Project 1 Part B

Due Wednesday, 10/19/2016 by 11:00pm

Change history

Overview

In this part of the project you will primarily learn how to use MySQL and how to access it from the Apache server. You will create a few tables to contain information about movies and actors, load them with (real) data we are providing, and run a few queries to get familiar with MySQL. You will also have to create a php page that allows users to query your MySQL database using HTML forms. Finally, you will also need to identify a few constraints that the tables should satisfy and enforce them for the tables.

Learn MySQL Command-Line Interface

Start with reading our Introduction to MySQL. Try the examples in the document, and experiment with the help command to get farmiliar with the MySQL command-line interface.

Description of Our Dataset and Table Schema

The dataset needed for Project 1 is located in the `~/data/` directory of the virtual machine. (The data files can also be obtained by downloading data.zip and unzipping it to the directory with the `wget` and `unzip` commands.) Browse the data files with `more [filename]` to see how fields are enclosed and delimited, how NULL values are represented, and so on. (In case you are concerned about the encoding of our dataset, you may assume the files are encoded in UTF-8, even if some characters do not show correctly. Also make sure to use the default encoding, UTF-8, for all character fields in your tables.)

Here is a brief description of the tables that you will have to create for this project and the data files that contain the tuples for the tables.

The Movie table: This table describes information regarding movies in the database. It specifies an identification number unique to each movie, the title of the movie, the year the movie was released, the MPAA rating given to the movie, and the production company that produced the movie. The schema of the Movie table is given as follows:

```
Movie(id, title, year, rating, company)
```

Name	Type	Description
id	INT	Movie ID
title	VARCHAR(100)	Movie title
year	INT	Release year
rating	VARCHAR(10)	MPAA rating
company	VARCHAR(50)	Production company

The load file for the table is movie.del in the zip file.

The Actor table: This table describes information regarding actors and actresses of movies. It specifies an identification number unique to all people (which is shared between actors and directors), the last name of the person, the first name of the person, the sex of the person, the date of birth of the person, and the date of death of the person if applicable. The schema of the Actor table is given as follow:

```
Actor(id, last, first, sex, dob, dod)
```

Name	Type	Description

id	INT	Actor ID
last	VARCHAR(20)	Last name
first	VARCHAR(20)	First name
sex	VARCHAR(6)	Sex of the actor
dob	DATE	Date of birth
dod	DATE	Date of death

There are three load files for the table: actor1.del, actor2.del, and actor3.del. You will have to load each file only once to the table.

The Director table: It describes information regarding directors of movies. It specifies an identification number of the director, the last name of the director, the first name of the director, the date of birth of the director, and the date of death to the director if applicable. The schema of the Director table is given as follow:

```
Director(id, last, first, dob, dod)
```

Name	Type	Description
id	INT	Director ID
last	VARCHAR(20)	Last name
first	VARCHAR(20)	First name
dob	DATE	Date of birth
dod	DATE	Date of death

Note that the ID is unique to all people (which is shared between actors and directors). That is, if a person is both an actor and a director, the person will have the same ID both in the Actor and the Director table.

The load file for the table is director.del.

The MovieGenre table: It describes information regarding the genre of movies. It specifies the identification number of a movie, and the genre of that movie. The schema of the MovieGenre table is given as follow:

```
MovieGenre(mid, genre)
```

Name	Type	Description
mid	INT	Movie ID
genre	VARCHAR(20)	Movie genre

The load file for the table is moviegenre.del.

The MovieDirector table: It describes the information regarding the movie and the director of that movie. It specifies the identification number of a movie, and the identification number of the director of that movie. The schema of the MovieDirector table is given as follow:

```
MovieDirector(mid, did)
```

Name	Type	Description
mid	INT	Movie ID
did	INT	Director ID

The load file for the table is moviedirector.del.

The MovieActor table: It describes information regarding the movie and the actor/actress of that movie. It specifies the identification number of a movie, and the identification number of the actor/actress of that movie. The schema of the MovieActor table is given as follow:

```
MovieActor(mid, aid, role)
```

Name	Type	Description
mid	INT	Movie ID
aid	INT	Actor ID
role	VARCHAR(50)	Actor role in movie

The load files for the table are movieactor1.del and movieactor2.del. You will have to load each file only once.

