Airline Database Project

Colden Johnson, Michael Cornell, and Ethan Deal

Duke Kunshan University, China

Abstract. Designed and developed an airline database to track key flight information as an internal tool, using real-world data scraped from Google Flights and public repositories. Our system includes real-time queryable flight data, layover calculation, salary data views, path finding, and crew assignment queries. The project was developed using a mix of MySQL and Python on the backend, combined with a terminal frontend where queries can be directly run on the database. This report is intended to outline the problem specification, design methodology, technical solutions, and results/insights from the project completion, with a focus on application and future opportunities for improvement.

1 Introduction

1.1 Background and Motivation

The aviation industry is not known for having the most technologically advanced or user-friendly systems. However, at the same time, databases are crucial to the work and scheduling assignments they do. These databases includes flight schedules, passenger information, crew data, weather conditions, maintenance logs, and technical plane details. To manage this data effectively is to improve efficiency and profit, and to fail to do so could hamstring an airline. The motivation for this project is Delta's slow recovery from the CrowdStrike software update. While other airlines resumed normal operations (albeit at a slow pace), Delta's inability to determine where their flight crews were slowed their recovery, to the tune of $500 million dollars. In our database system, we attempt to keep track of crew, passengers, planes, and other information potentially saving millions of dollars by allowing rapid access to critical information.

1.2 Project Vision

The Airline Database is intended to accomplish a set of key goals:

1. Develop a relational database that encompasses all data needed for the successful operation of an airline, following proper database design practices (as outlined by the ER Diagram and Relational Diagram).

2. Write views, functions, and procedures to assist with a variety of tasks, including layover calculations, salary data views, path finding, and crew assignments.

3. Enable real-time manual flight querying based on Google Flights and other online sources, that can then dynamically incorporated into the database.

4. Follow strong design principles across the entire database.

The objective of this project is to design a relational database tracking airline flight information. This will be designed to act as a useful internal tracking tool for carriers. We will populate the database using information scraped from Google Flights. We will also employ the Beautiful Soup Python package to assist with web scraping. The final product supports storage, retrieval, and reporting of real-time analytics when manually queried. Given the problems that many airlines have with antiquated database management systems, poor data hygiene, and a decrepit frontend, part of the objective is to identify future areas of improvement for these features. For this reason, we will prioritize creating a fairly resilient backend system. One of the big issues during the CrowdStrike outage was with assigning crews to planes; we will allow the ability/function to assign pilots and flight crew to different routes. Finally, we will work on minimizing errors and making a robust backend that can handle the (necessarily imperfect) scraped data input without breaking.

1.3 System Technology

The development of the database relied on several different technical frameworks for development and implementation. The primary database capabilities were built using MySQL, as a both efficient and flexible relational database management system. Choosing MySQL for the backend allows for the execution of advanced database queries. The database schema is designed to minimize redundancy and enforce data integrity (using the DDL to enforce good data practices), and leverages primary and foreign keys to define clear relationships between all entities. To support MySQL queries and to enter queries from the command terminal, we use Python as the primary scripting language for managing database interactions. In addition to managing user interactions, Python was also used for all aspects of the project involving data scraping. Using libraries like Playwright and Beautiful Soup, we were able to integrate real-world data scraped from Google Flights and public repositories into the database. By pairing Python with MySQL, we were able to fully connect between the database and computational logic, allowing for user interactions and real-time updating.

1.4 ER Diagram and Relational Diagram

In the ER diagram and relational diagram, we formally specify the database design to address the project use case. The ER (enity-relationship) diagram outlines the relational structure of the airline database, looking at key value entities, their attributes, and the relationships between them. The diagram (fig. 1) looks

