes work completed across all six phases of development—from schema design through Python analytics and final presentation deliverables, including the challenges and technical decisions involved.

---

2. Data Sources & Business Domain

2.1 Real Data Imported

OpenFlights

- I loaded airports.dat and airlines.dat after manually reviewing column definitions.
- Cleaned inconsistent IATA/ICAO codes and replaced \N with NULL before insertion.

U.S. BTS On-Time Performance

- I ingested 22,595 rows from a CSV containing arrival delays, carrier delays, weather delays, and cancellation/diversion indicators.
- I manually aligned column types with PostgreSQL tables (especially timestamps and integers).
- I standardized all numeric delay columns because the raw file contains blanks and mixed string representations.

2.2 Synthetic Data I Generated

Generated using Python (faker, NumPy, SQLAlchemy):

- **Flights**: Generated schedules, delays, departure/arrival timestamps, and statuses (Departed, Landed, Cancelled, etc.).
- **Passengers**: Names, DOB, emails, phone numbers, addresses.
- **Loyalty Accounts**: Random tier assignments, miles earned, join dates.
- **Miles Transactions**: Credits/debits that align with loyalty logic.
- **Bookings**: Ensured 1 booking per (passenger_id, flight_id) combination.
- **Payments**: Attached 1:1 with bookings with realistic timestamps and amounts.

This data includes:

- 5,000 flights
- 5,000 passengers
- 3,000 loyalty accounts
- 10,576 miles transactions
- 40,000 bookings

Why some dates extend into the future

This is a design choice, not an accident.
My synthetic data required:

- Full 12–24 month seasonality for trend charts
- A consistent monthly timeline with no gaps

- Enough variation in booking and payment dates to support BI-style time series
- Flexibility for recursive route queries that might span periods
- This is common in synthetic BI environments, and allows better analysis without affecting referential integrity.

The database combines OpenFlights datasets for airline and airport reference information, U.S. BTS On-Time Performance data for real arrival delays, and synthetic data generated using Python to model flights, bookings, payments, and loyalty behavior. Together, these sources create a realistic operational and commercial landscape for BI analysis. The business domain reflects real airline decision-making needs across operations, network planning, revenue management, and loyalty programs.

---

3. Phase 1 - Schema Design & ERD

Phase 1 established the relational backbone of the warehouse.

What I Built

Designed all tables manually in PostgreSQL with strong normalization:

- airports
- airlines
- aircraft
- routes
- flights
- passengers
- loyalty_accounts
- miles_transactions
- bookings
- payments
- flight_performance

Important Technical Choices

- Created ENUM types for controlled values (flight_status, fare_class, loyalty_tier, payment_channel).
- Added CHECK constraints for:
    - logical timestamps (arrival > departure)
    - positive mileage and payments
    - delay fields

- Ensured all FK relationships cascade logically (restrict vs cascade).
- Created indexes aligned to BI workloads:
  - flights(flight_date, airline_id)
  - bookings(booking_date)
  - payments(paid_at)
  - flight_performance(airport_iata)

Validation Work

- Loaded schema through 01_schema.sql
- Inspected structure in pgAdmin
- Exported ERD (ERD_v1.pdf, later updated to ERDv2.pdf)

The schema was intentionally designed using 3NF relational modeling to ensure clean joins, data quality, and analytical flexibility. Major entities include airports, airlines, routes, flights, passengers, bookings, payments, and loyalty accounts, with strict PK/FK relationships enforcing referential integrity. The ERD demonstrates a complete operational-to-commercial data flow suitable for BI workloads.

---

4. Phase 2 - ETL Pipeline Development & Data Ingestion

Phase 2 operationalized all real and synthetic data using structured ETL pipelines.

Scripts I Wrote

- load_openflights.py
  - Parsed OpenFlights files
  - Cleaned \N values
  - Normalized IATA/ICAO codes
  - Inserted airports and airlines
- load_bts_performance.py
  - Loaded raw BTS on-time performance
  - Converted numeric fields
  - Converted timestamps to UTC
  - Ensured missing values became NULL
- synth_flights.py
  - Generated flight dates up to 2026
  - Randomized departure/arrival times
  - Calculated delay columns
  - Generated realistic statuses
  - Ensured all flights map to valid airports & airlines

- synth_customers.py and synth_revenue.py
    - Created passenger demographics
    - Generated loyalty accounts
    - Created bookings → payments pipeline

What I Validated

- Row counts by table (captured in pipeline_row_counts.png)
    - Flights: 5,000
    - Passengers: 5,000
    - Loyalty Accounts: 3,000
    - Miles Transactions: 10,576
    - Bookings: 40,000
    - Payments: 40,000
- Foreign key health:
    - 0 missing references for every FK relationship
- Manual spot checks of synthetic data distributions (e.g., average fare amount, delay distribution, miles earned)

Data ingestion pipelines were built using Python and SQLAlchemy to load OpenFlights and BTS datasets and generate synthetic flights, revenue, and customer activity. Each ETL script performs validation and transformation to align formats, normalize codes, and maintain key relationships. This phase produced a fully populated database with consistent, realistic data across all domains.

