April 18, 2024

**Comfort Airlines**

**Functional Specification**

Client: Professor Michael Oudshoorn

Group Name: Byte Me

Contractors: Joellen A., Kyle B., Hanna Z., Alex H., Joshua T., JR V.

**About Functional Specification**

The "Functional Specification - Documentation" for Comfort Airlines meticulously details the technical requirements and specifications for the project, managed by the Byte Me group. This document encompasses a range of essential software components including algorithms for calculating passenger density and flight times, web pages for financial summaries and flight information, and database management for flights, airports, aircraft, and passengers. It serves as a comprehensive guide for developers, outlining the functionalities, system requirements, and execution details necessary for the software's successful implementation and integration within the broader airline management system.

**Files**

* passengerDensity.cpp
* Calc.php
* Index.php
* Route.php
* Repo.php
* flightTime.h
* flightTime.cpp

**Database Tables**

* Flight
* Airport
* Aircraft
* Passenger

**File #1:** PassengerDensity.cpp

#### **Overview**

The passengerDensity.cpp file calculates the estimated daily passenger traffic between pairs of airports based on their metropolitan area populations. The calculation assumes a specific proportion of the population travels daily, adjusting for the population of the departure airport.

#### **System Requirements**

* C++ compiler (e.g., g++, clang++)
* Standard C++ library

#### **External Dependencies**

None.

#### **Main Functionalities**

* Population Data Initialization: Initializes a vector of Airport structs, each containing an airport code and its metro population.
* Total Metro Population Calculation: Calculates the total population of all metro areas represented by the airports.
* Daily Passenger Traffic Calculation: Calculates and outputs the estimated number of passengers traveling on any given day between each pair of airports, excluding pairs where the departure and destination airports are the same.

#### **Functions and Structures**

* **Airport Struct**
	+ Description: A data structure to hold the airport code and its corresponding metropolitan population.
	+ Members:
		- std::string code - Airport code.
		- double population - Metropolitan population.
* **main Function**
	+ Description: The entry point of the program. Initializes airport data, calculates total metropolitan population, and calculates daily passenger traffic between all pairs of airports.
	+ Parameters: None
	+ Returns: Integer (0 on successful execution)

#### **Compilation Instructions**

To compile the passengerDensity.cpp file, use a C++ compiler such as g++. The command line instruction for compilation using g++ would be:

g++ -o passengerDensity passengerDensity.cpp

This command compiles the passengerDensity.cpp file and creates an executable named passengerDensity.

#### **Execution Instructions**

After compiling the program, it can be run from the command line by navigating to the directory containing the compiled executable and running:

./passengerDensity

This command executes the program, which will then output the estimated daily passenger traffic between each pair of airports defined in the program.

#### **Error Handling**

The program does not explicitly include error handling mechanisms as it operates with hardcoded, static data and does not perform any external file reading or user input that could result in errors.

#### **Output(Portion)**



#### **Conclusion**

The passengerDensity.cpp document outlines a straightforward C++ program for estimating daily passenger traffic between airports based on metropolitan populations. It includes clear instructions for compilation and execution, providing a simple yet effective tool for transportation analysis.

**File #2:** Calc.php

**Overview:** The calc.php file is designed to retrieve and calculate various operational costs for aircraft operations over a specified date range. It then prepares a financial summary report focusing on profit and loss aspects related to aircraft leasing and operational costs.

**System Requirements:**

PHP 7.4 or higher

PDO extension for PHP

MySQL database access

Web server (e.g., Apache, Nginx)

**External Dependencies:**

MySQL Database: comfort\_airlines\_byteme

Database Tables:

AIRCRAFT

FLIGHT

**Main Functionalities within Calc.php:**

#### **1. Database Connection**

* Description: Establishes a PDO connection to the MySQL database.
* Method Signature: N/A (Initialization within global scope)

#### 2**. Fetch Flight Data**

* Description: Executes a SQL query to fetch flight details, including tail number, flight date, and distance, from the AIRCRAFT and FLIGHT tables within a specified date range.
* Parameters: None (uses hardcoded date range within the SQL query)
* Returns: Array of flight details

#### **3. Calculate Leasing Fees**

* Description: Calculates leasing fees based on aircraft tail number prefixes and days of operation within the specified date range.
* Parameters: None (uses $flights array from database query)
* Returns: Populates $leasingFeesPerDay array

