Airline Database Optimization for Improved   
Logistics and Lifetime Customer Value

Patrick Copeland

Computer Science

Georgia State University

Atlanta, Georgia

[pcopeland6@student.gsu.edu](mailto:pcopeland6@student.gsu.edu)

Janis Grikstas

Computer Science

Georgia State University

Atlanta, Georgia

[jgrikstas2@student.gsu.edu](mailto:jgrikstas2@student.gsu.edu)

Amisha Gupta

Computer Science

Georgia State University

Atlanta, Georgia

[agupta30@student.gsu.edu](mailto:agupta30@student.gsu.edu)

Sanghamitra Volam

Computer Science

Georgia State University

Atlanta, Georgia

[svolam1@student.gsu.edu](mailto:svolam1@student.gsu.edu)

Pranav Chandiramani

Computer Science

Georgia State University

Atlanta, Georgia

[pchandiramani1@student.gsu.edu](mailto:pchandiramani1@student.gsu.edu)

Brian Franklin

Computer Science                        
Georgia State University

Atlanta, Georgia

[bfranklin12@student.gsu.edu](mailto:bfranklin12@student.gsu.edu)

Saahaj Mattey

Computer Science

Georgia State University

Atlanta, Georgia

[smattey1@student.gsu.edu](mailto:smattey1@student.gsu.edu)

*Abstract*— *Our project is a dummy database that simulates commercial airline data and focuses on building a model to organize optimal solutions to passenger flying patterns and customer flight selection. The model can be used to inform logistical, staffing, and customer needs in order to improve business practices and inform decisions. With this research project, we hope to enable the airports to better manage their customers, employees, and capital, and provide solutions for streamlining resource utility management.*

# Introduction

Our database was built to provide an airport with information that displays how customers fly, how people choose an airline, as well as the logistics of an airport. The database can be used to track how customers fly by using the airline company, passengers, and flights tables. It can also be utilized to evaluate how people choose an airline, analyzing various metrics such as destinations, origins, and the passenger.

Airports can use this database to manage their own business by looking at the statistics, and implementing possible changes to improve efficiency. The database can also be used to identify an airlines most valuable customers and to analyze trends to bring more passengers in, while retaining existing customers.

# Technology

The main technology used in the creation and development of the database system is PostgreSQL. The database system and its tables were built using the PostgreSQL platform on the local machine.

Python methods, third-party datasets, and integer/data randomizer methods were used to generate the data and save them as .csv files, which we then imported into the database system using SQL.

# Survey of Related Works

Our database design is similar to the design proposal for United Airlines but it is meant to be used by an airport to see how customers fly, and to see the distributed resources/logistics of the airport etc. Our database design tackles a key issue that the United Airlines database runs into: An itinerary list for customers’ flights.

To solve this problem, we have created a relationship table (*flying\_on*) between the *flights* table, which lists out all the flights leaving or arriving at a given airport, and the *passengers* table, which lists out the customers from that airport. Therefore, our database provides airports with a better way of understanding the airlines and the customers that they do business with.

# Data Sources

We built our dummy dataset using imported data from existing datasets, and by generating random data as needed when available datasets fell short of our requirements. The decision to build our dummy dataset on top of real-world data was based on the hypothesis that a dataset with more real-world data would lead to an easier to understand and more error-free result than a dataset built entirely from scratch. Basing our dataset on real-world data also provides challenges that give us opportunities to interact more with the data, and to hopefully increase our understanding.

## Finding Data

The primary source for data collection was kaggle.com. By searching for datasets using keywords such as “airline company”, “airline business”, “flight data”, etc., were able to collect a large amount of data which we then processed according to our own needs. We based those needs primarily on an preliminary ER diagram, but made adjustments as we felt they were needed throughout the process.

## Generating Data

Some data was not readily available and had to be generated randomly as needed. For instance, attributes containing personal information, such as employee or customer names and identification numbers, and phone numbers were all data that we expected would need to be randomly generated. We also found that in some instances, data had to be spoofed when we simply could not find any datasets containing those attributes, which was true in our case for flight numbers.

# Project Details

In order to create an accurate and usable database, we

started by surveying existing datasets to determine what, if any, usable data existed. Once our data was collected, our primary objective became cleaning and organized that data into a relational model that we could query. For this we primarily used Python.