---

5. Phase 3 - Data Cleaning, Standardization & Constraints

Phase 3 transformed raw ingested data into a clean, standardized, constrained warehouse.

What I Actually Cleaned

- Standardized airline/airport codes (uppercase).
- Normalized inconsistent emails and removed trailing spaces.
- Converted blank strings in BTS data.
- Cleaned flight timestamps so:
    - scheduled times
    - actual times
    - follow a logical order and pass CHECK constraints.

Deduplication Logic

- Unique airport and airline codes enforced.
- Removed synthetic duplicates where random generation overlapped.
- Added constraints to ensure:
  - 1 loyalty account per passenger
  - 1 booking per (passenger_id, flight_id)

Constraints & Indexing

- Added strict PK/FK definitions
- Enforced UNIQUE constraints
  CHECK constraints on:
  - Latitude/longitude
  - Currency value
  - Delay logic
- BI-optimized indexes:
  - flights(flight_date, airline_id)
  - bookings(booking_date)
  - payments(paid_at)

Quality Assurance

I used 02_data_quality_checks.ipynb to:

- Validate all constraints
- Check null patterns and duplicate
- Generate the visual quality proof (pipeline_quality_checks.png)

Phase 3 focused on enforcing strict data quality through DML cleanup scripts, normalization of codes, deduplication rules, and comprehensive NOT NULL, CHECK, and UNIQUE constraints. Additional indexing ensured efficient analytical queries across large joins and date-based filters. All corrections were validated through quality checks and a profiling notebook, confirming clean, reliable data for downstream analytics.

---

6. Phase 4 - Advanced Analytical SQL Development

Phase 4 produced the analytical intelligence layer—15 advanced SQL queries powering operational, commercial, and loyalty insights.

What I Implemented

15 advanced analytical queries using:

- **CTEs:** busiest airports, on-time performance, monthly passengers, fare-class revenue
- **Window Functions:** delay ranking, cumulative revenue, CLV scoring, percent delayed
- **Recursive Queries:** airport connectivity graph
- **Aggregations:** payment success, worst routes, top loyalty members

Examples of Queries I Designed:

- Busiest airports across arrivals + departures
- Airline-level delay scores using BTS metrics
- Customer lifetime value (window function over payments)
- Payment channel success rates
- Worst-performing routes (delay + cancellation composite score)
- Multi-hop route exploration (recursive)
- Network connectivity starting from busiest hub

Performance Tuning

I tested every query with:

- EXPLAIN
- EXPLAIN ANALYZE
- index usage checks
- timing analysis

If a query didn't use an index, I refactored it or added appropriate indexing.

Artifacts

- Full query catalog (phase_4_query_catalog.md)
- Notebook with all SQL and outputs (03_analytics_queries.ipynb)
- Analytical figures included (phase_4_analytics.png)

Fifteen advanced SQL queries were developed using CTEs, window functions, recursive CTEs, aggregations, and multi-table joins to answer key airline business questions. Performance tuning with EXPLAIN ANALYZE ensured that indexes and execution plans supported efficient analytics. The output includes metrics on delays, busiest airports, route performance, loyalty value, revenue distribution, and payment channel success.

---

7. Phase 5 - Python Integration, Analytics & BI Visualizations

Phase 5 operationalized SQL insights into Python for visual exploration and BI storytelling.

