CS 377: Database Systems
Final Project: Course Evaluation System

Due: April 30th at 11:59 PM on *Canvas*

Overview

The Mathematics and Computer Science (MathCS) Department wants to hire you to create a web application to take our course evaluations into the 21st century. The course evaluation system allows the users to input similar information that is gathered from the scantron form (from Office of Undergraduate Education) and the written course evaluations (from MathCS)[1]. The web application also allows the user to look up historical data on courses as well as instructors.

This project "walks" you through the process of building both the backend (database design, creation, and population) and the frontend, which will use PHP to provide a dynamic website. A few things of note:

- Your final project grade will be based purely on functionality (whether or not it meets the specified requirements) and not on aesthetics. However, aesthetics will likely factor into the competition for the top prize.
- The work must be your own. Given the amount of freedom that exists, each project should look (in both frontend + backend) unique. We will run similarity software (MOSS) on the final project.
- Your final project need not complete the bonus question to be eligible for the top prize (although it may help you get nominated).
- Although you will have your own webserver on the Google Cloud Platform, we will be re-creating your application on our own local Google Cloud server. This means that we *will add/delete tuples* from the data files you submit.
- You want to avoid hard-coding information into your web application (it should read dynamically from the database).

1 Database Design (15 points)

Design an ER diagram for the website based on the following requirements:

- A class is assigned a unique class number, a course name, section number, semester and year. For example, the database course you are currently taking has the class number 1859, the course name CS377, section 1, Spring semester, and 2018.
- Each course must be taught by an instructor.
- An instructor has a first name, a last name, and a unique instructor id.
- A student will be assigned a unique (and seemingly random) integer identifier. This allows their response to be anonymized.

[1] Due to the sheer amount of questions on both, we simplified and merged some of the questions.

- A student can take any number of classes.
- A student can evaluate any number of classes provided they have taken the class. However, the student cannot evaluate the same class more than once.
- A class evaluation must have at least one question and can have any number of questions.
- A question can take on the following four different types:
  - Agree/Disagree: 5 choice question where the choices are (1) strongly agree, (2) agree, (3) neutral, (4) disagree, and (5) strongly disagree. Example Q: "I would recommend the course for CS students."
  - Multiple Choice: Question-specific choices that need to be defined. Example Q: "What is your reason for taking the course? (1) requirement for major, (2) prerequisite, (3) pre-major requirement, and (4) interested".
  - 1-10: On a scale of 1-10, with 1 being the lowest and 10 being the highest, how ... was the class? Example Q: "How well organized was the class?"
  - Open-Ended: Any textual response is allowed. Example Q: "Comment on the strength of the course."
- Each class can share any number of questions with other classes (for example, the recommendation question from the agree/disagree is present for all CS courses).

Justify any additional assumptions you make about your project.

2 Relational Model Creation (10 points)

Design the relational model from your ER diagram. Make sure the key attributes are underlined and the foreign keys arrows are drawn to the correct place.

3 Database Normalization (15 points)

What are functional dependencies that you think should hold in your application? Normalize your relational schema to meet 3NF based on your functional dependencies.

4 SQL Database Creation (15 points)

Create two MySQL files:

- `create-tables.sql`: Create the tables (implement the relational schema from the previous part)
- `drop-database.sql`: Clean out your database (should remove any semblance from the MySQL server)

Both you and the TAs will use this file to build / re-build your database. It is important to assume that you have a "clean" MySQL instance when we run `create-tables.sql`.

5 Database Population (15 points)

You will use the bulk load option to populate your database. Your first task is to take the data that we've provided and convert it into a format that is consistent with your relational schema. We have provided the data file `evaluations.csv`, where each line contains the responses to the course evaluation for a student[2]. Inside the data file, each line will represent a tuple with commas separating the different attribute values. The file is formatted such that:

- The type of question is specified in the header row in parenthesis.
- There are at least 2 entries for every class number. If you see that none of the tuples has a value for a "question", you can assume that the class evaluation did not have this question.
- For the multiple choice questions, all the possible choices are contained within the spreadsheet.
- Open ended questions need not have a textual response.
- Students without any values in the questions are students that either have not filled out a questionnaire or are currently taking the class.

Take the data in the `evaluations.csv` file and make new `*.csv` files such that each relation has its own data file (`<relation-name>.csv`). As an example for the company database, the `department.csv` file will have 3 rows:

```
'Research', 5, '333445555', '1988-05-22'
'Administration', 4, '987654321', '1995-01-01'
'Headquarters', 1, '888665555', '1981-06-19'
```

You are welcome to add more evaluations and questions, but it must contain all the information provided as a minimum.

Create another SQL file called `populate-tables.sql` that will populate all your relational tables using the bulk import capability. Here is an example of the bulk load command in MySQL[3]:

```
LOAD DATA LOCAL INFILE 'department.csv' INTO TABLE department FIELDS TERMINATED BY ',';
```

6 Google Cloud Server Setup (15 points)

You will set up a virtual machine (VM) on the Google Cloud Platform that will serve as the webserver and sandbox for your final project. Through the Google Cloud Platform Education Grants program, each student has been approved for a $50.00 credit. Assuming you've set up the VM based on the specs, you **will not have to pay anything for this project**. You will be responsible for stopping it in the middle of May to avoid being charged.

1. Request a Google Cloud coupon for the course from this URL: `https://goo.gl/PEBLTS`. You will be asked for a name and email address (emory.edu address). A confirmation email will be sent to you with a coupon code. Please make sure you request **ONLY ONE** coupon – there is a maximum number of coupons that were assigned to the course.

[2]The responses have been randomly generated and may not reflect any student's actual opinion.

