COMP3311 19T3 Database Systems

Assignment 3
SQL and Python and Psycopg2

Last updated: **Saturday 9th November 6:34pm**
Most recent changes are shown in red ... older changes are shown in brown.
[Specification] ... [Database] ... [Expected Results]

Aims

The aims of this assignment are to:

- use Python/psycopg2 and SQL/PLpgSQL to access and process data from a database

Admin

Submission: Use the command `give cs3311 ass3 ass3.sql q?.py` or use Webcms3

Deadline: Monday 18th November 12:00

Late Penalty: Late submissions will have marks deducted from the maximum achievable mark at the rate of 0.7% of the total mark *per hour* that they are late (i.e., around 17% per day).

Database

In this assignment, you will work with a database of UNSW timetable data. More details on the structure of this database are available in a separate page.

Getting Started

Make sure you read this entire specification thoroughly, then create and load the database using the commands:

```
grieg% dropdb a3
grieg% createdb a3
grieg% psql a3 -f /home/cs3311/web/19T3/assignments/ass3/files/timetable.sql
```

If you're working on your own machine with PostgreSQL installed, you'll need to download a copy of the dump file from:

```
/home/cs3311/web/19T3/assignments/ass3/files/timetable.sql
```

Note: use right mouse click and "Save Link As" for the above, or you might get ~11MB of text dumped into your browser.

Once you've loaded the data, start a `psql` session explore the database, using the `psql` meta-commands (e.g. `\d`) and running some SQL queries.

It is useful to create a separate directory just for this assignment. Once you have created that directory, you can simply run the command

```
$ 3311 ass3setup
```

and it will produce appropriate stub files for you in that directory. **Note: If you run this command again, it will over-write your files. Be careful.**

Note that when solving the problems above, you can define as many auxiliary views and functions as you want. These must also be included in the `ass3.sql` file you submit, and must be ordered so that the file can load into a fresh database in one pass (i.e. no forward references).

A readable copy of the schema is available in the file:

```
/home/cs3311/web/19T3/assignments/ass3/files/schema.sql
```

You do *not* need to load this schema into your database. It is already a part of `timetable.sql`.

Exercises

Queries should not take more than 0.5 seconds to run (unless otherwise specified); queries that take longer to run will be penalised.

Provide Python solutions for each of the following:

1. **For each course offered in 19T3, produce a list of all courses with a quota more than 50 and that are over-enrolled. I.e. when the number of enrollments exceeds the quota for the course. Results should be ordered by course name (ascending)**
   - To clarify: offered(C,19T3) & quota(C) > 50 & nEnrolledStudents(C) > quota(C)
   - Your solution should run with `python3 q1.py`
   - When automarked, your script for this question will have a time limit of 0.8 seconds.

   Format of output: 0 or more rows, where each row is

```
[course code] [percentage]%
```

[Expected result] (1 mark)

2. **Someone has asked you "I wonder how many cases there are were 5 UNSW courses share the same 4 numbers in their course code?". Write a solution that produces a list of all different cases where there are X UNSW courses that share the same course code numbers, where X is passed in as the command line argument.**

   - Your solution should run with `python3 q2.py [incommon]` where `incommon` is an integer between 2 and 10. If no argument is provided, the default `incommon` should be 2.
   - The numeric codes should be in ascending order. The 4-char codes should be sorted alphabetically in each row.
   - When automarked, this question will have a time limit imposed. Because the time limit depends on the input, we can't give you the number we'll end up using. However, if running `python3 q2.py 2` takes less than 0.3 seconds, you should be fine. If it takes longer than that, make your code faster.

   Format of output: 0 or more rows, where each row is

   ```
   [course code 4 digit component]: [list of 4 characters preceding course codes separated by a space]
   ```

   The course codes should be listed in ascending numerical order. For each course code, the course characters should be listed in alphabetical order.

   [Expected result] (3 marks)

3. **Given a 4 letter course code prefix (e.g. COMP) as a command line argument, produce a list of buildings, where for each building a sub-list is produced showing which courses have classes in them during 19T2.**

   Buildings with no applicable courses done in it are not listed

   - Your solution should run with `python3 q3.py [code]` where code is a valid 4 digit prefix to course codes. If no argument is provided, the default code should be `ENGG`. You can assume all codes passed in will be valid.
   - When automarked, this question will have a time limit imposed. Because the time limit depends on the input, we can't give you the number we'll end up using. However, if running `python3 q3.py ENGG` takes less than 0.3 seconds, you should be fine. If it takes longer than that, make your code faster.