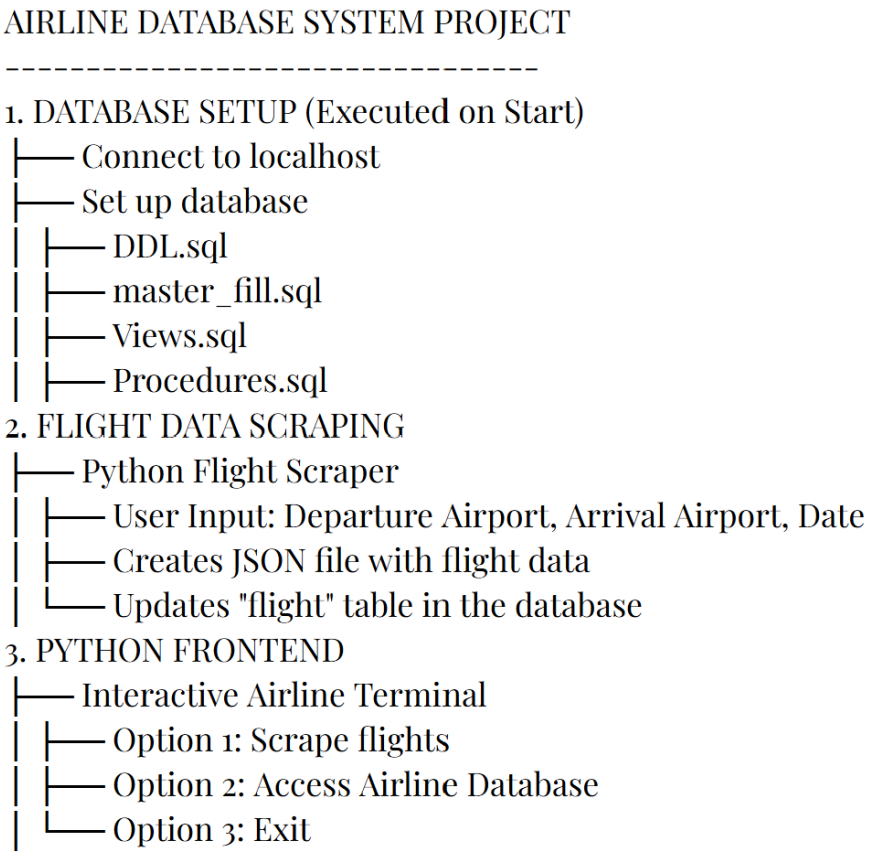


Fig. 1. Airline Database System Structure

at how our relations (flights, crew, passengers, aircraft, bookings, etc.) are interlinked. Each entity (rectangle) connects between a relevant attribute (flight schedule, crew, etc.), while relationships (diamonds) look at the connections between entities (such as 'assigned', etc.).

Some specific areas of interest are the 'bookings' relation. This entity has total participation in both flight and passenger, and is also a weak entity set in these relations. This means that booking lacks a primary key (does not have a unique 'bookingid'), and is dependent on the combination of those two strong entities for identification. Also of interest is the international() calculated attribute (updated via trigger). This attribute is updated using a trigger, and records a value of '1' if the arrival and departure airports are in different countries, and a '0' otherwise. This attribute does not require manual entry, but rather automatically updates depending on the departure and arrival airport values. It is important to note the large number of entities that have complete many to one participation – these include aircraft to airline, maintenance to aircraft, airport to country, etc. Some of the very few entities that do not have many to one participation is crew and flight (where one crew member can be assigned to many flights, and likewise one flight can have many crew assigned). Flight and airport, interestingly, is a many:2 relation, as every flight will be assigned to 2 distinct airports (departure and arrival).

The relational diagram is then derived directly from the ER diagram, translating it into a relational schema that specifies how the data will be organized in tables. Each entity becomes a complete table, with primary keys ensuring the uniqueness of records and foreign keys enforcing relationships between tables. For instance, the 'flight' table includes attributes such as flight ID, departure and arrival times, and is linked to the "aircraft" and "crew" tables through foreign keys. This schema then ensures data integrity across the database.

2 Methods/System

The methodology is the plan of action or strategy with which the outcomes have been reached. This chapter will cover those methods and outcomes.

2.1 Requirements Elicitation and Process

Data Scraping Process To conduct web scraping of our airline database we leveraged playwright, an automation framework, to extract real-world, real-time flight data from Google Flights. This component interacted with the website's interface programmatically, mimicking user actions to navigate the platform, filter for relevant results, and finally to retrieve relevant information. By automating this process we were able to embed this scraper as an element of the final project, and avoid having to do manual data entry.

User Interaction Mimicry The scraper simulates the behavior of a user searching on Google Flights, taking in an input array of a) departure airport, b) arrival