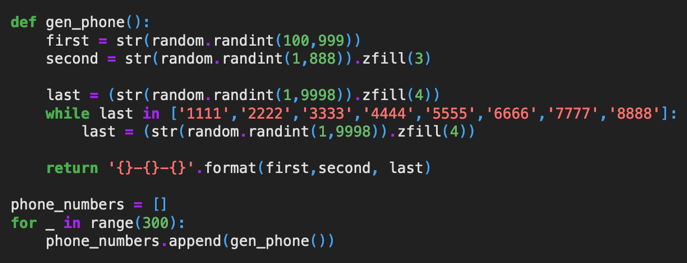
## Implementation (Python)

Cleaning the data was relatively straightforward because we already had a rough ER diagram to guide us in deciding what features we wanted to include in each table. This mean that, after loading the tables into an IDE, cleaning our data mostly meant deleting columns that we deemed irrelevant, and removing rows with missing values.

The next step of our implementation involved generating random data where it was needed. As an example, for the *employees* table, we randomly generated the employee\_id feature using the random.randint function from Numpy:



Similarly, we generated the customer\_name field in the *passengers* table using an existing Python function called names.get\_full\_name, and for the phone numbers in *passengers* table, we wrote a function to generate them, as you can see below:



Once we completed the work on our tables, we saved them all as .csv files and imported them to PGAdmin to build our database.

## Implemantation (SQL)

We primarily used the PGAdmin GUI to import our data to the database, but the software provides access to the SQL code it uses in the background, which we have submitted with this paper.

## Key Findings

We realized that having good, clean data that is both realistic and accurate is very crucial to have queries that are accurate and reasonable. For example, in our flights table, the number of seats for some rows is in the thousands, which is totally unreasonable. If we could go back, we would spend more time as a group analyzing situations like this before uploading the tables, which would have made our queries more accurate. It is a lesson that we have learned through trial and error, and something we will work to avoid in the future.

# Analysis

We ran into several difficulties over the course of this project. Surprisingly, almost all of our issues involved either the data itself, or the technology we used to create our database.

## What Didn’t Work

One of the problems we had to face was inconsistencies and anomalies in the data tables. These arose a variety of reason, but the main reason was simply due to the fact that there was a learning curve in terms of forethought when building the tables.

For instance, when we merged columns for the number of seats and passengers for the *flights* table, we failed to consider realistic limits on those values. This oversight led to many flights having thousands of available seats where a more realistic upper limit might be something closer to 300 seats.

We also had a number of issues with our relationship tables. One key problem arose when we tried to create these tables and realized that many of them had unequal column lengths.

At first, PGAdmin wouldn’t even allow us to create these tables, but ultimately a larger problem arose as we could not find a way to create the tables without the rows containing a NAN value being edited so that all entries in that row became NAN. As a result, many of our keys were effectively deleted from the relationship tables. To illustrate one way this has affected our data, consider the *works\_for* table.

The *works\_for* table contains 2 features, *airline* and *employee\_id*. The *airline* feature, from the *airline\_company* table, has 281 rows, while the *employee\_id* feature, from the *employees* tables has 295 rows. But when we looked at the *works\_for* table, we saw that the total number of rows was 280, representing a loss of 15 rows. We also noted that each listed airline only had one employee.

It’s difficult to say the true extent to which this error has affected the database because, as of now, we have still been unable to solve this problem, despite many hours devoted to troubleshooting. Were we to approach attempt this project again, we would most like fix this problem on the from end by insuring the data tables were more consistent from the start.

## What Worked

Even though we ran into problems, we were still able to build a functioning database that allowed us to see the relationships between the various features of our dataset. We spent a good deal of time testing the dataset to see how and why it would break and, overall, we were pleased to find that it performed well for most queries. We have provided some sample queries below that appear to function as they should.

## What surprised us

We were surprised by the ease of pulling data from the database using queries after the database was created. When we were first creating the database and setting everything up, it was a hassle, so we assumed pgAdmin would be problematic as well, but using the query tool in pgAdmin proved to be simple.

# Conclusion

Our project shows that we were able to model an airline database successfully, and pointed toward simple steps that could be taken in future efforts to immediately improve the efficiency and performance of a such a database.

By making a more detailed plan with consideration for how the data across various table interact, our work would have been much easier, and our results would have been more informative. If we were to go back, we would spend more time in the table creation portion of the project, as it would have made everything after more accurate and reasonable.

Furthermore, in the future, we would hope to have more time to become comfortable with PGAdmin in order to make troubleshooting technological issues easier and more productive. We spent a good amount of time working through issues that were only prevalent for some of us. We could have also used a cloud-base server, which might have made it easier to collaborate with our group members.

We felt that with this project, we were able to utilize almost every skill we have learned throughout the entirety of this course, whether it was ER diagrams and basic database setup rules from the beginning of the course, or the complex queries that we learned towards the end of the course.

Even though our final product has some limitations, in the end, we learned a great deal about how to collect data, build databases, think about databases and their relationships, and conceive and execute queries on a database. We believe these skills can easily be built on in the future and applied to future projects and real-world applications.

