CS 480
Database Systems
Summer 2018

Project #4: Multi-User Coursemo App

Complete By:	Saturday, August 11th @ 11:59pm
Assignment:	completion of following programming exercise
Policy:	Individual work only, late work *is* accepted (see "Policy" section at end of doc for more details)
Submission:	electronic submission of VS project folders to BB

Overview

You have probably heard of **Venmo**, the digital wallet app that let's you collect and make financial payments. In this exercise we're going to develop a database app called **Coursemo**, that let's you add, drop, and swap course registrations. Example: you are registered for course X, and a friend is registered for course Y. How many times have you wished you could just swap registrations? Well now you can :-)

You'll be given 2 .csv files as input, denoting students and available courses for a single semester. You'll need to design, create, and populate a database that allows students to add, drop, and swap classes. Your design will also need to support a FIFO waitlist for each course; graduate students have an additional requirement to maintain a complete history of all transactions (i.e. every add, drop, and swap).

Input Files

There are two input files, "**students.csv**" and "**courses.csv**". You'll use these files to initially populate your database. The file "students.csv" is a CSV file containing N > 0 rows, where each row consists of last name, first name, and Netid:

```
Hummel,Joe,jhummel2
.
.
.
```

You may assume the **Netid** is unique. Make no other assumptions about the input file. The file "courses.csv" is a CSV file containing M > 0 rows, where each row consists of: department abbreviation, course number, semester, year, CRN, type (lecture or lab), day, time, and class size:

```
CS,107,fall,2018,10634,lecture,TR,0800-0915,22
CS,107,fall,2018,10630,lab,M,0800-0850,6
```

```
CS,107,fall,2018,10631,lab,M,0900-0950,6
CS,107,fall,2018,26611,lab,M,0900-0950,6
CS,107,fall,2018,26613,lab,M,1000-1050,6
CS,107,fall,2018,39412,lab,M,1000-1050,6
CS,107,fall,2018,42028,lab,M,1100-1150,6
CS,109,fall,2018,36425,lecture,MW,1000-1050,34
CS,109,fall,2018,30607,lab,R,0800-0850,6
CS,109,fall,2018,30608,lab,R,0800-0850,6
.
.
.
ENGR,194,fall,2018,41365,lecture,TR,0330-0445,15
IT,202,fall,2018,39823,lecture,M,0430-0545,8
IT,202,fall,2018,39825,lecture,W,0430-0545,8
```

Note that the class sizes are artificially low to enable easier testing of your waitlist feature; otherwise these are the actual classes offered by the CS dept this coming Fall. You may assume the **CRN** is unique. For the **department**, you may assume standard UIC abbreviations: CS, ENGR, IT, MATH, BIOE, etc. The **semester** is always one of following: fall, spring, summer1, or summer2. The **type** is either lecture or lab. The **day** and **time** should be stored as strings, since it can vary widely or represented as "TBA" (to be announced). The **class size** is an integer >= 0. Assume all courses contain 0 enrollments when you first populate the database.

Sample input files are available on the course web page: see Projects, then "project04-files". Best to use the "download" button in the upper-right corner to download to your local machine. Here are direct links:

Courses.csv: https://www.dropbox.com/s/pwv0pj2rsu547s5/courses.csv?dl=0
Students.csv: https://www.dropbox.com/s/3r5kc3nd5fcywwh/students.csv?dl=0

We reserve the right to use different input files when testing your program; solve for the general case within the guidelines of the input files.

Requirements

You need to build and submit two applications, modelled after project 1 and project 3. The first application builds your database based on your design, specified in an SQL DDL file, and then populates the database based on the provided input files.

The second app should be a multi-user app that allows the user to add, drop, and swap classes. Much like the BikeHike app, the students should be displayed (name and netid) along with the available courses (dept, course num, and CRN). When a student is selected, the app should:

1. **Display** student's current courses (enrolled and waitlisted); display dept, course number, and CRN.
2. **Enroll** a student in a course: make sure the student is not already enrolled, and enroll the student if there's room in the class; if the course is full, put student at end of waitlist.
3. **Drop** a student from a course / waitlist: if the student is enrolled, drop them from the course; if there's a waitlist, enroll the first student on the waitlist. If the student is on a waitlist, drop them from the

waitlist (preserving FIFO order of the remaining students).