#### **4. Calculate Terminal Fees**

* Description: Calculates terminal fees based on takeoffs and landings per flight.
* Parameters: None (uses $flights array from database query)
* Returns: Populates $terminalFeesPerDay array

#### **5. Calculate Fuel Fees**

* Description: Calculates fuel fees based on flight distance, aircraft type, and fixed price per gallon.
* Parameters: None (uses $flights array from database query)
* Returns: Populates $fuelFeesPerDay array

#### **6. Calculate Total Operational Costs**

* Description: Sums up leasing, terminal, and fuel fees to calculate total operational costs per day and for a two-week period.
* Parameters: None (uses previously populated arrays)
* Returns: Populates $totalOperationalCostsPerDay and $totalOperationalCostsOverTwoWeeks variables

#### **Error Handling**

* Utilizes try-catch blocks to manage database connection errors and ensure robust error reporting.
* Includes checks for empty result sets, displaying a message if no flights are found.

#### **HTML Output**

* Produces a comprehensive financial summary report in HTML format, detailing operational costs, revenue, and occupancy rates in tabular form, accompanied by CSS styling for visual presentation.

#### **Security Considerations**

* Employs PDO for database interactions to protect against SQL injection.
* Implements error reporting and exception handling for enhanced application security and reliability.

**Conclusion**

This functional specification document provides a detailed overview of the calc.php file's capabilities, including database interactions, calculations for operational costs, and the generation of a financial summary report. It serves as a guide for understanding the file's structure and functionality for development or integration within broader systems.

**File #3:** Index.php || Main Function/File

#### **Overview**

The system presents flight information for arrivals and departures at selected airports. It dynamically generates a webpage displaying flight details such as flight number, number of passengers, scheduled and actual arrival/departure times, and the tail number of aircraft. Additionally, it includes functionalities for formatting output and displaying status based on flight timings. This file serves as our main function in the project due to its integration with the main functionality, user interaction, error handling, and security considerations of the project.

#### **System Requirements**

* PHP 7.4 or higher
* Access to a MySQL database with tables for flights, airports, aircrafts, and passengers.
* Web server (e.g., Apache, Nginx) capable of executing PHP scripts

#### **External Dependencies**

* comfort\_db.php: A PHP file that establishes a connection to the database and provides access to the $db variable for executing SQL queries.

#### **Main Functionalities**

* Flight Information Retrieval
	+ Retrieves and displays flight information for arrivals and departures based on the selected airport code.
* Data Formatting
	+ Provides a method for formatting flight information fields to ensure consistent presentation within the user interface.

**Main Function/File**

This index.php file is effectively the main function of the project for several reasons:

* Core Functionality:
	+ The file serves as the entry point and orchestrator of the entire system. It retrieves flight information, formats it, and dynamically generates the webpage to display the information. This means that without this file, the essential function of the project—displaying flight information—cannot be achieved.
* Integration of Various Components:
	+ index.php integrates various components of the project seamlessly. It interacts with the database through comfort\_db.php to retrieve flight data, formats the data for consistent presentation, and dynamically updates the webpage based on user interaction. It acts as a bridge between the backend (database) and the frontend (web interface).
* User Interaction:
	+ The file provides a user interface for interaction with the system. Users can select airports, view arrival or departure information, and see flight statuses. This interaction is crucial for the usability and effectiveness of the project, and index.php facilitates it.
* Error Handling and Security Considerations:
	+ Although the file itself does not handle database connections or detailed error handling, it outlines the necessity for error handling and security considerations. This shows that index.php is not only responsible for core functionality but also for ensuring the robustness and security of the system as a whole.
* Deployment and Execution:
	+ index.php provides clear instructions for deployment and execution, making it easy for developers to integrate into a web server environment. It specifies the PHP version required, the necessity of a database connection, and considerations for error handling and security.
* Natural Main in PHP:
	+ In PHP “index” is a reserved word, serving as the default entry point or main function within web applications. When ‘index.php’ is accessed through a web server, it acts as the default handler for requests to the directory it resides in. This convention is deeply ingrained in web server configurations, where the server automatically looks for files named "index" (and specific extensions like ".php") when a directory is requested. Therefore, by naming a file "index.php," developers signal to the web server that this file serves as the primary entry point for the corresponding directory. In the context of this project, this specific “index.php” encapsulates the core functionality and user interaction, making it the natural choice for the main function of the project.