[3]When you connect to your MySQL server, you may need to enable the local infile option. The command should be `mysql -u cs377 -p --local-infile`

2. Setup a VM instance with the following specifications:
   - Zone: Select any one of the us-east1 zones
   - Machine type: The default should be the 1 vCPU, 3.75GB memory with automatic CPU platform.
   - Boot disk: Ubuntu 16.04 LTS (you are free to select any of the other free ones, but the instructions we provide are for Ubuntu 16.04 LTS)
   - Firewall: Select allow HTTP and HTTPS traffic.

   The estimated cost should not exceed $25.00 a month.
3. Connect to your VM instance using SSH. You have two options:
   (a) Use the Google SSH interface (link to instructions).
   (b) Use the command-line SSH by connecting via the external IP (link to instructions).
4. Install the LAMP stack on your VM machine. Digital ocean provides a set of instructions for installing LAMP stack on an Ubuntu 16.04 LTS machine (link to instructions). For additional resources, Lynda has a video on the LAMP stack installation and configuration process (Lynda website).
5. Set up your MySQL server to have the user 'cs377' and password 'cs377_s18'. This should be *EXACTLY THE SAME* as the course server `cs377db.mathcs.emory.edu`. Make sure that you grant access to the cs377 user to your database schema. Your PHP scripts **MUST USE** the cs377 user so that we can replicate your web application on our Google Cloud Platform server.
6. Load the database file from the creation and population steps.

7 User Input Form(s) (35 points)

Design a webpage(s) that allows a student to enter in their class evaluation for a single class. The webpage (or webpages) should meet the following requirements:

- Verify the user (unique identifier) is a valid one (in the system). This should not be in the form of a drop-down menu as this will not scale well to a large number of users. Consider that the MathCS department has at least 600+ students take a MathCS course and trying to find 1 out of 600 will not be a great user experience.
- Verify the user is allowed to provide a class evaluation for the course.
- Dynamically loads the evaluation questions pertinent to each course.
- Ensure all the questions have an associated response.
- Allow only one evaluation per student (and it should be the first one submitted).

You have freedom of choice in terms of how you format the course evaluation form and the order in which you present the questions. If the user provides incorrect or invalid input, your webpage should be able to specify the error so the student can fix the error.

8 Class Report (35 + 45 = 80 points)

Design webpages that provides reports for each course. It will be tailored specifically for each user.

8.1 Student View of Class

Design a webpage that will allow the user to view summary statistics of a course. The webpage should allow the user to choose the course name and the instructor and then return aggregate statistics for all the questions except the free-text ones. For example, Dr. Ho has previously taught 2 versions of CS377 so your webpage should return aggregate statistics from the evaluations from all the students who have taken CS377 with her. The aggregate statistics for each question should be calculated as such:

- 1-10: Average (mean score) of all the responses.
- Agree/Disagree: Percentage of students who responded agree or strongly agreed.
- Multiple-choice: Broken down by percentages for each category.

8.2 Faculty View of Class

Design a webpage that will allow the faculty to view his/her statistics for a single class (e.g., CS377 in Spring 2017 for Dr. Ho). The faculty page will contain a breakdown of each of the responses:

- 1-10: The total number of counts for each response and the median. For example, if there are 4 7's, 13 8's, and 22 9's for a question, this will be shown as well as a median of 9.
- Agree/Disagree: The breakdown for each category in terms of the number of counts.
- Multiple-choice: Broken down by counts for each category.
- Text: All the responses are made available.

Bonus (20 points)

Design a Department Chair view for the class. The chair view should perform the following things:

- Select a question that he/she is interested in analyzing further.
- Given the question of interest, show the overall statistics for the entire department broken down by semester and year (similar to the faculty view except that now it is for all the course offerings).
- Given the question of interest, identify the classes in your department (e.g., CS) that are doing poorly in comparison with the other classes offered by the department. Define poorly as a score of mean - 1.5 $\times$ standard deviation for the 1-10 questions and similarly for the agree/disagree where strongly agree is represented by the value 5, and strongly disagree is represented by 1. For example, if the CS department chair wanted to identify which classes

were poorly organized, he/she would select that question. You would calculate the mean and standard deviation based on all the CS offerings, and then find any classes where the question achieved an average score worse than the specific value.

- Given the question of interest, identify instructors within the department who are performing poorly (same definition of poor as above).

Submission Instructions

All the work should be submitted electronically *in a single zip file* (e.g., FinalProject.zip) via *Canvas*. Do not nest another layer of folders in your `zip` file. The contents of the zip file must contain and adhere to the following guidelines:

- A `README.txt` file that contains your VM machine's external IP and the honor code where you signed your name, otherwise points will be deducted.

  ```
  Server IP: 35.231.90.6
  /* THIS CODE IS MY OWN WORK.
  IT WAS WRITTEN WITHOUT CONSULTING CODE WRITTEN BY OTHER STUDENTS.
  _Your_Name_Here_ */
  ```

- ER design (pdf, jpeg, or png file) that is appropriately named.
- Relational schema (pdf, jpeg, or png file) that is appropriately named.
- Database normalization work (pdf, jpeg, or png file) that is appropriately named.
- `create-tables.sql` and `drop-database.sql`
- Relation data files: `<relation-name>.csv`
- `populate-tables.sql`
- `*.php` files for your web application.

We will load all your files onto our own Google Cloud server instance. To test your web application, we will add/delete some tuples to ensure that your program dynamically adapts to the database instance while still adhering to the requirements we have specified. If we are unable to successfully recreate your project on our server, we will use the link provided in the `README.txt` file. However, if we are forced to grade your project from your google server, the **maximum points you will get is 180**.