The Review table: Later in Project 1C, you will create a Web interface where the users of your system can add "reviews on a movie (similarly to Amazon product reviews). The Review table stores the reviews added in by the users in the following schema:

```
Review(name, time, mid, rating, comment)
```

Name	Type	Description
name	VARCHAR(20)	Reviewer name
time	TIMESTAMP	Review time
mid	INT	Movie ID
rating	INT	Review rating
comment	VARCHAR(500)	Reviewer comment

Each tuple specifies the name of the reviewer, the timestamp of the review, the movie id, the rating that the reviewer gave the movie (i.e., x out of 5), and additional comments the reviewer gave about the movie.

Since this data will be added by your users, there is no load file.

The MaxPersonID and MaxMovieID Tables: Later in Project 1C, you will also construct a Web interface where users can add new actor, director or movie information to the database. Once a user adds a new actor/director, your system should assign a new ID to the actor/director and insert a tuple to the Actor/Director table. Similarly, your system should assign a new ID to a new movie.

In order to assign a new ID to, say, an actor/director, your system has to remember what was the largest ID that it assigned to a person in the last insertion. The MaxPersonID table is used for this purpose, which has the following schema:

```
MaxPersonID(id)
```

Name	Type	Description
id	INT	Max ID assigned to all persons

MaxPersonID is a one-tuple, one-attribute table which maintains the largest ID number that the system has assigned to a person so far. Whenever a new actor/director is inserted, the system looks up this table, increases the ID value of the tuple by one, and assigns the increased ID value to the new actor/director. You may consider this MaxPersonID table as a "persistent variable" that remembers its value even after your program stops.

The MaxMovieID is used similarly to assign a new ID to a new movie inserted by the user. As a new movie is added to the database, the entry in this table is incremented and assigned to the new movie. The schema of the MaxMovieID table is given as follow:

```
MaxMovieID(id)
```

Name	Type	Description
id	INT	Max ID assigned to all movies

You will have to create these two tables, and insert the tuple (69000) to MaxPersonID table and the tuple (4750) into the MaxMovieID table.

Loading and Querying the Dataset

Now that you understand our provided dataset, it is time to create the tables in MySQL. Write a SQL script named `create.sql` that creates all tables above. If you are not sure how to create a table in MySQL or how to write and run a SQL script, please read our Introduction to MySQL again. In creating the tables, please make sure that their schemas are ***exactly as we specified above including their cases***, because having the same schema among all students is essential for the correct functioning of Part C of this project.

Note that in the VMware image, we have created two databases in MySQL: `TEST` and `CS143` (note these database names are case sensitive). The `CS143` database is your "production" database, meant for use in the final versions of your code. The `TEST` database is for any experimentation and for use during development and debugging. We have created two MySQL users for the database. For your project work, use the MySQL user `cs143` (no password), which has full access to the `CS143` and `TEST` databases. The user `root` with password `password` has full unrestricted access to everything and should be used only for special adminstrative operations, like creating new users and databases, etc.

Once you the finish writing `create.sql`, create the tables in the `CS143` database, using a command like

```
% mysql CS143 < create.sql
```

in Ubuntu shell (or using `SOURCE` command in `mysql`).

Notes on CR/LF issue: If your host OS is Windows, you need to pay special attention to how each line of a text file ends. By convention, Windows uses a pair of CR (carriage return) and LF (line feed) characters to terminate lines. On the other hand, Unix (including Linux and Mac OS X) uses only a LF character. Therefore, problems arise when you are feeding a text file generated from a Windows program to a program running in our guest OS (such as `mysql`). Since the end of the line of the input file is different from what the tools expect, you may encounter unexpected behavior from these tools. If you encounter this problem, you may want to run the `dos2unix` command from the guest OS on your Windows-generated text file. This command converts CR and LF at the end of each line in the input file to just LF. Type `dos2unix --help` to learn how to use this command.

Now that you have created tables, it is time to load data. Create a MySQL script, named `load.sql`, that loads all our provided data into the created tables. Again, if you are not sure how to load data into tables, read our Introduction to MySQL. Read MySQL Reference on LOAD DATA as well to learn more detailed options available for MySQL LOAD command. Assume that all data files are available at `~/data/` directory, like `~/data/director.del` and write your script accordingly. Make sure that the double quotes enclosing string fields in the load files are removed when inserted into a table. Finally, make sure your script inserts the needed tuples for MaxPersonID and MaxMovieID tables as well. Load all your data into the tables by running the command

```
% mysql CS143 < load.sql
```

in your Ubuntu shell.

Now explore the dataset by running the SELECT statements that answer the following questions from your database:

- Give me the names of all the actors in the movie 'Die Another Day'. Please also make sure actor names are in this format: **<firstname> <lastname> (seperated by single space, **very important**)**.
- Give me the count of all the actors who acted in multiple movies.