**Functions**

* formatField
	+ Description: Formats a string to display within a specified maximum number of characters, ensuring uniformity in the visual presentation of data.
	+ Parameters:
		- $string (string): The string to be formatted.
		- $maxChars (int): The maximum number of characters to display.
	+ Returns: A string with HTML span elements encapsulating each character for styling purposes.

#### **Web Interface Components**

* Dropdown Menu for Airport Selection
	+ Allows users to select an airport code from a dropdown menu populated with codes from the AIRPORT table in the database.
* Arrivals and Departures Display
	+ Dynamically updates the webpage to display either arrival or departure flight information based on the user's selection.
* Flight Status Indication
	+ Indicates the status of each departure (Cancelled, Early, On-Time, Delayed) based on a comparison of scheduled and actual departure times, with distinct styling for each status.

#### **Compilation and Execution Instructions**

This PHP script does not require compilation. It should be placed in a web server's document root or an appropriate directory. The web server must be configured to execute PHP scripts. To access the functionality, navigate to the URL corresponding to the location of the script on the web server.

#### **Error Handling**

* The system assumes that comfort\_db.php successfully connects to the database. Error handling for database connection issues should be implemented within comfort\_db.php.
* Uses htmlspecialchars to prevent XSS (Cross-Site Scripting) attacks when displaying data from the database.

#### **Security Considerations**

* Sanitizes output using htmlspecialchars to enhance security against XSS attacks.
* Should validate and sanitize user inputs (e.g., the selected airport code) to prevent SQL injection and other web vulnerabilities, although specific mechanisms for this are not detailed in the provided code snippet.

### **Conclusion**

This functional specification outlines a system for displaying flight information based on user-selected airport codes. It includes a web interface for interaction, methods for data retrieval and formatting, and considerations for security and error handling. This document serves as a guide for understanding, developing, and integrating the system within a broader web application context. It also serves as the main function for the project as it encapsulates the main functionality, user interaction, error handling, and security considerations of the project, making it the central component and effectively the main function of the entire system.

**File #4:** Route.php

**Overview:**

The route.php file is crafted to manage and display detailed information regarding flight routes. It is equipped to handle data such as flight frequency, average occupancy, and operational metrics for each route.

**System Requirements:**

* PHP 7.4 or higher
* PDO extension for PHP
* MySQL database access
* Web server (e.g., Apache, Nginx)

**External Dependencies:**

* MySQL Database:
* comfort\_airlines\_byteme
* Database Tables: ROUTE, FLIGHT

**Main Functionalities within Route.php:**

**Database Connection**

* Description: Sets up a PDO connection to the MySQL database.
* Method Signature: N/A (Initialization within global scope)

**Retrieve Route Data**

* Description: Executes SQL queries to retrieve data related to specific routes from the ROUTE and FLIGHT tables.
* Parameters: None (may utilize a user-provided route code)
* Returns: Array of route details

**Display Route Information**

* Description: Formats and displays the route data, including operational statistics.
* Parameters: None (utilizes the data array from database queries)
* Returns: HTML content

**HTML Output:**

Outputs detailed route information in a structured HTML format, complete with CSS styling for clarity and presentation.

**Security Considerations:**

Uses PDO for secure database interactions to mitigate SQL injection threats.

Incorporates error handling to provide stable and secure user experiences.

**Conclusion:**

The route.php script provides a structured interface for presenting route-specific information, allowing for insightful analysis of flight route performance and operational metrics.

**File #5:** Route.php

**Overview:**

The repo.php file is designed to process and present financial reporting for the airline's operations. It calculates and summarizes costs and revenues, highlighting profit and loss outcomes.

