Colby Adams

Eric McAllister

Arch Henderson

May 1, 2016

CSC 315 Final Project Write-up

        For this project, our group was required to create a database that would function like an airline company’s database. In other words, it would provide printed reports for the user that would give information on all pertinent data associated with flights into and out of various airports as well as information on customer ticket requests, payment methods, seating preferences, frequent flyer information, etc.

Our group used the MySQL Workbench program to create this database.  MySQL Workbench is an open source visual database-designing tool. It has a user friendly GUI (Graphical User Interface) designed to make creating databases easier and intuitive. MySQL Workbench integrates SQL development, database design, administration, creation, and maintenance into a single development environment. This allows someone to easily create and maintain a database designed for the MySQL relational database management system.

The database developed for this project was required to be able to retrieve data from various tables and then print that data.  As one would expect, the results of the query must also provide accurate results when the required queries were posed to it. This report will detail the methodology we used to construct the database, the assumptions that were made for each of the queries, the extensions made from the tables, and the normalization efforts we used to reduce redundancy in the database.

        The first table created was the Airport\_T. This table contains the primary identification data for each of the various airports. Each airport was given a unique three character code as the primary key to identify them from each another. The MySQL Workbench table included this three character code, the airport’s name, the city in which the airport was located, the country in which the airport was located, which were all of type varchar(255), and each airport’s respective phone number. For the purposes of this database, the country was set to the United States of America as the default country. This table’s primary key was the Airportcode field.

        The second table created was the AirlineCarrier\_T table. Since airports generally support multiple airlines, it was necessary to differentiate each of the airlines from one another. This table contains both the airline’s name and their respective call sign which is used by the pilots over the radio to identify their airline. Each airlines’ address was also included in the format of its street, city, country, and phone number. Finally, each airline has a unique three character code used by the FAA to help identify flights. This table used the airline’s name as the primary key as specified in the assigment, but the FAA code is used as a the foreign key when referenced in other tables.

        The third table was the FlightPath\_T table. This table was created to identify which airline was providing flights, and where those flights were departing from and to where they were arriving. The table contains the airline code, the flight number which is assigned by each airline to a particular path, the departure airport, and the arrival airport. Each of the flight numbers is a four digit code assigned by the airline however they want them to be implemented.  Each airline uses a unique code for each one of their routes.  For our database’s flight numbers, we used numbers we created ahead of time. In a normal database system, however, this number would most likely have been generated by a number generator. This table had a constraint in that its primary key was both the airline code *and* the flight number.  To avoid data redundancy, the airline code was a foreign key which referenced the AirlineCarrier\_T table for the respective code. The departure and arrival airport data entries were also foreign keys, both of which referenced the Airport\_T table, where it received the codes for the respective data entries.

        The fourth table we created was called the Plane\_T table. Since there are multiple types of planes providing services at any given airport, it was necessary to categorize them. This table contains a ten character tail number which can uniquely identify each individual plane, a VARCHAR field for the plane’s model whose field length could be up to 255 alphanumeric characters, the engine type (turbo-prop or jet), and the number of seats that are currently on that plane.  Since the number of seats can vary from plane to plane and can even be changed out in between flights, we included the number of seats on each plane in the Plane\_T table rather than creating a separate table for planeType that held the number of seats for each type of plane. It was also necessary to clarify the plane’s engine type as this places limitations on which planes can service which flights. Turbo-prop planes are generally smaller, and often have a shorter range when compared to jet planes of similar size. They also tend to be slightly slower than their jet powered counterparts. The largest planes almost always use jet powered engines. They can also carry the largest number of passengers over the longest distances. The seats in the planes were divided into two types: first class/business seats (identified by a three digit number) and economy class seats (identified by a four digit number). This table used the plane’s tail number as the primary key.