Feel free to run other interesting queries and learn about the data. Create a MySQL script named `queries.sql` that contains:

- the two SELECT statements above (as the first two queries)
- another query that you came up with

Add an English description of each query as a comment to the script. You can use the `--` tag to make comments within your script.

Remeber to use the `cs143` database when you load the data and run queries. Otherwise, you will get error messages. Pay attention to CR/LF issue and run `dos2unix` command on your script files if necessary.

Building a Web Query Interface

Now that all your data has been loaded, you have to create a php page that allows users to type in a SQL SELECT statement in a text input box and submit the query through a Web browser. Given a user query, your php code needs to execute the query on MySQL, return the results as the results page in an HTML table. The query result should be presented in a legible form. The input box and the submit button should be preserved in the result page to allow for additional querying. To give you a more concrete idea on what is expected for this part of the project, we made a demo site available at HERE. Visit the site, type in an SQL query, like "SELECT * FROM Actor WHERE id < 20", and see what result you get. Of course, your implementation does not have to be identical to the demo site. It is only to give you a general idea on what is expected from this part. To make this part of project manageable, for this part (and this part only) you may assume that users will always issue correct SELECT queries and all user inputs can be trusted.

You have to use a TEXTAREA input box (with the name "query") as the mean of input from your user. The input box should be large enough to allow for a decent size query by the user. (Different types of HTML input boxes were explained in PHP input handling tutorial linked in Project 1A.) Once the user submits her query in the box, it should be sent back to your php page, which displays the result of the query. For information on how you can interact with MySQL from PHP (e.g., issuing a query to MySQL and retrieving results), see our brief tutorial on MySQL and PHP. MySQL functions in PHP page has the full list of MySQL functions available in PHP. In particular, there exist PHP functions to obtain the number of columns in the results from MySQL and the names of those columns.

Please make sure that all URLs in your php page is ***relative***, so that your code does not break when we grade them. Make sure that you connect to the database with the **username "cs143" and empty password and use the database `cs143`**. Finally, name your php file as `query.php`.

Enforcing Data Integrity

If the data in your MovieDatabase system at a given point in time represents a correct state of the real world, a number of constraints are expected to hold. Now it is time to identify the constraints that you believe should hold in a correct MovieDatabase (i.e., in a database that represents a viable state of the real world). To get you started, here are a few possible examples, some of which depend on a particular schema:

- Every movie has a unique identification number.
- Every movie must have a title.
- Every actor must have a date of birth.

Write down in English a list of constraints that you can think of that your database should satisfy. Consider only constraints that go beyond what is already enforced in the schema (e.g., you do not need to specify that ratings are integers).

Of course, not every real-world constraint can be expressed in SQL. To learn about constraint support in MySQL, read our short tutorial on Constraints in MySQL.

Now From the constraints that you listed before, identify the ones that can be enforced using either primary key constraints, referential integrity (foreign key) constraints, or CHECK constraints and specify them in SQL. In order to get the full credit, you should specify at least

- Three primary key constraints
- Six referential integrity constraints, and
- Three CHECK constraints

and express them in SQL.

As we explained in the tutorial, note that MySQL supports several "storage engines" for storing tables into the disk. The default MyISAM engine, unfortunately, does not support referential integrity constraints. For this project you will have to use the InnoDB storage engine, which is the only one to support foreign keys. You can make a table to use the InnoDB storage engine by adding the clause "`ENGINE = INNODB`" at the end of the `CREATE TABLE` statement. Also note that MySQL does not support CHECK constraints, so even if you add CHECK constraints to tables, MySQL simply ignores them. Therefore, for the three CHECK constraints, just write the constraints pretending that MySQL supports them.

Modify your `create.sql` script, so that all tables use the InnoDB storage engine and the 12 constraints that you identified above are added to the database. The simplest way to achieve this is to change your `CREATE TABLE` statements in the script to include the constraint definitions and `ENGINE = INNODB`. Once you modify your script, you can simply drop all existing tables (using `DROP TABLE` command) and run the `create.sql` and `load.sql` scripts again to recreate and load the tables and add the constraints. For each constraint that you have added to `create.sql`, write down its English description as a comment.

Once all constraints are added to your database, for each constraint, demonstrate at least one database modification that violates it, and explain why the operation violates the constraint. Create another MySQL script named `violate.sql` that contains all such modification statements. For each statement, explain what constraint it violates in detail as a comment in the script. Again, for the three CHECK constraints, pretend that MySQL supports them and write the modification statements that will violate them if MySQL supports them.