**System Requirements:**

* PHP 7.4 or higher
* PDO extension for PHP
* MySQL database access
* Web server (e.g., Apache, Nginx)

**External Dependencies:**

* MySQL Database: comfort\_airlines\_byteme
* Database Tables: COSTS, REVENUE

**Main Functionalities within Repo.php:**

**Database Connection**

* Description: Initiates a PDO connection to the MySQL database.
* Method Signature: N/A (Initialization within global scope)

**Fetch Financial Data**

* Description: Retrieves financial data for a given period from COSTS and REVENUE tables.
* Parameters: None (could be dependent on user-selected date range)
* Returns: Array of financial data

**Calculate Financial Summaries**

* Description: Processes the retrieved data to compute various financial metrics, such as operational costs and revenues.
* Parameters: None (works with fetched data)
* Returns: Populated arrays with calculated data

**Generate Financial Report**

* Description: Assembles a detailed report from calculated data, presenting it in a user-friendly format.
* Parameters: None (uses calculated data arrays)
* Returns: HTML output of the financial report

**HTML Output:**

Creates a financial report in HTML format, showcasing operational costs, revenues, and profit/loss figures, complemented by CSS for proper formatting.

**Security Considerations:**

Employs PDO for interactions with the database, providing a safeguard against SQL injection.

Implements error handling mechanisms to enhance the application’s reliability and security.

**Conclusion:**

Repo.php functions as a critical component for generating comprehensive financial reports, essential for monitoring the fiscal aspects of airline operations.

**File #6:** flightTime.h

**Overview**

This functional specification outlines the flightTime.h header, which is part of a software system designed to calculate and provide the flight time between two airports. It is a header file that defines the functions that are needed for flightTime.cpp to run.

**System Requirements**

* Operating System: The software is designed to be operating system agnostic, provided the system supports C++ compilation and execution environments.
* Compiler: A C++ compiler that supports C++11 standard or later, such as GCC (GNU Compiler Collection) version 5.0 or later, or Clang version 3.4 or later.
* Memory: Sufficient memory to compile and run C++ applications, which will vary based on the complexity and requirements of the implementing software that utilizes flightTime.h.

**External Dependencies**

* Standard Template Library (STL): The code utilizes components from the STL, such as <string> for handling string operations and <cmath> for mathematical operations.
* I/O Libraries: The <iostream> library is used for input and output operations, which suggests the implementing software will involve console-based interactions.

**Functions and Structures**

* main() (int): Presumably the entry point for the implementing program. However, it's unusual for main to be declared in a header file, and its purpose here is unclear without the implementation context.
* flightTime(Airport, Airport, Plane, double) -> double: Calculates the flight time between two locations. The function takes two Airport struct arguments, one Plane struct argument, and one double argument, returning a double that represents the flight time in hours. The exact semantics of the arguments are not described in the provided header comment, but they could represent departure and arrival locations, the type of aircraft, and other flight-related parameters.
* Airport: Structure defined with a name (character array), latitude, and longitude.
* Plane: Structure defined with a name (character array) and speed.

**Compilation Instructions**

To compile a software program that includes flightTime.h, follow these general steps:

* Ensure all external dependencies are met and the appropriate C++ compiler is installed.
* Include flightTime.h in your source file that implements the main function and uses the flightTime function.
* Use the C++ compiler to compile your source file(s). For example, using GCC: g++ -o yourProgramName yourSourceFile.cpp -std=c++11

**Execution Instructions**

* After compilation, run the executable generated by the compiler. The execution method will depend on the operating system.
	+ On Unix/Linux/Mac: ./yourProgramName
	+ On Windows: yourProgramName.exe
* Follow any runtime instructions provided by the program, which may involve entering specific inputs related to flight time calculation.

**Conclusion**

The flightTime.h header provides the foundation for a flight time calculation tool, outlining the basic structure required for such a functionality. However, specific implementation details, including the logic within the flightTime function and how inputs are processed, are not covered in this document and must be defined in the corresponding source files. This specification should serve as a guideline for developers looking to integrate or further develop the flight time calculation feature within their software systems.

**File #7:** flightTime.cpp

**Overview**

 This module provides functionality to calculate the flight time between two airports, considering the aircraft's speed and applying a delay factor based on the geographical position of the departure and destination airports. The core of this module is the ‘calcFlightTime’ function, which outputs the estimated flight duration in hours.

**System Requirements**

* C++11 or higher
* A compatible C++ compiler (e.g., GCC, Clang)
* Standard library support

**External Dependencies**

* flightTime.h: A header file that declares the Airport and Plane structs, which are essential for the input parameters of the calcFlightTime function.