        The fifth table was the FlightInstance\_T table. This table was created to identify the planned flights from any given airport and airline on a given date. The table contains the airline code, the flight number, the plane ID, the flight date, the departure time, the arrival time, ticket prices for first class/business seats, ticket prices for economy seats, any reserved economy seats (default at zero), and any reserved business seats (also default at zero). This table’s primary key was constrained to contain the AirlineCode, FlightNumber, and FlightDate. This constraint was necessary since the airline needed both the flight number and its associated date to uniquely identify a particular flightInstance. By subtracting the number of reserved seats from the total number of seats on the plane, the database can compute whether or not any seats are available to be reserved. The plane ID was a foreign key that referenced the Plane\_T table’s tail number. The AirlineCode and FlightNumber entries were also foreign keys, which referenced their respective entries in the FlightPath\_T table.

        The sixth table was the Customer\_T table. This table was created for the people who are customers of the air travel ageny.  They may have registered in the airline’s frequent flyer system for quick access to ticket purchases/reservations. This table contains a customer ID, the customer’s name, an address (in the form of street, city, and country), and a phone number. This table uses the assigned customer ID as the primary key. This table has several other tables, enumerated below, which the airlines use to manage each customer’s individual membership, payment and flight preference details.

        The seventh table was the FrequentFlyerMembership\_T table. This table was created to store the individual customer’s membership information in the airline’s frequent flyer program. This table contains the customer ID, the airline ID, and the miles acquired by the frequent flyer customer. The primary key was constrained to include the customer ID and the airline ID, since the membership of any customer must be associated with an airline’s program.  Each customer can have many FFMemberships, but may only have one Membership per AirlineCarrier.  The customer ID and airline ID are foreign keys, however, that reference the customer ID on Customer\_T and the airline name on AirlineCarrier\_T, respectively. The miles field defaults to zero, until the customer acquires miles.

        The eighth table was the PaymentMethod\_T table. This table was made to store the customer’s standard or desired method of payment. It also contains the customer ID, the payment method, and the card number if applicable. The primary key was constrained to include the customer ID and the preferred payment method. Each Customer can have multiple payment methods. The payment method is left as a VARCHAR to allow different methods to be entered, such as cash or a credit card company name. The card number is only used if the payment method is a credit card, and is otherwise left blank. The customer ID is once again a foreign key that references the Customer\_T table’s customer ID.

        The ninth table was the Preference\_T table. This table was made for storing a customer’s preferred seating arrangement. It contains the customer ID, the preferred seat class, and the details of the seat location that they prefer. The primary key was constrained to the CustomerID and Preference. The preference simply refers to what class the customer prefers, business or economy class. The details entry field pertains specifically to the seat type, such as a window seat or an aisle seat. The CustomerID is again a foreign key that references Customer\_T table’s customer ID.

        The tenth table was the Reservation\_T table. This table manages a customer’s desired reservation by checking if a desired flight is available. This table contained the reservation ID, the customer ID, the payment method, frequent flyer membership should any of them be applicable to the reservation, the customer’s preference for seating, the name of the travel agent who assisted them with the reservation, and a Boolean to confirm whether or not all of the flights have been booked or are avaiable. This table uses the ReservationID as the primary key. The CustomerID is a foreign key that references the Customer\_T table’s customer ID. Rather than using more foreign keys, the PaymentMethod, frequent flyer membership, and preference fields all reference each of their appropriate tables. This methodology works well because each of the entries is directly associated with the CustomerID in their respective tables.  A Boolean is a bit which is often used as conditional check, usually as an if/else statement which will return as either true or false. If all of the flights on the reservation are available, then the Boolean will return as true, and allow the reservation to be processed. If the seat is not available, the Boolean will return as false, and the customer will have to retry their request.

        The eleventh table was the ReservationFlight\_T table. This table was created to store a confirmed reservation made by a customer. The table contained a reservation ID, an airline code, the flight number, the flight date, and an availability Boolean. Each ReservationFlight must belong to a Reservation, but each reservation can have multiple flights.  Therefore the composite primary key of the Reservation ID and the FlightInstanceID was used as the primary key of the reservationflight.  This table contained a constraint on the primary key, combining the ReservationID, AirlineCode, FlightNumber, and FlightDate. This allows a reservation to remain accurate as possible changes are made throughout the various tables. A constrained foreign key of the AirlineCode, FlightNumber, and FlightDate which references the FlightInstance\_T table and their respective entries. The ReservationID is also a foreign key that references the Reservation\_T table’s reservation ID.