Submission Instruction

Preparing Your Submission

Please create **a folder named with your UID**, put all your files into the folder, then compress this folder into a single zip file called "P1B.zip". That is, the zip file should have the following structure.

```
P1B.zip
 |
 +- Folder named with Your UID, like "904200000" (without quotes)
     |
     +- readme.txt
     |
     +- team.txt
     |
     +- create.sql
     |
     +- load.sql
     |
     +- queries.sql
     |
     +- query.php
     |
     +- violate.sql
```

Please note that the **file names are case sensitive**, so you should use the exact same cases for the file names. (For team work, only the submitter's UID is needed to name the folder.) Here is more detailed description of each file to be included in the zip file:

- `readme.txt`: Readme File

- **`team.txt`**: A plain-text file (no word or PDF, please) that contains the UID(s) of every member of your team. If you work alone, just write your UID (e.g. 904200000).
  If you work with a partner, write both UIDs separated by a comma (e.g. 904200000, 904200001). **Do not include any other content in this file!**
- `create.sql`: The MySQL script that you used to generate the tables with the appropriate constraints. Please make sure **the created tables follow the schema exactly as we specified**. Be extra careful with case sensitivity. For each constraint, write an English description of the constraint that you are enforcing as comments in the script. Make sure your script runs without any error when you execute the command `mysql CS143 < create.sql`.
- `load.sql`: The MySQL script that loads all tuples into the tables. Please make sure that **the data files are loaded from `~/data/` directory**, so that your script is able to access the files during our grading. Otherwise, your script is likely to generate error and you may get as low as zero point for this part. Also make sure that the script inserts appropriate tuples to MaxPersonID and MaxMovieID tables.
- `queries.sql`: The MySQL script that contains **three** SELECT statements to issue queires to the dataset together with their English description as comments. Each SELECT statement should end with a semicolon(;).
- `query.php`: Your PHP code that allows users to issue a SELECT statement to the MySQL database. Please make sure the **all URLs in your file are relative**. Make sure that you connect to MySQL with the **username "cs143" and empty password and use the database `CS143`**. Make sure your PHP code does **not depend on any other files** (e.g., images, css file, other html files, etc). Otherwise, you may get as low as zero point due to errors during grading. Furthermore, the query.php is more like a question-answering interface, and therefore you should use **HTTP GET** protocol to process the user input as suggested by W3C.
- `violate.sql`: The list of SQL modification statements (INSERT/DELETE/UPDATE) that violate each constraint. For each statement, write a short explanation on which constraint it violates and why. Also, after each explanation of the 9 primary and foreign key constraints, "cut-and-paste" the output that you get from MySQL when the statement is executed.

Testing of Your Submission

Grading is a difficult and time-consuming process, and file naming and packaging convention is very important to test your submission without any error. In order to help you ensure the correct packaging of your submission, you can test your packaging by downloading this test script. In essence, this script unzips your submission to a temporary directory and tests whether or not you have included all your submission files. Once you download the test script, it can be executed like:

```
cs143@cs143:~$ ./p1b_test <Your UID>
```

(Put your P1B.zip file in the same directory with this test script, you may need to use "chmod +x p1b_test" if there is a permission error).

You **MUST** test your submission using the script before your final submission to minimize the chance of an unexpected error during grading. Significant points may be deducted if the grader encounters an error during grading. When evertything runs properly, you will see an output similar to the following from this script:

```
Check File Successfully. Please upload your P1B.zip file to CCLE.
```

Submitting Your Zip File

Visit the Project 1B submission page on CCLE to submit your zip file electronically by the deadline. **Submit only the "P1B.zip" file!** In order to accomodate the last minite snafu during submission, you will have 30-minute window after the deadline to finish your submission process. That is, as long as you start your submission before the deadline and complete within 30 minutes after the deadline, we won't deduct your grade period without any penalty.

Late Submission Policy

To accommodate the emergencies that students may encounter, each team has 4-day grace period for late submission. The grace period can be used for any part of the project in the unit of one day. For example, a student may use 1-day grace period for Project 1A and 2-day grace period for Project 2B. Any single project part may not be more than 2 days late. Note that even if a team submits a project 12 hours late, they would need to use a full day grace period to avoid late penalty. If your project is submitted late, we will automatically use the available days in your grace period unless you specifically mention otherwise in the README file.