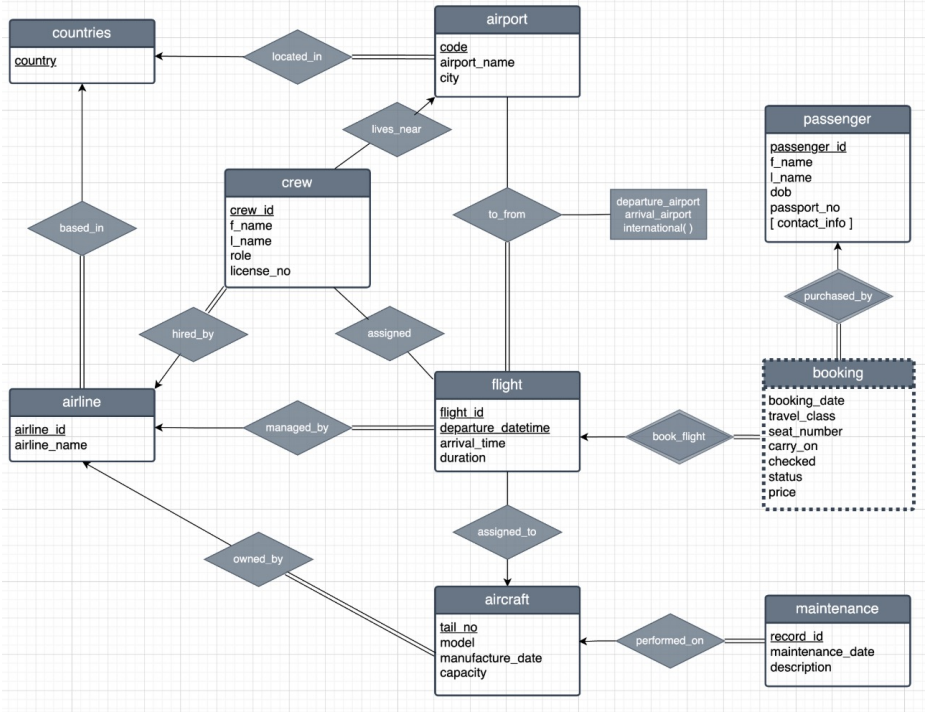


Fig. 2. ER Diagram

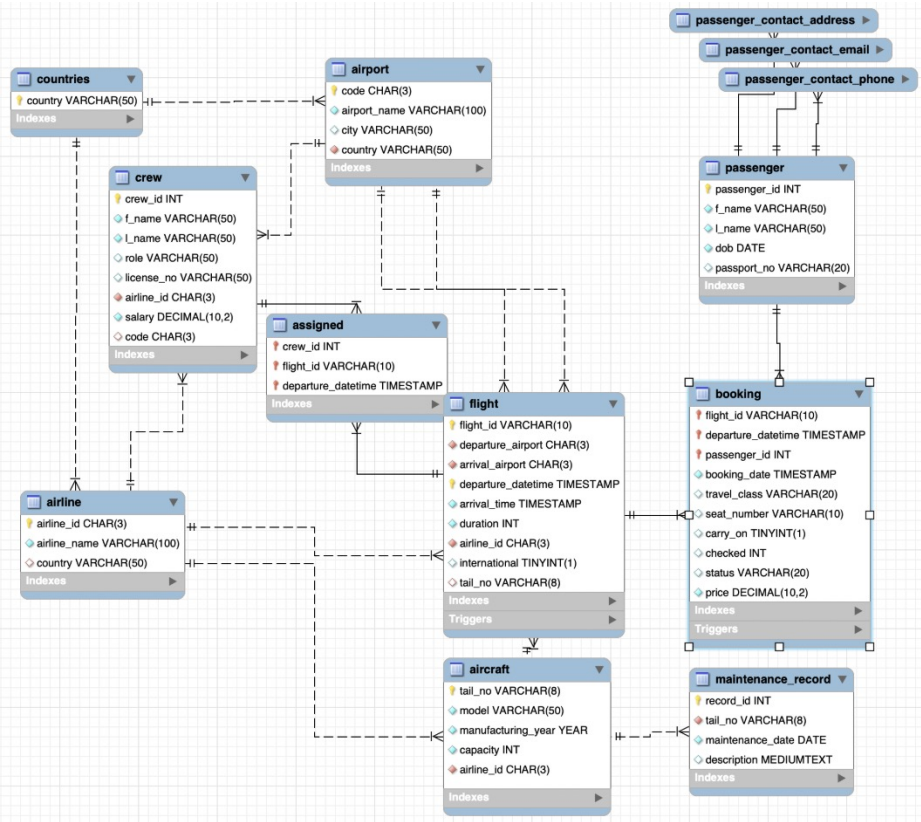


Fig. 3. Relational Diagram

airport, and c) travel date. It then enters additional important search parameters (ex, one-way direct tickets) and locates specific elements on the webpage, including drop-down menus and input fields to interact with. It iterates quickly through all flights in a given query, expanding the drop-down menu to extract maximum data from the div. The scraper is also equipped to handle dynamic content by scrolling through the page and clicking relevant buttons such as 'show more flights' to load additional options. This dynamic handling makes certain that the scraper retrieves all relevant results, even when the initial page load only displays a small subset of the desired flights.