4. **Swap** a course with another student: allow the current student S to swap their enrollment in course X with another student T who is enrolled in course Y. Make sure S is enrolled in course X, and T is enrolled in course Y. No swapping of waitlist positions, just enrollments.
5. **History** (graduate students only): show the transaction history for the current student. A summary of adds, drops, and swaps.

When a course is selected, the app should:

1. Display **course info**: semester, year, type, day, time, class size, current enrollment, # on waitlist.
2. Display **enrollment**: netid of each student enrolled.
3. Display **waitlist**: netid of each student waiting to enroll, in FIFO.

Additional requirements:

1. Add a "reset" button that resets the database back to its initial state: no enrollments, no waitlists, no history, etc.
2. Use database transactions as necessary to ensure correctness in a multi-user environment.
3. Add a way to simulate and test correctness in a multi-user scenario. Add a delay like we did in project #3, driven by a textbox where the user can specify the delay in milliseconds. Use this delay within the steps of your transactions to delay processing and test correctness. Here's the delay code once again:

```
int timeInMS;
if (System.Int32.TryParse(this.txtTimeInMS.Text, out timeInMS) == true)
  ;
else
  timeInMS = 0;  // no delay:

System.Threading.Thread.Sleep(timeInMS);
```

4. Do not use a "flat" solution where all the database access code is in the UI. Take one or more of the approaches discussed in class: stored procedures, LINQ, data-binding, or ORM. Note that you must go beyond the data access tier (DAT) provided in earlier projects, a DAT alone is not sufficient to meet this requirement. Feel free to mix-and-match the available techniques.

Electronic Submission

First, we want both applications that you created for this assignment. Create a folder named say **P4Both**, and then place copies of your project folders into P4Both. Next, be sure there exists a header comment at the top of your Project 4 main C# source code files ("Program.cs") along the lines of:

```
//
// Multi-user Coursemo application for UIC course registrations.
```

```
//
// <<YOUR NAME HERE>>
// U. of Illinois, Chicago
// CS480, Summer 2018
// Project #4
//
```

To submit, find your combined **P4Both** folder. Create an archive (.zip) of this folder: right-click, Send To, and select "Compressed (zipped) folder". Then submit the resulting .zip file on Blackboard: open "Projects" and submit under "Project 04: Multi-User Coursemo App".

Your program should be readable with proper indentation and naming conventions; commenting is expected where appropriate. You may submit as many times as you want before the due date, but we grade the last version submitted. This implies that if you submit a version before the due date and then another after the due date, we will grade the version submitted after the due date — we will *not* grade both and then give you the better grade. We grade the last one submitted. In general, do not submit after the due date unless you had a non-working program before the due date.

Policy

Late work *is* accepted. You may submit as late as **48** hours after the deadline for a penalty of 10%. After **48** hours (i.e. after Monday, August 13th @ 11:59pm), no submissions will be accepted.

All work is to be done individually — group work is not allowed. While I encourage you to talk to your peers and learn from them (e.g. via Piazza), this interaction must be superficial with regards to all work submitted for grading. This means you *cannot* work in teams, you cannot work side-by-side, you cannot submit someone else's work (partial or complete) as your own. The University's policy is described here:

https://dos.uic.edu/docs/Student%20Disciplinary%20Policy%2017-18%20(FINAL).pdf

In particular, note that you are guilty of academic dishonesty if you extend or receive any kind of unauthorized assistance. Absolutely no transfer of program code between students is permitted (paper or electronic), and you may not solicit code from family, friends, or online forums. Other examples of academic dishonesty include emailing your program to another student, copying-pasting code from the internet, working in a group on a homework assignment, and allowing a tutor, TA, or another individual to write an answer for you. Academic dishonesty is unacceptable, and penalties range from failure to expulsion from the university; cases are handled via the official student conduct process described at http://www.uic.edu/depts/dos/studentconductprocess.shtml.