FReserve

A Flight Database Implementation

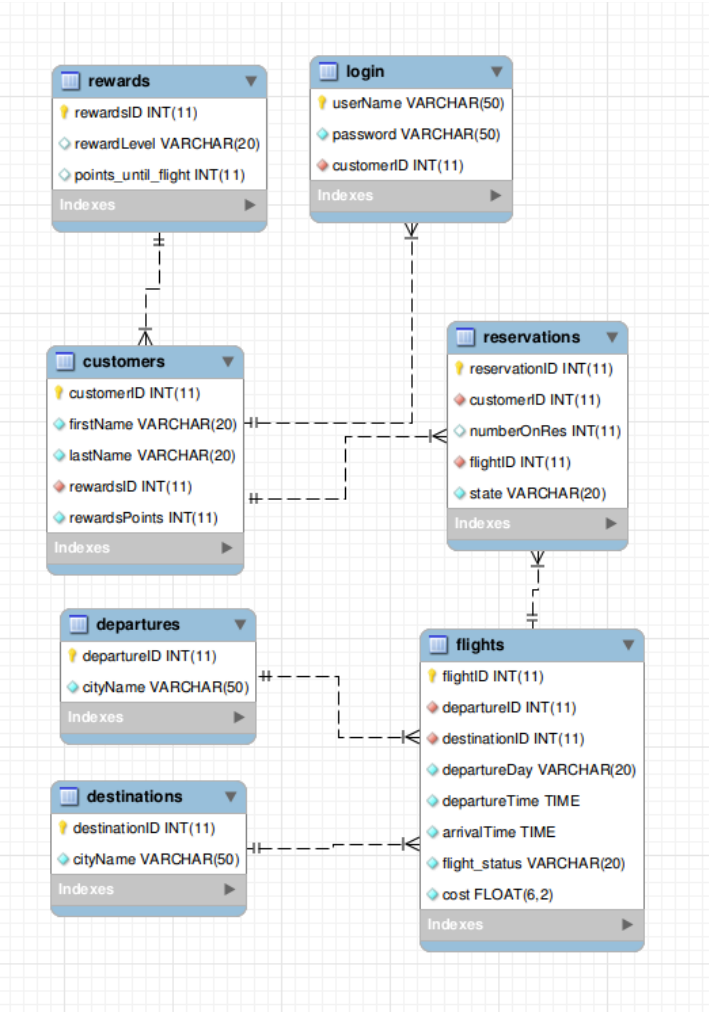
Created By: Alejandro Caicedo
Created By: Matthew Chan

Part 1: Sub-Section b

The project being submitted is an implementation of a Third Form Normalized (3rd 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 / Rewards Points Modification
- Flight Creation / Modification / Deletion
- Departure City or Arrival City Addition / Modification / Deletion

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 below, 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 3rd 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 can 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 3rd 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.

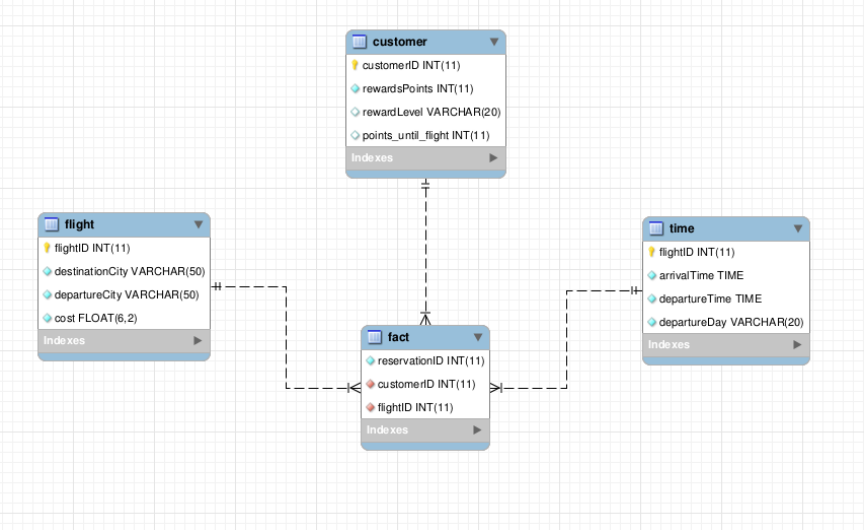
And now we begin to see how the queries would be formed, by joining the customers, reservations and flights tables we can get all of the relevant flight information for any customer, or see which flights have sold the best, or any other On-line *Analytical Processing* (OLAP) data we may want to drill 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 points and how much more they have to spend to get a free flight, with a simple join we 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 3rd 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.

Part 2: The Data Warehouse



In order to promote Business Intelligence, and proliferate good decision making among this company an *OnLine Analytical Processing* OLAP Data Warehouse was implemented. The Data Warehouse would be published using the scheduled execution of a SQL script using a Chron tool or something similar. In this the tables are created from select statements so data publishing becomes simple and the primary/foreign key relations are maintained for ease of query creation. The star diagram is shown below.

The view that was created to assist the sales team is titled: *'destination_MoneySpent_byRewardLvl'* it groups the average amount of money spent by each customers rewards status per destination, allowing them to target their marketing more directly to the grade of customer at each location.

Part 2: Address issues

The TA pointed out that the way we were forming the strings that were used as the SQL queries, i.e. "ABC"+userInput was succeptible to SQL injection and to instead use the "ABC%s"form of string and query creation, passing the parameters to the cursor instead. This advice was propigated to all the CGI scripts