Data Extraction Once flight results are fully loaded, the user will notice that the scraper hangs for a period of time (roughly equivalent to O(n) time + a fixed overhead time cost. The scraper identifies each flight option on the page and extracts information for every div. Error handling was very complicated for this extraction, and would have been much more handleable if the Google Flights API had simply been purchased and directly queried. Timeouts, VPN issues, network issues, and IP bans were all hurdles that needed to be addressed and overcome. To address these challenges, the scraper was written to include robust error handling mechanisms, with functions wrapped in retry logic, except statements, and long timeout buffers. The data was then stored into a structured JSON format, which allows for easy integration into the airline DB. In terms of webscraper efficiency, the code is entirely set up to run with multiple threads for efficiency enhancement, but this has not been implemented due to a lack of need (already meets the 'quick enough' requirement). However, because functions are modular and written with Python's asyncio and Playwright's asynchronous API, the scraper does a good job of minimizing idle time.

Public Datasets Process To obtain data for airlines, airplanes, and airports, we desired to use real data to make our project more applicable to real scenarios. Data regarding airlines was scraped from a Wikipedia list of all airlines. This data was then fed through a Jupyter notebook that cleaned the data and removed any airlines that were defunct, had duplicate two letter airline codes, or were cargo/logistics airlines. Data about airports came from a GitHub repository that was "an up-to-date CSV dump of the `Travelhackingtool.com` airport database with basic information about every IATA airport and city code in the world". As this data only had country codes, not full names it was then paired with a country-code to country name dataset to fill our **countries** table and match airports and countries. This data was similarly cleaned. Lastly, data regarding airplanes was taken from the FAA's aircraft registration database, which publishes an up-to-date and complete list of every aircraft registered in the United States. Of these aircraft, we only considered those that were 'Class 3', meaning they had a all-metal body, and those with capacity greater than 100 seats. This limited our dataset to aircraft most likely used to fly commercial flights in the United States. We assigned each of these aircraft to either a major carrier or random airline.

Existing Databases In addition to scraping data and using public datasets, there were some tables, such as **passenger** and **crew** that required personal information to fill. As this information was not available online, and it was not viable to use real data from an existing database, we located data from the MySQL example database at `https://dev.mysql.com/doc/airportdb/en/airportdb-introduction.html` to fill the crew database. The **passenger** table was filled with 20 fake passengers using data generated by the authors.

Table 1. Some example values in the **passenger** table.

passenger_id	f_name	l_name	dob	passport_no
1	Colden	Johnson	1995-07-16	P12345678
2	Ethan	Deal	1987-03-21	P23456789
3	Michael	Cornell	1999-11-02	P34567890
4	Gavin	Huang	2001-08-14	P45678901
5	Sophia	Martinez	1990-04-10	P56789012

The data from the example MySQL database was then cleaned and each crew member was assigned a role and country, with most being located in the United States to help make example queries easier.

2.2 Database Queries

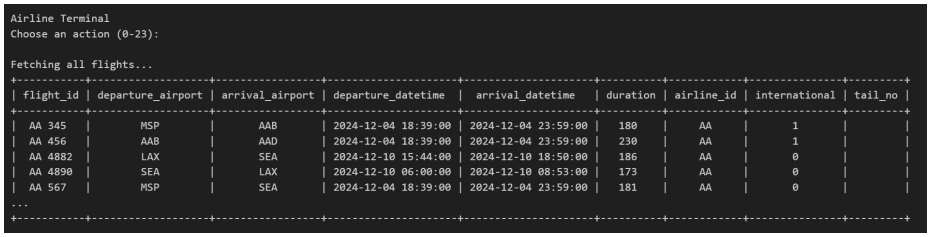
List of terminal query commands listed in table 2.

Selected Basic Queries This section allows the user to retrieve essential information about flights, crew, passengers, and maintenance records. Data can be filtered by various parameters such as airline, airports, or flight assignments. These commands provide basic functionality to the database, allowing users to explore the airline's operations and understand specific details.

View flights The terminal gives the user access to all the essential relations in the database. For instance, the user can display all flights, flights through a specified airline, and flights to and from specified airports.

