ASSIGNMENT 3 - 601.315/415/615 - Databases

Due Date: Tuesday November 8, 2022, 3 PM (Baltimore time)

You **are** allowed to work on this assignment with a final project partner (or anyone else in the class), and are encouraged to do so. This not only helps reduce system load, but it helps try out a working relationship with potential final project partners. For this reason you are are encouraged to start early, and not leave the assignment until the time of the due date. In no case should this homework be submitted by groups of more than 2 people. *You MUST write all partners' names on your assignment*, and should turn in 1 assignment per group.

The Assignment:

The goal of Assignment 3 is to implement a number of MySQL stored procedures that function on a course grade database similar to database that would be used in a class like 601.315/415. In addition, you will implement simple web interfaces that can access a database.

The database itself has been defined for you, and can be found in the following SQL file on dbase: `/home/cs415/grades.sql`. Your task is to write the MySQL stored procedures that access and manipulate this database, as well as HTML forms that that interface with the database, and possibly with the MySQL procedures you created The specifications that these procedures/forms must satisfy are found below.

Format of the Database:

The 601.315/415/615 grade database is a simple flat table of the following form:

SSN	LName	FName	Section	HW1	HW2a	HW2b	Midterm	HW3	FExam
9176	Epp	Eric	415	99	79	31	99	119	199
5992	Lin	Linda	415	98	71	29	83	105	171
3774	Adams	Abigail	315	85	63	27	88	112	180
1212	Osborne	Danny	315	29	31	12	66	61	106
4198	Wilson	Amanda	315	84	73	27	87	115	172
1006	Nielsen	Bridget	415	93	76	28	95	111	184
8211	Clinton	Chelsea	415	100	80	32	100	120	200
1180	Quayle	Jonathan	315	50	40	16	55	68	181
0001	TOTAL	POINTS	415	100	80	32	100	120	200
0002	WEIGHT	OFSCORE	415	.10	.10	.05	.25	.10	.40

This relation (called *rawscores*) contains the raw scores on individual assignments, along with the names and section numbers of each student and the last 4 digits of their social security number (which is assumed to be unique and serves as the primary key).

The problem has been made more complex in that the **total points** possible for each assignment is stored as part of the same table, with the special SSN *0001*. Unlike in the real 601.315/415/615, the total points possible for 315 and 415/615 are the same. Also, the weight to be given to each assignment (as a percentage summing to 1) is given under the special SSN *0002*.

Note that in order to simplify your MySQL coding, you *are* allowed to create views called `TotalPoints` and `Weights` that have the same attribute names as in the full table,

but contain only 1 special tuple (0001 and 0002 respectively) with just the total points or weights. This can avoid the necessity to extract these values in an embedded query.

Finally, there is a single additional relation called *Passwords* with a single attribute called *CurPasswords*. Although the relation normally only has one tuple (the single current password), it may potentially have multiple tuples, all of which are valid current passwords.

Passwords:

CurPasswords
OpenSesame
GuessMe
ImTheTA

What To Do:

You should write MySQL procedures and HTML interfaces that provide the following functionality:

(a) **Print a single student's raw scores:**

You should write a MySQL stored procedure *ShowRawScores* that takes a single argument (SSN) and prints out the tuple of values in the *rawscores* table that correspond to that SSN. Note that the only security mechanism here is that the user must know the SSN to have access to a student's scores.

(b) **Print a single student's percentage scores and weighted average:**

You should write a MySQL stored procedure called *ShowPercentages* that takes a single argument (SSN) and prints out the tuple of values in the *rawscores* table that correspond to that SSN, but where each score has been divided by the total points for that assignment and multiplied by 100, yielding a percentage value.

This procedure should then compute the weighted average of all the scores for the student, using the relative weights given in the tuple with SSN = *0002*, as shown above. For example, this average would be computed in a formula like:

```
(score.hw1 * (1/totpts.hw1) * weight.hw1 +
 score.hw2a * (1/totpts.hw2a) * weight.hw2a +
 score.hw2b * (1/totpts.hw2b) * weight.hw2b +
 score.midterm * (1/totpts.midterm) * weight.midterm + .....)
```

Note that you can simplify this formula by creating additional views such as `WtdPts`, which contains the precomputed products of (1/totpts.hwi) * weight.hwi.

The output for this weighted average should be a separate SELECT statement stating "The cumulative course average for FName LName is CumAvg", where FName and LName correspond to the provided SSN, and CumAvg is the value computed as shown above.

(c) **Print a full table of the raw class scores**

You should write a MySQL procedure *AllRawScores* that prints out the basic full *rawscores* table, excluding the totalpoints and weight tuples. You should sort by section number first, last name and then first name.

The procedure should take a single string argument called *password*, which is a system access password. This procedure should first check that the password provided appears in the table *Passwords* to prevent unauthorized access. Needless to say, this check should occur *before* printing the table. If the password isn't in the table of current passwords, the procedure should instead print an appropriate error message.

(d) **Print a full table of the percentage scores and weighted total**

You should write a MySQL procedure *AllPercentages* that performs a similar task to *ShowPercentages* in computing the percentage values for each assignment and a weighted average of them. However, *AllPercentages* should print one such line for *every student*, sorted by the section (315/415/615) first and the weighted average second.

AllPercentages should take and handle a single *password* argument just like *AllRawScores*.

(e) **Compute aggregate statistics on the tables** (*601.415/615 only*)

You should write a MySQL procedure *Stats* that is a modification of *AllPercentages* and prints the following additional aggregate information below the primary table: the mean of the percentage scores, the minimum percentage scores, the maximum percentage scores, and the standard deviation of the percentage scores. Those statistics should be calculated for each individual assignment, as well as for the cumulative average. Each of those four statistics (mean, min, max, and std. dev.) should get its own one-line table. Each of these four additional tables should have the assignment attribute labels (hw1, hw2a, etc) acting as headers, in addition to an extra header titled "Statistic" whose value in the sole entry is one of "Mean"/"Minimum"/"Maximum" and "Std. Dev."

For some extra credit, these aggregate statistics should be computed separately for the two sections (315 and 415), in which case you are also required to separate the original table into two tables, one for each section.

(f) **Write a procedure to change student scores** (*601.415/615 only*)

You should write a MySQL procedure *ChangeScores* that takes 4 arguments (password, SSN, AssignmentName and NewScore). It should verify the password as before, and then replace the current score for the assignment called 'AssignmentName' for the given SSN with the NewScore. Doing this in a general way can be tricky, but you are allowed to use brute force (i.e. a long code segment testing for each of the assignment name labels.) As before, the password should be verified before replacing the score.

In addition to actually changing the score, the procedure should print out the full raw score tuple for that student before *and* after the change. testing for each condition).

(g) **Write simple PHP/MySQL Interfaces for Updating a Student's Scores and for Displaying Results** *(everyone)*

Details of this task and examples will be given and demonstrated in class and communicated via an e-mail to the class. Notes:

Notes:

- In all of your procedures, you should guard for errors (e.g. invalid SSN or password) and should also print appropriate error messages to the user. Improper handling of errors will be penalized.
- All those procedures should be written in a text editor and formatted clearly in a single text file called `DavidYarowsky_HW3.sql`, where you you use your own first and last names instead.
- This assignment is actually shorter than it might appear to you. A considerable percentage of the code for procedure declarations can be shared between the different parts of the assignment.

What To Hand In:

You should write all your procedures in a text editor, formatted clearly. Submit this file where the procedures using gradescope, equivalent to HW2. as in HW2.

In addition, you are required to submit any files created for item (g) above using the HW2 gradescope instructions.