

# FReserve

## A Flight Database Implementation

Created By: Alejandro Caicedo  
Created By: Matthew Chan

## Part 1: Sub-Section b

Our focus for this project is to create a flight reservation system. The goal of the project is to allow users to access the flight reservation system to search for and reserve a spot in the flight of their choosing as well as allowing the user to add or remove flights from the web interface.

The project being submitted is an implementation of a Third Form Normalized (3<sup>rd</sup> Form) database back end with a website front for customer and admin interaction. The web-site's HTML calls python based *Common Gateway Interface* (CGI) scripts for user checking, record fetching, and record updating.

The information its meant to take in most frequently consists of the following in order of frequency:

- Flight Reservation
- Customer Registration / Reward Points Modification
- Flight Creation / Modification / Deletion
- Departure City or Arrival City Addition / Modification / Deletion

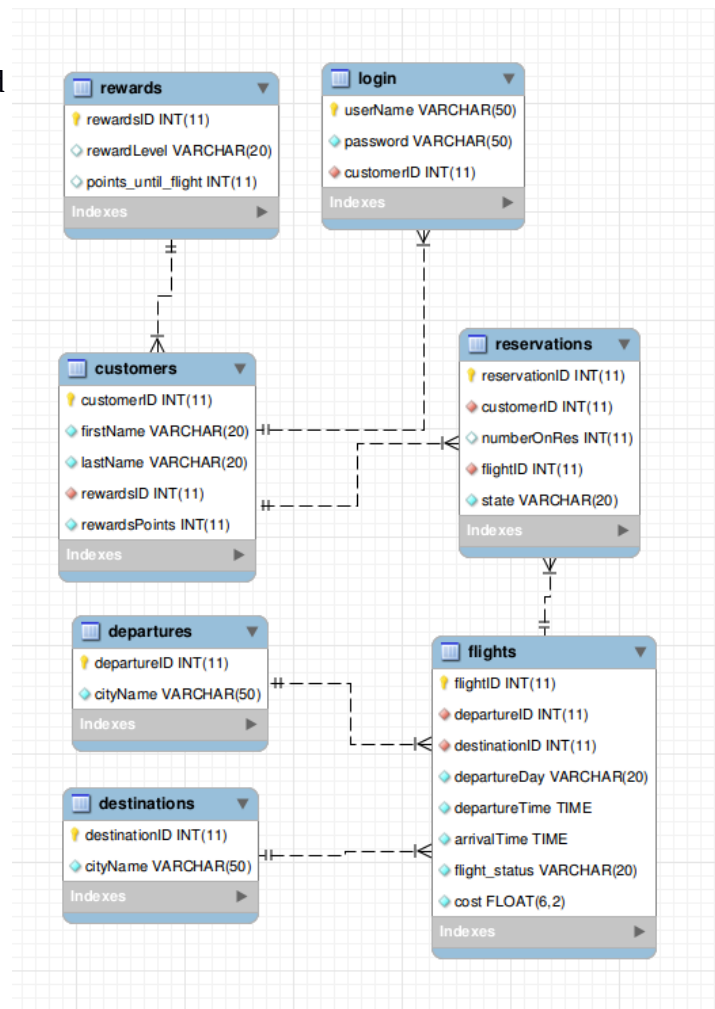
To reserve a flight the DB must know who is reserving the flight as well as which flight the user would like to reserve. As such the Customer Registration and Reward Points information should have the customer's first and last name, their reward points level and amount, as well as how many more points are needed for the customer to get a free flight.

The Flight Creation, Modification, and Deletion requires information about the departure and destination cities. It should also have the departure day and time and when the flight will arrive at its destination.

The above is shown to illustrate the type of information that needs to be processed by this database. The detailed column information will not be expounded upon since that is shown in the *Enhanced Entity Relationship* (EER) diagram to the right, and they are all self evident. Instead the design philosophy of the *database* (DB) will be explained.

We begin with the flight offerings. Before we even have our first customer we must have something to sell them. That is where the flights table comes in. In an effort to keep this as 3<sup>rd</sup> Form as possible, we have split up the information that does not rely solely on the *flight identification number* (flight id) into its own databases. The flight numbers could stay constant, however destination and arrival cities might change, as such those have been moved into their own tables. The flights table has a *many to one* relationship with the reservations table as we would expect.

Now that we have something to offer, we move to users of this system. The target audience is



the consumer who wants to book a flight using this website from the comfort of their own home or on the move. We expect to receive repeat business from these consumers as well, as such we have them create a Customer Account, which immediately places them into our DB allowing us to assign a *customer identification number* (customer id) to them. We have decided to separate the login table from the customer table for performance gains with allowing user logins. The login table has a one to one relationship with the customers table which does not violate 3<sup>rd</sup> form normal form explicitly. This customer id returned from the login will be used to bring up the clients information and be used to track which reservations in our system they have purchased and which are upcoming. Furthermore it allows us to track their rewards points towards their next free flight. As we would expect the customers have a *one to many* relationship with the reservations table.

And now we begin to see how the queries would be formed, by joining the customers, reservations and flights tables we can get all of the relevant flight information for any customer, or see which flights have sold the best, or any other On-line *Analytical Processing* (OLAP) data we may want to drill down into.

The implementation on the web-site front end will use simple queries from the departures and destination tables to limit user options to only those flights offered. When the user logs in, immediately from just a simple query they know their reward points and how much more they would have to spend to get a free flight. With a simple join we would know what flights they have coming up, and if they enjoyed a previous flight, we can query their past flights as well to facilitate re-booking.

## Part 1: Sub-Section c

This design is 3<sup>rd</sup> form normalized. All information in the tables is based only on the primary key of that table and there is no repeated information in any of the tables.