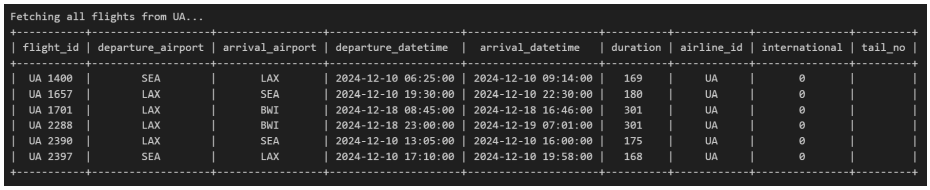
Find shortest route In many cases, direct flights between two airports may not be available. The airline terminal offers a pathfinding feature that uses Djikstra's algorithm to determine the shortest route between two specified airports, minimizing the number of connecting flights required to complete the journey. Dijkstra's algorithm is a graph-based approach that iteratively explores the shortest path from a starting node (departure airport) to a target node (arrival airport).

Statistics Queries The commands in this section focus on analyzing aggregated data, such as airport connectivity, salary metrics, and layover durations. These queries offer insights into operational efficiency and resource allocation.



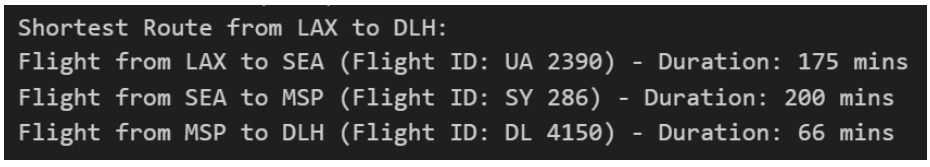
```
Airline Terminal
Choose an action (0-23):

Fetching all flights...
+-----------+-------------------+-----------------+---------------------+---------------------+----------+------------+---------------+---------+
| flight_id | departure_airport | arrival_airport | departure_datetime  |  arrival_datetime   | duration | airline_id | international | tail_no |
+-----------+-------------------+-----------------+---------------------+---------------------+----------+------------+---------------+---------+
|  AA 345   |        MSP        |       AAB       | 2024-12-04 18:39:00 | 2024-12-04 23:59:00 |   180    |     AA     |       1       |         |
|  AA 456   |        AAB        |       AAD       | 2024-12-04 18:39:00 | 2024-12-04 23:59:00 |   230    |     AA     |       1       |         |
|  AA 4882  |        LAX        |       SEA       | 2024-12-10 15:44:00 | 2024-12-10 18:50:00 |   186    |     AA     |       0       |         |
|  AA 4890  |        SEA        |       LAX       | 2024-12-10 06:00:00 | 2024-12-10 08:53:00 |   173    |     AA     |       0       |         |
|  AA 567   |        MSP        |       SEA       | 2024-12-04 18:39:00 | 2024-12-04 23:59:00 |   181    |     AA     |       0       |         |
...
+-----------+-------------------+-----------------+---------------------+---------------------+----------+------------+---------------+---------+
```

Fig. 4. Flights Relation



```
Fetching all flights from UA...
+-----------+-------------------+-----------------+---------------------+---------------------+----------+------------+---------------+---------+
| flight_id | departure_airport | arrival_airport | departure_datetime  |  arrival_datetime   | duration | airline_id | international | tail_no |
+-----------+-------------------+-----------------+---------------------+---------------------+----------+------------+---------------+---------+
|  UA 1400  |        SEA        |       LAX       | 2024-12-10 06:25:00 | 2024-12-10 09:14:00 |   169    |     UA     |       0       |         |
|  UA 1657  |        LAX        |       SEA       | 2024-12-10 19:30:00 | 2024-12-10 22:30:00 |   180    |     UA     |       0       |         |
|  UA 1701  |        LAX        |       BWI       | 2024-12-18 08:45:00 | 2024-12-18 16:46:00 |   301    |     UA     |       0       |         |
|  UA 2288  |        LAX        |       BWI       | 2024-12-18 23:00:00 | 2024-12-19 07:01:00 |   301    |     UA     |       0       |         |
|  UA 2390  |        LAX        |       SEA       | 2024-12-10 13:05:00 | 2024-12-10 16:00:00 |   175    |     UA     |       0       |         |
|  UA 2397  |        SEA        |       LAX       | 2024-12-10 17:10:00 | 2024-12-10 19:58:00 |   168    |     UA     |       0       |         |
+-----------+-------------------+-----------------+---------------------+---------------------+----------+------------+---------------+---------+
```

Fig. 5. Flights by Airline



```
Shortest Route from LAX to DLH:
Flight from LAX to SEA (Flight ID: UA 2390) - Duration: 175 mins
Flight from SEA to MSP (Flight ID: SY 286) - Duration: 200 mins
Flight from MSP to DLH (Flight ID: DL 4150) - Duration: 66 mins
```

