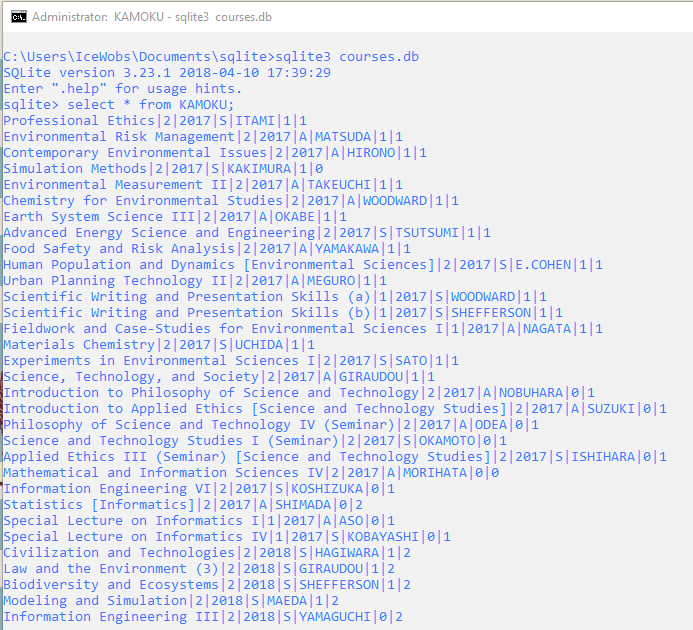
Alexander Taniguchi(Wiegman)  
08-174510

Homework Week 4: More Database SQL with My Courses

Logic

The assignment is to modify a database of my courses and show a few more operations on such a database. We will base this assignment off the same ‘courses.db’ that was created in the previous week. While I see that there is a worry for redundancy, this is altogether too unlikely (at least from a personal standpoint) and was not something I was giving much consideration towards. As a reminder, this is the database table as I currently have it.



In this table, I have created the columns for the ‘course TITLE’ as the ‘name’, ‘CREDITS obtained’ as an integer, ‘YEAR course was taken’ as a date, ‘SEMESTER course was taken (summer or autumn)’ as a one-letter ‘s’ or ‘a’ textual representation, ‘PROFESSOR teaching the course’ using their ‘LAST\_NAME’, ‘REQUIRED course or elective’ as ‘BOOLEAN’ value (0 is elective, 1 is required for major), and ‘PASS’ as an integer representation where 0 is not pass (fail), 1 is pass, and 2 is “didn’t sit exam / drop / other”.

With this simple table, we will now move out of logic and into usage.

Usage

*Mathematical Operations*

I think it could be possible to count my total credits. This can be done with relative simplicity using the *sum* command.

 But quickly, I realize this is wrong. Why?

Sometimes, the course is not one we are able to count. It is one that I did not pass for a various reason, or one that I am currently taking. Hence, we must employ the *where* command to further constrict our criteria.

 This looks much better now.

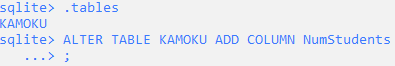
[I am still behind and need to take many more classes T\_T!]

A more interesting calculation would involve taking an average of any single course’s credit hours from the courses that I have passed. Still, like all mathematical commands in programming languages, a relatively simple thing to have output.

 ~ 1.8 credits per course. Considering my faculties are made of 1 or 2 credit courses, I am taking an efficient load of mostly 2 credit courses, and not very many 1 credit courses.

*Adding a Column in SQL for adding some extra information*

A question I had from a previous week wondered how I could easily edit the table in the database I have created to make an additional column – for example, I liked how one student noted the number of students in a course, or perhaps to give the course a rating from 1-5.



After confirming the name of the table, we use the ‘*ALTER TABLE*’ command and the ‘*ADD COLUMN*’ command to select, mark for change, and add the column ‘*NumStudents*’ which will list the number of students present in a course.



Success! Now, we see an entry for 26 students (and indeed in the following entries, there is a column-space of blank entries). The | shows the existence of the column.



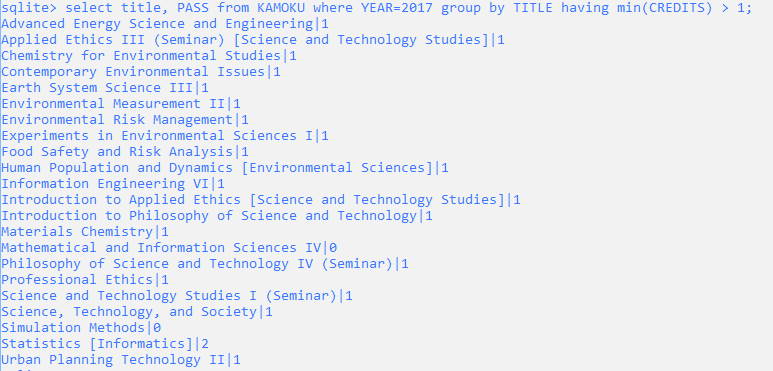
However, this is where I noticed I also messed up in last week’s assignment big-time. It would have been well to have included an ID for each course (row) so that I would be able to uniquely identify a course by ID = 1… n (or even using the ‘08xxx’ designation that Komaba Campus uses), instead of TITLE = ”(name)” in a manual-labor intensive row recursion.

Update from a few days later: A re-read of the course material shows me that SQLITE uses the command ‘*rowid*’ to individually number each row.

Indeed, this is much better.

*Querying Data from the Table*

* Show me all courses and if I passed them that I have taken in 2017, where results are listed in alphabetical order and only list a 2-credit course.

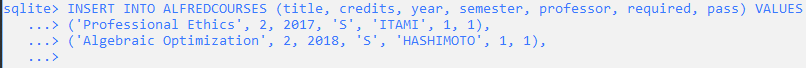


This happens to be a very good use of the ‘*having*’ qualifier-type command.

* While I don’t happen to have a second table, if you had two people compare the courses they took in 2017 and output all courses in common. Output the common courses in any order.



First, I initialize a table for my friend’s courses.



Next, I insert some values into the table.



The ‘*distinct*’ command will help avoid redundancy and the intersect command finds the common courses of the two sets. However, this will output the whole line. Hence, we change the wildcard to a specific entry for course TITLE(s).



* Non-redundantly display a list of professors from courses I took. Show results alphabetically. Further, query the total count of professors I have met through taking their course.

The key here is a command called ‘*distinct*’, helping us avoid 2+ *Woodward* entries, for example.



That looks like a small library of professors. Now, let’s envelope the whole command in a ‘*count*’ statement and see if I get the correct number of professors.

 This is incorrect.

 This is correct.

The *order of keywords*, as we can see, is equally as important as the use of an *accurate* term!

*(End of Week 4 Assignment)*