**Functions and Structures**

calcFlightTime

* Description: Calculates the flight time between two airports.
* Parameters:
	+ destinationAirport (Airport): A struct containing the name, latitude, and longitude of the destination airport.
	+ departureAirport (Airport): A struct containing the name, latitude, and longitude of the departure airport.
	+ planeType (Plane): A struct containing the name and speed (in mph) of the plane.
	+ distance (double): The distance between the two airports in miles.
* Returns: A double representing the flight time in hours.

**Compilation Instructions**

To compile a software program that includes flightTime.cpp, follow these general steps:

Ensure all external dependencies are met and the appropriate C++ compiler is installed.

Include flightTime.cpp in your source file that implements the main function and uses the flightTime function.

Use the C++ compiler to compile your source file(s). For example, using GCC: g++ -o yourProgramName yourSourceFile.cpp -std=c++11

**Execution Instructions**

After compilation, run the executable generated by the compiler. The execution method will depend on the operating system.

* + On Unix/Linux/Mac: ./yourProgramName
* Follow any runtime instructions provided by the program, which may involve entering specific inputs related to flight time calculation.

**Output**

DepartureAirportName -> DestinationAirportName, Flight time: nan

ATL -> CLT, Flight time: 1.015358

CLT -> DCA, Flight time: 1.191907

DCA -> BNA, Flight time: 1.696960

BNA -> ATL, Flight time: 1.057163

JFK -> CDG, Flight time: 3.507527

CDG -> JFK, Flight time: 3.806973

JFK -> BOS, Flight time: 0.947709

BOS -> DCA, Flight time: 1.397837

DCA -> JFK, Flight time: 0.992690

JFK -> SEA, Flight time: 4.706486

SEA -> MSP, Flight time: 3.237106

MSP -> JFK, Flight time: 2.554813

ATL -> TPA, Flight time: 1.413181

TPA -> MIA, Flight time: 1.593838

MIA -> MCO, Flight time: 0.958142

MCO -> ATL, Flight time: 1.315978

DFW -> PHX, Flight time: 2.093641

PHX -> IAH, Flight time: 2.520581

IAH -> DFW, Flight time: 1.011588

DFW -> LAS, Flight time: 2.408958

LAS -> DEN, Flight time: 1.690445

DEN -> DFW, Flight time: 1.845430

ORD -> SEA, Flight time: 3.527489

SEA -> ORD, Flight time: 3.828827

ORD -> SFO, Flight time: 4.061261

SFO -> ORD, Flight time: 3.739912

ORD -> SLC, Flight time: 2.963139

SLC -> ORD, Flight time: 2.736363

DFW -> ORD, Flight time: 1.986789

ORD -> DFW, Flight time: 2.142925

DFW -> ATL, Flight time: 1.865074

ATL -> DFW, Flight time: 2.009739

DFW -> JFK, Flight time: 2.976630

JFK -> DFW, Flight time: 3.226048

ORD -> ATL, Flight time: 1.782189

ATL -> ORD, Flight time: 1.657119

ORD -> JFK, Flight time: 2.024585

JFK -> ORD, Flight time: 1.878638

JFK -> ATL, Flight time: 2.064142

ATL -> JFK, Flight time: 1.914791

CLT -> ORD, Flight time: 1.618510

ORD -> TPA, Flight time: 2.520944

TPA -> CLT, Flight time: 1.491454

LAX -> JFK, Flight time: 4.798627

JFK -> PHX, Flight time: 4.627688

PHX -> LAX, Flight time: 1.256759

ORD -> MCO, Flight time: 2.520942

MCO -> SAN, Flight time: 4.251817

SAN -> ORD, Flight time: 3.535225

JFK -> LAS, Flight time: 4.800688

LAS -> CLT, Flight time: 4.189329

CLT -> JFK, Flight time: 1.546578

MSP -> TPA, Flight time: 3.075827

TPA -> DCA, Flight time: 2.008019

DCA -> MSP, Flight time: 2.199573

PHX -> DCA, Flight time: 3.963601

DCA -> ORD, Flight time: 1.663689

ORD -> PHX, Flight time: 3.314689

SEA -> BNA, Flight time: 4.304745

BNA -> DFW, Flight time: 1.826017

DFW -> SEA, Flight time: 3.429627

SAN -> IAH, Flight time: 3.063322

IAH -> CLT, Flight time: 2.169989

CLT -> SAN, Flight time: 4.487667

SLC -> DFW, Flight time: 2.485223

DFW -> BOS, Flight time: 3.263837

BOS -> SLC, Flight time: 4.538165

DEN -> ATL, Flight time: 2.871667

ATL -> PHL, Flight time: 1.756692

PHL -> DEN, Flight time: 3.529825

**Conclusion**

The ‘flightTime.cpp’ module is designed to calculate flight times between two airports. It serves as a utility within a broader application context, such as flight management systems for airlines. The module emphasizes functionality over user interface, with future development focused on enhancing reliability and incorporating error handling mechanisms.