Fig. 6. Pathfinding Function

Most connected countries This query returns the most connected countries, calculated by the number of airlines in the country. The US has a very commanding lead in terms of the total number of airlines present. There is a correlation between economic power (US, China, Japan, Germany), as well as population size. It also appears that certain countries, where air travel is more practical to connect the country (very large countries, like Russia and Canada), are overrepresented. The most interesting country on this list is Indonesia – it is heavily overrepresented here because it is an **island** nation, and therefore needs to connect geographically across the ocean (practically accomplished with air travel).

```
Top 10 connected countries by number of airlines:
+----------------+---------------+
|    country     | airline_count |
+----------------+---------------+
| United States  |      74       |
|     China      |      23       |
|     Canada     |      23       |
|     Russia     |      16       |
|     France     |      16       |
| United Kingdom |      16       |
|     Japan      |      14       |
|    Germany     |      14       |
|   Indonesia    |      12       |
|   Australia    |      12       |
+----------------+---------------+
```

Fig. 7. Top 10 Most Connected Countries

Crew Salary Metrics This query provides a battery of metrics related to crew salary, and can be filtered by crew role. Metrics include simple built-in functions (such as min, max, range, and total count). However, it also includes the more complicated values of median, quartile 1, and quartile 3 salaries by role. This query is especially interesting in its implementation, as the summary values of median and quartiles are not built-in functions. They were calculated by ordering the table data, assigning each row a number using the OVER keyword, and then performing simple math operations to identify the correct positions.

```
Fetching crew salary metrics by role...
+---------------------+------------+------------+--------------+-------------+------------+-----------+---------------+-----------+
|        role         | min_salary | max_salary | salary_range | total_count | avg_salary | Q1_Salary | Median_Salary | Q3_Salary |
+---------------------+------------+------------+--------------+-------------+------------+-----------+---------------+-----------+
| Air Traffic Control | 267905.52  | 799059.12  |  531153.60   |     175     | 507181.17  | 382906.38 |   481804.56   | 643175.64 |
|        Pilot        | 414364.68  | 1230105.54 |  815740.86   |     193     | 745882.93  | 570463.02 |   716123.34   | 933836.70 |
|      Airfield       | 240000.00  | 702817.50  |  462817.50   |     205     | 423287.23  | 331919.34 |   404271.00   | 500187.48 |
|  Flight Attendant   | 222181.86  | 585295.38  |  363113.52   |     197     | 362954.38  | 284467.32 |   359371.08   | 422287.38 |
|     Gate Agent      | 189295.26  | 503193.00  |  313897.74   |     182     | 312964.30  | 240000.00 |   299947.29   | 390843.30 |
+---------------------+------------+------------+--------------+-------------+------------+-----------+---------------+-----------+
```

Fig. 8. Crew Salary by Role

Assignment Queries This section deals with assigning essential resources—such as crew and aircraft—to flights. These commands provide functionality to ensure that flights are adequately staffed and equipped.

Assign available crew This is a procedure that automatically fills a specified flight with a minimum of 2 pilots and 5 flight attendants. It checks for crew using the following steps:

1. First, check for crew members whose 'code' attribute matches the departure airport of the selected flight. The 'code' attribute represents the airport that a crew member was most recently sent to.

2. Ensure that the crew members have the 'Pilot' or 'Flight Attendant' role, that they work for the same airline as the selected flight, and that the most recent flight they are assigned to (if any) does not have a arrival datetime after the departure datetime of the selected flight.

3. Update the assigned table to include the new assignments.

4. Update the 'code' attribute in the crew table to match the arrival airport of the selected flight.

Assign available aircraft This is a procedure that automatically locates an available aircraft and assigns it to a given flight. The command follows a similar logical process to the **Assign available crew** procedure.

1. Check for planes owned by the same airline managing the flight

2. Check the number of bookings in the **bookings** table and make sure that the plane chosen has the capacity to handle the number of passengers.