        The database was required to be able to return accurate results based on the data requested in each query. Each query would print a self-generated table that contained the requested data. However, most queries will often require data from multiple tables with the same entries. In order to prevent data corruption and to prevent repeated results, certain assumptions about the data had to be established. There were six queries that were made to test the functionality of the database.

        The first query was to print the flight number, departure airport and arrival airport for each flight. For this query, it was assumed that all flight paths are uniquely defined by AirlineCode and FlightNumber, so AirlineCode was included in the results. This query was a SELECT \*, or a select all, and therefore displayed every data field from FlightPath\_T. These results were then ordered by the AirlineCode and the FlightNumber.

        The second query was required to display all of the flight numbers as well as their departure and arrival times for all flights that depart from DFW and arrive at LAX on 12/31/06. This query had an identical assumption as the first query regarding the inclusion of AirlineCode with flight number to uniquely identify each path, therefore AirlineCode was once again included in the results. The query was written to select the AirlineCode, FlightNumber, DepartTime, and ArriveTime from the FlightInstance\_T and FlightPath\_T tables. The displayed results were filtered to include only flights that had the same AirlineCode in both tables that depart from DFW and arrive at LAX on the date 12/31/06.

        The third query was required to print the airline name and the number of flights that it had departing from DFW on 12/31/06. The query would first select an airline’s name and retrieve its respective code. The system would then count all departing flights associated with that airline’s code.  This required a reference to the FlightInstance\_T and FlightPath\_T tables to acquire the actual planned departure times for the specified date. This was repeated for each airline, counting all flights departing. These results were then grouped together by the airline’s name.

        The fourth query was required to print the airport name, the airline name, the number of flights from that airline departing from that airport, and the average cost of the economy class for those flights for every airport and airline. The query would select the AirportName as the “Departing Airport”. It would then count the number of flights associated with each AirlineCarrier by counting the instances of AirlineCode where departing airport matched that airport as “# of Flights”.  The average price of tickets was also computed as “AVG Ticket Price”. The query selected this data from the AirlineCarrier\_T table, the FlightInstance\_T table, the FlightPath\_T table, and the Airport\_T table. The resulting data was then grouped by the departure airport, and then ordered by the airport’s name.

        The fifth query was required to print all flight numbers from DFW to LAX that have at least one available seat and cost less than $500 in economy class. This had a similar assumption as the first query, and so it includes the AirlineCode. It also includes the date of the flight in the event that that particular flight may happen on more than one day. The query would select the AirlineCode, FlightNumber, and FlightDate from the FlightInstance\_T table. The system would then naturally join with the FlightPath\_T table. Finally is uses the Plane\_T table to gather the departure airport and arrival airport. This data was then filtered so that only flights where the departing airport was DFW, and the arrival airport LAX, and where the amount of reserved economy seats was less than the total economy seats. This query also required the PlaneID in order to get the number of seats on each respective plane.

        The sixth query was required to print a table style report with the flight number and the number of available seats for any flight that departs from DFW and arrives at LAX that have at least one seat available. This query was nearly identical to the one for query five, but with two additional columns for available seats and for the ticket price. This query selects the AirlineCode, FlightNumber, FlightDate, the EconomySeats minus ReservedEconomySeats as “Available seats”, and the TicketPriceEcon as “Ticket Price”. The data is pulled from the FlightInstance\_T table and naturally joins to the FlightPath\_T table, along with data from the Plane\_T table. All this data is then filtered according to the query. First, the departing airport must be DFW. Secondly, the arrival airport must be LAX. Thirdly, the plane must have available seats for reservation in economy class. Lastly, the ticket price for the available seats must be less than $500.