**File #8:** flighttable.csv (database)

**Overview**

This file contains comprehensive data on flights managed by Comfort Airlines, including details such as flight number, departure and arrival airports, flight dates, and times. The information in this CSV file can be used to analyze flight schedules, calculate flight durations, and optimize airline operations.

#### **System Requirements**

- A software capable of reading CSV files (e.g., Microsoft Excel, Google Sheets, or a programming language like Python with libraries for data manipulation).

- Adequate storage to handle potentially large data volumes.

**External Dependencies**

- `pandas` library for Python: Provides high-performance, easy-to-use data structures, and data analysis tools.

- Any CSV reader like Excel or similar software that can import and process CSV format.

**Functions and Structures**

The CSV file is expected to have the following columns:

- `FlightNumber`: The unique identifier for each flight.

- `DepartureAirport`: The airport code from where the flight departs.

- `DestinationAirport`: The airport code where the flight arrives.

- `NumberOfPassengers`: The number of passengers booked on the flight.

- `ScheduledDepartureTime`: The scheduled time of departure.

- `ActualDepartureTime`: The actual time of departure.

- `ScheduledArrivalTime`: The scheduled time of arrival.

- `ActualArrivalTime`: The actual time of arrival.

- `TailNumber`: The aircraft's tail number used for the flight.

- `Distance`: The distance between the departure and destination airports.

- `DepartureDate`: The date on which the flight departs.

- `ArrivalDate`: The date on which the flight arrives.

- `Day`: The day of the week the flight operates.

- `PassengerDensity`: A derived metric indicating how full the flight is relative to its capacity.

**Compilation Instructions**

No compilation is necessary for CSV files as they are read directly by software that supports this format.

**Execution Instructions**

1. Open your preferred CSV handling tool (e.g., Microsoft Excel, Google Sheets).

2. Import or open the `flighttable.csv` file.

3. Utilize functions within the software to sort, filter, and analyze the flight data as needed.

**Conclusion**

The `flighttable.csv` serves as a fundamental data source for operations and strategic planning at Comfort Airlines. It enables a broad spectrum of analytical tasks from operational adjustments based on passenger density to strategic decisions about flight schedules. The simplicity of the CSV format ensures compatibility with numerous systems, emphasizing utility and accessibility.

**File #10:** airporttable.csv (database)

**Overview**

This table contains detailed information about airports used by Comfort Airlines. It includes data on airport codes, passenger statistics, gate usage, and geographical coordinates. The `AIRPORT` table is crucial for managing operations, including gate assignments, tracking airport traffic, and strategic planning.

**System Requirements**

- Access to a database management system (DBMS) that supports SQL, such as MySQL, PostgreSQL, or Oracle.

- Adequate permissions to view and manipulate the database tables.

**External Dependencies**

- SQL management tools such as phpMyAdmin, MySQL Workbench, or similar tools for database interaction.

- Drivers and connectivity for the database server (e.g., ODBC, JDBC).

**Functions and Structures**

The `AIRPORT` table consists of the following columns:

- `AirportCode`: A unique identifier for each airport.

- `NumberOfDepartingPassengers`: The total number of passengers departing from this airport.

- `GateUsed`: The gate most frequently used for departures.

- `Latitude`: Geographical latitude of the airport.

- `Longitude`: Geographical longitude of the airport.

**Compilation Instructions**

No compilation is necessary as this involves direct database manipulation and querying.

**Execution Instructions**

1. Connect to your database using your preferred SQL management tool.

2. Query the `AIRPORT` table using SQL commands to retrieve, update, or analyze the data. Example query:

 ```sql

 SELECT \* FROM AIRPORT;

 ```

3. Use JOIN operations with other tables (e.g., `FLIGHT`) to perform comprehensive analyses, such as calculating distances between airports or determining the busiest routes.