Each of these conditions is given a point value, and then the tail number connected to the plane that scores the highest is inserted into the **flight** table.

```
Fetching crew assignment information...
+---------+-----------+---------------------+
| crew_id | flight_id | departure_datetime  |
+---------+-----------+---------------------+
|   242   |  UA 1400  | 2024-12-10 06:25:00 |
|   345   |  UA 1400  | 2024-12-10 06:25:00 |
|   368   |  UA 1400  | 2024-12-10 06:25:00 |
|   489   |  UA 1400  | 2024-12-10 06:25:00 |
|   888   |  UA 1400  | 2024-12-10 06:25:00 |
|   891   |  UA 1400  | 2024-12-10 06:25:00 |
|   897   |  UA 1400  | 2024-12-10 06:25:00 |
|   242   |  UA 1657  | 2024-12-10 19:30:00 |
|   314   |  UA 1657  | 2024-12-10 19:30:00 |
|   345   |  UA 1657  | 2024-12-10 19:30:00 |
|   368   |  UA 1657  | 2024-12-10 19:30:00 |
|   409   |  UA 1657  | 2024-12-10 19:30:00 |
|   489   |  UA 1657  | 2024-12-10 19:30:00 |
|   891   |  UA 1657  | 2024-12-10 19:30:00 |
+---------+-----------+---------------------+
```

Fig. 9. Crew Assignment Information

```
+------------+-------------------------------------------------------+
| @o_tail_no |                       @message                        |
+------------+-------------------------------------------------------+
|   1013A    | Assigned aircraft with tail_no 1013A to flight CA 1501 |
+------------+-------------------------------------------------------+
```

Fig. 10. Aircraft Assignment Procedure

Maintenance Queries Commands in this group are designed to manage and update information about aircraft maintenance and crew salaries.

Table 2. Airline Terminal Commands Overview

Category	Command ID	Description
Basic	0	Exit terminal
	1	View all flights
	2	View flights by airline, input airline
	3	View flights to and from specified airports, input one, two, or zero airports
	4	Find shortest route between two specified airports, input two airports
	5	View crew working for specified airline, input airline
	6	View all maintenance records
	7	View all maintenance records
	8	View upcoming flights for specified passenger, input passenger ID
	9	View crew assignment table
	10	View crew assigned to flight
Statistics	11	View airport count by country
	12	View most connected countries by num airports
	13	View crew salary metrics
	14	View crew salary metrics by role
	15	View highest and lowest salary
	16	View shortest layovers, input number of layovers to display
	17	View busiest airport
Assignment	18	Manually add passenger booking, input all booking parameters
	19	Assign available crew to specified flight, input flight information
	20	Assign aircraft to specified flight, input flight information
Maintenance	21	Remove old aircraft
	22	Give all crew raises, input percent increase
	23	Give all crew specified role raises, input role name, percent increase

3 Results and Evaluation

3.1 Existing Literature

Natural Language and Ease of Use In the paper by R. L. Klass and another by M. Mony, J. M. Rao, and M. M. Potey, the authors stress the need for flight database systems to handle natural language queries. They also highlight the challenges associated with implementing such systems. They point out that it is difficult to tokenize the input, and also that it is hard to strike a balance between allowing the verbose and wide-scope of human natural language input while retaining the specificity of SQL or backend that the database is queried upon.

This is avoided in our implementation, as we present a menu of options for the user to choose from, and then allow them to numerically navigate the menu. This approach allows for users to understand what each option does while

maintaining the specificity of each choice. There is no disconnect between choice and evaluation in our implementation.

Various Architectures In 'Online Airline Customer Management System Case Study: Eagle Air Uganda Limited', the authors A. Kwesiga and M. S. Salama make the distinction between Service oriented Architecture and Traditional software oriented Architecture defining them as follows

Service Oriented Architecture This type of architecture is used in a distributed system, where by ticketing agents are referred to as services, this architecture is usually implemented using XML (extensible Markup Language), WSDL (Web service description Language), SOAP(Simple object access protocol) and UDDI (Universal Description Discovery and Integration).In this architecture the designed web service that was implemented to follow this architecture would be uploaded to the service registry using UDDI so that client or customer can access the web service from their local operating system. By so doing the customer can access the implemented web service from their various homes, they can reserve flights with ease, search for seats and make payment with ease without going to the airport to do it the old fashioned way.

Traditional Software Oriented Architecture This type of architecture is the type of architecture that involves ticketing agent desktop application software that is not connected to the web service and it requires customers to move down to the ticketing reservation place to book or reserve flights. This ticketing reservation architecture requires a lot of time and effort from both the ticketing agent and the customer, while in the service oriented architecture you can reserve flights with just a click away.

Our Choice Due to time and capital constraints, we have chosen a Traditional Software Oriented Architecture with a Service Oriented element. Partially, this is for ease of use, as all of the software can be run from a single PC. We have incorporated an interface with Google Flights as an example of an outside extension to our project that interacts with the internet. In theory, other portions of our data insertions and interactions with the database could be moved online.

