COSC3380: Database ER design, Normalization and Web App
An Airline Database System

Instructor: Carlos Ordonez

1 Introduction

You will develop a web application program that combines SQL statements and a host language which can be executed on a web browser. Your application will feature both transaction processing and query processing, building upon previous homeworks.

You can assume flight information will be ready. The input will be provided by the user in a web form(s), validated and then you will build SQL statements sending them to the DBMS.

2 Program major requirements

1. Execution:

   your program will be tested on a browser, like Chrome or Firefox. Your program will generate and send SQL statements to the DBMS. The app will open in your local computer (e.g. your laptop). The connection will be established from the web server to the database server.

2. Transactions:

   Transactions should update several tables as seen in class. You program should be prepared to handle any errors. You can assume one customer buys up to 5 tickets in total. The ticket can be for the same person or other people, like family/couple.

3. Languages and OS.

   The OS for our web and DBMS servers is Linux (any distro is fine). Since your program will work on the browser it is expected it works well on any OS, but Windows would be the default. The programming language is: Javascript, combined with html. The DBMS is: PostgreSQL.

4. You cannot export or read tables row by row, doing processing outside the DBMS. You cannot do process data outside the DBMS with the host language low-level processing mechanisms (i.e. locking, text files). The bulk of processing must be done with SQL statements, with high-level control in the host language.

2.1 Database ER Design and Programming details

1. Extending ER model

   You need to extend the existing ER model with the following information. If you find any requirement too difficult you can skip it and write a short explanation.

   direct or indirect flights, up to 1 connection (2 connections extra credit). customer id, customer name, customer telephone, customer email, id used for boarding, family/couple/group traveling together;

payment: card number, taxes (as many as you want), discounts, amount in dollars (payment in other coin like euros or pesos optional); boarding: baording time, gate, waitlist, checked bags; arrival: actual arrival time, arrival gate, baggage claim #; flight info: movie y/n, meal y/n.

Optional: multiple airlines, pilot, crew, fuel info, aircraft mechanical checks, etc.

2. Normalization:

   for each table you need to normalize it up to 3NF/BCNF. Notice derived attributes or quick lookup values are acceptable (like price+tax, fare class), but you need to explain why.

3. ER diagram and short project description

   You need to draw a diagram with all tables in your ER database model. You should include all PKs and FKs, but you can omit other attributes. You can draw the diagram by hand or use any diagram tool (e.g. lucidchart). The main design requirement is showing the relationship cardinalities 1:N,M:N.

   Save the ER diagram and your project report in one PDF file: airline-ersummary.pdf.

4. Transaction:

   all changes to the database must be done by transactions, never isolated SQL statements.

   Tables to update: You are supposed to update and insert into tables to manage reservations, boarding and the flight itself. Make your app realistic.

5. Reports

   Your app should provide standard queries to explore flights, in the same manner you use airline websites. Also, include queries from the airline clerk side (e.g. listing passengers on each seat, listing people in waitlist). You have freedom to create queries, but they must provide useful information to find and compare flights. On the other hand, queries should provide useful information to verify status of passengers, flights, airports and so on.

6. Multithreading: not needed. Just process all updates with transactions assuming your program can be used by several customers from different connected computers to the web server.

7. Initial database state:

   zero reservations, flights ready to be booked (created by you), flights available for 2 months in the future (say December 2020, January 2021). Give an idea about available flights to the user in the web app via queries.

8. Your program must be prepared for failures and errors.

9. SQL:

   Create ”transaction.sql” with the main transaction SQL code and ”query.sql” with a list of your main queries. These two files will be checked by the TAs.

3 Program call and output specification

- The user will open the airlineweb.html file. This file will bring up the GUI and will connecto to the DBMS as appropriate.
- If the connection to the DBMS fails, the program must display an error message.

- Output:

  messages and small tables with relevant information on the screen.

4 Grading

Submission: all programs, PDF will be uploaded to a web server (TBA). The database will remain in the same DBMS. All source code, report and documentation must be anonymous. The TAs will provide a folder name, like ZC1TA2 (similar to a reservation code, look at the ticket the last time you flew).

Testing: Your program will be tested by 3 other teams. They will fill an anonymous form with scores covering these aspects: ER design, normalization comprehensiveness of information, database correctly updated, ability to book transactions and reports. Each aspect will receive a score from 1 to 10. Teams will not know who tested their program. Teams should not manually login to the tested team Linux accounts; only via the web app. The TAs will make a quick final test anyway. Therefore, you need to test and retest your program before submission. Also, include a README file.

Demo: You will give a 5 minute demo, pre-recorded. All teams, TAs and the professor will attend.

Sizes: You can work with a toy airline, having a few flights and small airplanes. That is, you can work with small tables.

Score: a well designed database, with a well developed app will receive a high score. Each team will also receive credit for how well they test another team project (within reason). Given the freedom in the design, tables and GUI there is no specific correctness score (other than not failing). We would prefer not to have any resubmissions, but I can consider suggestions.

Code and app originality: It is expected all projects will be different, with different design. However, it is acceptable they resemble some real airline web site.