**Conclusion**

The `AIRPORT` table is a fundamental component of the data infrastructure at Comfort Airlines, supporting essential functions from operational logistics to strategic analysis. Its integration with other database tables allows for a holistic view of airline operations, facilitating improved decision-making and efficiency.

**File #11:** passengertable.csv (database)

**Overview**

The PASSENGER table in the Comfort Airlines database contains individual records of passengers for each flight. This table is instrumental for managing passenger information, tracking flight bookings, and streamlining airport security and boarding processes. It enables personalized services and efficient management of customer data.

#### **System Requirements**

- A relational database management system (RDBMS) capable of handling SQL operations, such as Microsoft SQL Server, MySQL, PostgreSQL, or Oracle.

- Appropriate database access permissions for executing queries and managing data.

**External Dependencies**

- Database administration tools such as phpMyAdmin, SQL Server Management Studio, or Oracle SQL Developer for accessing and manipulating the database.

- Connectivity tools and drivers for accessing the database server, like ODBC or JDBC drivers.

**Functions and Structures**

The `PASSENGER` table includes the following columns:

- `PassengerID`: A unique identifier for each passenger.

- `FlightNumber`: References the flight associated with each passenger, linking to the `FLIGHT` table.

- Additional demographic and travel-related columns (the specifics depend on what is available in your actual database).

**Compilation Instructions**

No compilation is required as the data management involves database operations directly.

**Execution Instructions**

1. Connect to the Comfort Airlines database using your database management tool.

2. Execute SQL queries to manage and analyze passenger information. For example, to retrieve passenger details for a specific flight, you could use:

 ```sql

 SELECT \* FROM PASSENGER WHERE FlightNumber = 'FL123';

 ```

3. Use SQL JOIN operations to correlate passenger data with flight schedules, aircraft details, or other relevant tables for comprehensive reporting and analysis.

**Conclusion**

The `PASSENGER` table is a critical asset for Comfort Airlines, offering detailed insights into passenger demographics and travel habits. This table supports a variety of operational and analytical functions, from improving customer service to enhancing operational efficiency. The straightforward structure of the table facilitates easy integration and manipulation within the broader database ecosystem, ensuring that the airline can adapt swiftly to changing business needs and regulatory requirements.

**File #12:** aircrafttable.csv (database)

**Overview**

The AIRCRAFT table houses comprehensive details about the airplanes in the fleet of Comfort Airlines. This table is pivotal for managing aircraft information, scheduling maintenance, tracking aircraft utilization, and planning flight routes based on aircraft capabilities. It serves as a critical resource for ensuring operational safety and efficiency.

**System Requirements**

- A relational database management system (RDBMS) capable of handling SQL queries, such as MySQL, PostgreSQL, Oracle, or Microsoft SQL Server.

- Appropriate permissions to access and manipulate the database.

**External Dependencies**

- SQL management tools such as MySQL Workbench, phpMyAdmin, or Oracle SQL Developer to facilitate database interactions.

- Database connectivity drivers (e.g., ODBC, JDBC) for connecting application software to the database.

**Functions and Structures**

The `AIRCRAFT` table contains the following key columns:

- `TailNumber`: A unique identifier for each aircraft.

- `Speed`: The cruising speed of the aircraft, typically in miles per hour.

- `TotalSeats`: The total number of seats available on the aircraft, which helps in planning capacity and ticket sales.

**Compilation Instructions**

No compilation is required for database operations, as interaction with the table is done through SQL queries and database management interfaces.

**Execution Instructions**

1. Connect to the Comfort Airlines database using a suitable SQL management tool.

2. Use SQL commands to query or update the aircraft information. For example, to view all aircraft sorted by total seats, you could use:

 ```sql

 SELECT \* FROM AIRCRAFT ORDER BY TotalSeats DESC;

 ```

3. Advanced queries might involve joining the AIRCRAFT table with the FLIGHT table to analyze flight and aircraft performance, or to schedule maintenance based on flight hours or usage.

**Conclusion**

The AIRCRAFT table is an essential component of the aviation data infrastructure at Comfort Airlines. It supports a wide range of activities from fleet management to operational planning. Its detailed records help ensure that aircraft are efficiently utilized and maintained, contributing to the overall safety and profitability of the airline. The straightforward schema of the table also allows for easy integration with other operational systems, enhancing data-driven decision-making.