Ontologies Versus Relational Models R. M. Keller argues in favor of use of ontologies in 'Ontologies for aviation data management'. He writes that 'triples' are the ideal way to represent data, as they are able to describe entities in the modeling domain, their properties, and their relationships to other entities, all at the same time. Figure 10 demonstrates a visual representation of how this might work. Keller argues that the uniformity and reflexivity of ontologies makes it easy to reason about data and metadata in one framework.

The authors this paper find Keller's method overly complex, and are not able to find a single instance of this method being implemented in practice. In addition, XML can impose constraints on itself, making it able to satisfy the

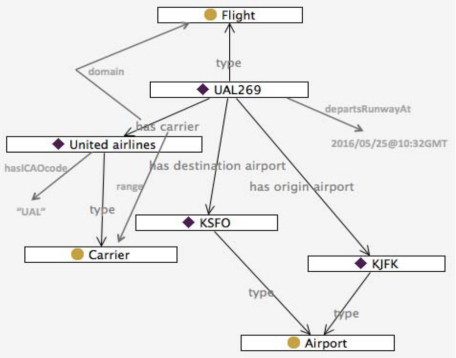


Fig. 11. Figure from cited paper

'triples' that Keller describes. Additionally, a relational model is able to store all the same information given in the example.

4 Conclusion

4.1 System Summary

In this project we designed and developed an airline database to track key flight information. As part of this project, we developed a relational database encompassing all data needed for the successful operation of an airline, following proper database design practices. We wrote necessary views, functions, and procedures to assist with tasks including scheduling, layover calculations, salary data, path finding, and crew/plane assignments. We fully populated the database using information scraped from online, and created a robust backend that can correctly handle the input data.

4.2 Further Development

Although our system does accomplish the goals we set out to achieve, there are certainly avenues for improvement. Given more time, we would have liked to improve the front-end experience, and create a GitHub repository for version control and ease of collaboration. We also would have liked to scrape more data for the **flights** table. Additionally, it would have been a nice feature to have more data automatically pulled from the internet in addition to flights, eg. getting tail numbers automatically from a Air Traffic Control (ATC) API.

5 Appendix

Timeline
Week 1: Project topic approval

Week 2: Project requirement document submission
Week 4: Database design document submission
Week 5: Completed ER Diagram and Relational Diagram. Wrote finalized code for DDL.
Week 6: Began data scraping process and generated/filled data for all tables.
Week 7 (Mon-Thurs): Coded SQL queries, functions, views, procedures, triggers, etc. Connected frontend and backend.
Week 7 (Thurs - Sun): Wrote report and finalized connection of functions with database frontend.

Team Contributions

Colden Google Flights data scraping, data cleaning procedures and JSON transformation, ER Diagram, give crew raises procedure, view busiest airports, give crew specified role raises procedure, view shortest layovers, view highest/lowest salary (cursor), view crew metrics, view most connected countries, view airport count by country, selected report sections.

Michael DDL, functional-dependencies, relational model, international insertion trigger, checked baggage insertion trigger, filling non-flights tables, remove old aircraft procedure, assign aircraft procedure (cursor), selected report sections.

Ethan Terminal frontend (including query connection), project structure, database design, data insertion, SQL file reading functions, data scraping frontend implementation, all table viewing queries, shortest route pathfinding algorithm, available crew assignment procedure, selected report sections.

References

1. 'List of airline codes'. Wikipedia, 08 2020.
2. Lxndrblz, 'GitHub - lxndrblz/Airports: A complete list of IATA Airports including IATA code, ICAO code, Time zone, name, city code, two-letter ISO country code, URL, elevation above sea level in feet, coordinates in decimal degrees, geo encoded city, county and state'. GitHub, 2021.
3. Mysql, 'MySQL Setting Up the airportdb Database'. Mysql.com, 2024.
4. R. L. Klaas, 'A DSS for airline management', ACM SIGMIS Database: the DATABASE for Advances in Information Systems, vol. 8, no. 3, pp. 3–8, 1977.
5. M. Mony, J. M. Rao, and M. M. Potey, 'An overview of NLIDB approaches and implementation for airline reservation system', International Journal of Computer Applications, vol. 107, no. 5, 2014.
6. A. Kwesiga and M. S. Salama, 'Online Airline Customer Management System Case Study: Eagle Air Uganda Limited', 2012.
7. R. M. Keller, 'Ontologies for aviation data management', in 2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC), 2016, pp. 1–9.