Connection Layer

I built:

- get_engine() – SQLAlchemy engine using .env
- get_df() – wrapper for pd.read_sql()
- Global plot styling (white background, larger fonts, rotation settings)

Python Analytical Functions

I created Python wrappers for all Phase 4 SQL queries so they can be used like an API:

- get_busiest_airports()
- get_monthly_revenue()
- get_airline_punctuality()
- get_clv_distribution()
- get_payment_success_by_channel()
- get_worst_routes()
- get_network_connectivity()

Visualizations Produced

I generated over 10 BI-style visualizations, exported to /docs/, including:

- Monthly Revenue Trend
- Revenue by Fare Class
- Delay Distribution
- Airline Delay Ranking
- Flights Delayed by Month
- Payment Success Rate
- CLV Distribution
- Top 10 Loyalty Customers
- Airport Network Map
- Busiest Routes Sankey (Plotly limitations documented)

Troubleshooting

- I fixed issues with Plotly's missing px.sankey() by switching to go.Sankey().
- I adjusted timestamps after CHECK constraint failures (caused by date shifting scripts).
- I regenerated some synthetic flights that failed logical time validations.

The final Python notebook is 04_python_analytics.ipynb.

A dedicated Python notebook established the SQLAlchemy connection layer and wrapped analytical SQL in reusable helper functions. Pandas DataFrames and Matplotlib/Plotly visualizations were used to analyze trends such as revenue over time, delay patterns, fare-class performance, CLV distribution, and payment success rates. This phase translated raw SQL outputs into BI-ready storytelling and visual interpretation.

---

8. Key Business Insights

Across phases 4 and 5, several actionable insights emerged:

Operations

- Delay rates fluctuate seasonally (peaks in March/December)
- Certain routes exhibit extreme delay and cancellation behavior
- BTS data reveals meaningful variation in delay causes

Network

- Many busiest airports are remote, low-volume nodes—indicating synthetic distribution vs real-world hubs.
- Recursive connectivity charts illustrate fragmented route networks

Revenue

- Revenue is dominated by Basic, Standard, and Flexible fare classes.
- Monthly revenue exhibits stable trends with peaks in mid-year months.

Payments

- Payment success rates are low (~15% across all channels), revealing opportunity for commercial optimization.

Loyalty

- Top 5% of customers contribute disproportionally to CLV.
- Loyalty tiers may not align with earned miles (possible tier inflation).

Analysis revealed seasonal delay trends, revenue concentration by fare class, significant differences in payment success by channel, and strong skew toward high-value loyalty members. Network analysis identified low-volume but high-delay routes and highlighted potential operational bottlenecks. Together, these findings inform opportunities for route optimization, payment process improvement, targeted loyalty marketing, and holistic performance monitoring.

---

9. Challenges, Assumptions & Limitations

Major Technical Challenges

- Fixing future-date CHECK constraint issues
- Regenerating flights after constraint failures
- Coordinating LARGE synthetic datasets with correct FK alignment
- Getting Python + SQLAlchemy + Plotly running reliably in VS Code
- Making route and delay logic realistic enough for BI analysis

Dataset Limitations

- Synthetic route networks don’t resemble real airline networks
- Payment success is artificially low
- Synthetic revenue has limited variance

Assumptions

- Time-series synthetic data needs future values for balanced trends
- Loyalty tiers loosely follow real airline patterns
- BTS data is representative despite small sample size

Major challenges included aligning real BTS performance data with synthetic schedules, maintaining referential integrity during synthetic generation, and ensuring repeatable ETL pipelines. Assumptions were required regarding revenue amounts, loyalty behaviors, and aircraft assignments due to the synthetic nature of parts of the dataset. While the environment is realistic, it remains a scaled-down model of full airline enterprise systems.

---

10. Future Work & Enhancements

Potential expansions:

Modeling

- Advanced delay prediction using gradient boosting
- Market demand forecasting
- Customer churn modeling

Engineering

- Materialized views for BI workloads
- Airflow orchestration for repeatable ETL
- Docker containerization for deployment

Analytics

- Tableau / Power BI dashboards
- Real-time monitoring via streaming ingestion
- Geo-visualization of global flight patterns

Future enhancements could include materialized views for faster BI refresh, a dashboard layer (Tableau or Power BI), predictive modeling for delay risk, and expansion of real-world data integration. Additional features such as passenger segmentation, dynamic pricing simulation, or operational forecasting would further enrich the analytical capability. The project provides a strong foundation for scaling into a full analytics platform.

---

11. Conclusion

This capstone project demonstrates a complete BI system lifecycle:

- Well-designed relational schema
- Custom Python ETL pipelines
- Clean, standardized, validated dataset
- Advanced analytical SQL library
- BI-ready Python visualizations
- Extensive documentation and reproducibility
- Final integration into a structured portfolio-ready repository

The resulting Airline BI Database is a robust, scalable, analytics-ready environment suitable for operational BI dashboards, commercial insights, and advanced analytics.