EPBI 414 (Fall 2016) - Assignment 6

Basic SQL

Overview

In this homework, you will work with some basic SQL queries on the `hal` database server. In Part 1, you will work with MySQL on the command line, and will use the shell to do post-processing. In Part 2, you will work with more complex data, using the MySQL Workbench.

Logistics

Submit your assignment in a .zip file labeled in the following matter:

<Case ID>_EPBI414_Fall2016_A6.zip

So, if your Case ID is tar9, you would submit the following zip file:

tar9_EPBI414_Fall2016_A6.zip

Part 1

For this part, you should plan to use the command line interface with MySQL. We will use the data found in `epbi414_employee_data` on `hal`. You are looking at employment data, and your supervisor has asked you to pull some salary information to assess if there is a difference in salary between male and female employees.

You supervisor wants you to pull a set of employee salary records from the group of employees who were hired in 1985 and 1986. From that group, she wants you to retrieve the thirty highest-paid ***current*** employees of each gender in the database, male and female. She wants you to dump this output into a CSV file so she can easily load it into Excel.

To do this, you will write two SQL queries, and then execute them on the command line using a shell script. Your shell script will then catch the output from your SQL queries and clean it up into a nice CSV file.

You should submit two files: a file containing SQL statements, and a shell script which executes those SQL statements and cleans up the output. For the SQL queries, here are a few notes:

- The `employees` table does not contain information about whether someone is a current employee. However, you can assume anyone who has a record in `salaries` with a `to_date` of `9999-01-01` is a current employee. This means you will need to filter the records in `salaries` before you do your join, so you should plan to write a subquery and use an alias.

- It is fine to start your SQL file with a `USE` statement - you do not need to put the

fully-qualified database.table name for every query.

- You should plan on writing two SQL queries that are almost the same, except that one will retrieve the male records, and one will retrieve the female records. In future weeks, we'll discuss how you can use the `UNION` statement to combine those into one - but that's not required right now.

- Save your SQL queries into a file labeled `sql_e414_f2016_A6_P1.sql`.

For your shell script, here's a few notes:

- Your shell script should use...

```
mysql -u [YOURUSER] -p < sql_e414_f2016_A6_P1.sql
```

  ...to execute your SQL queries. You will need to pipe the results of that command into other commands. You should include your username in your file, but not your password.

- You will need to turn the standard `mysql` output, which is separated by tabs, into a comma-separated file. To do this, you will want to replace the tab characters (denoted by `\t`) with commas. Do this using `sed`.

- Also using `sed`, you should delete the headers that are output by `mysql` along with the data. However, your file should have one header at the top. You might think that you should solve this by somehow deleting the second appearance of the header, and keeping the first. It's actually easier to use `sed` to remove both headers, and then to pipe the output of that to `sed` again, where you insert a new header.

- All told, you should use a single line with three pipes, with your output redirected to a file `gender_salary_query.csv`. Save your command as a shell script, with the filename `sh_e414_f2016_A6_P1.sh`.

- For bonus points, make your script accept the name of the output file as a command-line argument.

You can compare your file to the correct version which is available with the homework, to make sure your program is producing the right output.

Grading

Comments and Formatting - SQL File	5 points
Comments and Formatting - Shell File	5 points
Includes Both Files	5 points
SQL Query File Correct	20 points
Shell File Correct	20 points

Bonus for Command Line Argument	5 points
TOTAL	**55 points**

Part 2

For this part of the assignment, you should use MySQL Workbench. We will be accessing the data stored in `epbi414_ehr_data`. This is demonstration data from OpenMRS, an open-source, enterprise-grade medical record system. You can find out more about OpenMRS at their website:

http://openmrs.org/

When you look into the OpenMRS data, you'll see that it can be pretty complicated. We won't be using all of the features of the OpenMRS data this week - and indeed, we probably won't touch all of it through the whole course. When you look at it, you can see why it is sometimes wise to take data from this type of "operational environment" and load it into a database that can be specifically used for research and reporting.

To start, we are going to pull some basic data about the patient encounters stored in this system. OpenMRS seems to store the test "type" as something it calls a ***concept***, which can be found in the table of the same name. Individual clinical measurements are stored in the `obs` table.

In general, your program should be properly commented and should include a preamble describing your name, the name of the program, and what the program is intended to do. It should be written in a manner that makes the code clear and easy to read.

You should write all three of your queries in a single file and submit that with your assignment. Name the file `sql_e414_f2016_A6_P2.sql`.

1. To start, write a query which returns the `person_id`, `encounter_id`, `short_name`, and `value_numeric` for all the observations with the following short_names:

SBP	systolic blood pressure
DBP	diastolic blood pressure
HR	heart rate
TEMP (C)	temperature in Celsius
WT	weight
HT	height

   To do this, you will need to join together the `obs` and `concept tables`, and you will need to filter `short_names`. You can do the filtering of `short_names` within the `ON` statement of your join in this case. We'll talk more about that in the next unit. Sort your output by person ID, then encounter ID, and then the name of the concept. [25 points]

2. Next, revise your previous query so that you return only the row containing the highest

temperature that was ever recorded. In addition to the variables from the previous question, also get the observation date. Don't change the query to be hard-coded - you still need to query the `concept` table to get the right concept ID. As before, go ahead and apply your filter in the `ON` statement. [10 points]

3. Let's imagine that we are interested in tracking locations where we have seen very high temperatures. Write a query which captures all the observations that had a temperature over 39.5 ℃, and reports the patient's ID, the encounter ID, the name of the concept, the actual temperature recorded, the date it was recorded, and the name of the location where it was recorded. Sort your output with the highest temperature last. Information about the locations can be found in the `location` table.
   For this query, you will need to join together three tables: `obs`, `location`, and `concept`. As with the previous version, don't change the query of concept ID to be hard-coded.

   You will need to apply a few different logical filters on this query. You should plan to use a `WHERE` statement on any filters which apply to `obs`, and should do the other filtering using the `ON` part of your join. [30 points]