   Format of output: 0 sets of responses, where each response has 1 or more rows

   ```
   [Building 1]
   [1 space][course code 1]
   [1 space][course code 2]
   [1 space][...]
   [Building 2]
   [1 space][course code 3]
   [1 space][course code 4]
   [1 space][...]
    [...]
   [...]
   ```

   The buildings should be listed in lexicographical order. Within each building section, the course codes are listed in lexicographical order

   [Expected result] (3 marks)

4. **Given a 4 letter course code prefix (e.g. COMP) as a command line argument, produce a list of terms (ordered ascending), where for each term a sub-list is produced showing which courses run in that term and how many students are currently enrolled in those courses.**

   Terms with no courses having students enrolled in them are not listed

   - Your solution should run with `python3 q4.py [code]` where code is a valid 4 digit prefix to course codes. If no argument is provided, the default code should be `ENGG`. You can assume all codes passed in will be valid.
   - When automarked, this question will have a time limit imposed. Because the time limit depends on the input, we can't give you the number we'll end up using. However, if running `python3 q4.py ENGG` takes less than 0.5 seconds, you should be fine. If it takes longer than that, make your code faster.

   Format of output: 0 sets of responses, where each response has 1 or more rows

   ```
   [Term item 1]
   [1 space][course code 1]([number of current enrolled students])
   [1 space][course code 2]([number of current enrolled students])
   [1 space][...]
   [Term item 2]
   [1 space][course code 3]([number of current enrolled students])
   [1 space][course code 4]([number of current enrolled students])
   [1 space][...]
   [...]
   ```

   The terms should be listed in lexicographical order. Within each term section, the course codes are listed in lexicographical order

   [Expected result] (3 marks)

5. **For a given course (e.g. COMP1511), find all classes (lectures, tutes, etc) that have less than 50% enrolment in 19T3**

- Your solution should run with `python3 q5.py [course]` where course is a valid 8-character course code. If no argument is provided, the default code should be `COMP1521`. You can assume all course codes passed in will be valid.
- When automarked, this question will have a time limit imposed. Because the time limit depends on the input, we can't give you the number we'll end up using. However, if running `python3 q5.py COMP1521` takes less than 0.5 seconds, you should be fine. If it takes longer than that, make your code faster.

Format of output: 0 or more rows, where each row is

```
[Class type] [Class tag] is [full]% full
```

Each line of output should be ordered by class type name, then by class tag, then by the final percentage.

[Expected result] (2 marks)

6. In the `Meetings` table, the `weeks` column currently stores information about the weeks that a meeting runs in the format "1-5,7-10" or "2,4,6,8,10". While this makes sense to a human, it's not a friendly format to use when processing large datasets.

   You must write an `UPDATE` query that takes the text format of the `weeks` column and outputs a binary format in the column `weeks_binary`.

   For example, "1-5,7-10" would convert to "11111011110", and "2,4,6,8,11" would convert to "01010101001"

   If a `weeks` column contains an 'N' character or a '<' character, this is a strange edge case, and you can convert this to just "00000000000" (all zeroes)

   The `weeks_binary` column is a text string consisting of 11 chars, each of which must be '0' or '1'.

   - Your solution should run with `python3 q6.py`
   - When automarked, this question will have a time limit imposed of 4s seconds to run your script.

   [Expected result] (5 marks)

   ## Help with this question

   If you are struggling to implement this solution:

   - Start by trying to output all of the UPDATE statements to stdout (or in a file)
   - Compare those update statements to our produced UPDATE statements in q6helper.sql
   - Once output matches, change the **q6.py** script so that it now executes the UPDATEs directly, instead of writing them to stdout

7. **What percentage of rooms at UNSW are *underused* during a term at UNSW? Underused is defined as when across weeks 1-10 inclusive, the room is used on average for less than 20 hours a week**

   - Your solution should run with `python3 q7.py [term]` where term is one of 19T1, 19T2, or 19T3. If no argument is provided, the default term should be 19T1. You can assume all terms passed in will be valid.
   - The definition of "rooms at UNSW" is any room whose code looks like `'K-xxxxx'` (i.e. Kensington campus)
   - When automarked, this question will have a time limit imposed. Because the time limit depends on the input, we can't give you the number we'll end up using. However, if running `python3 q7.py 19T1` takes less than 0.8 seconds, you should be fine. If it takes longer than that, make your code faster.
   - You will want to utilise the `weeks_binary` column you produced in question 6 to make this easier. If you are unable to complete question 6, you can run this query on your database to add the correct values to the `weeks_binary` column.
   - Your solution is allowed to ignore double-booking and overlapping class issues, but such a solution is only worth 4/5, since it slightly underestimates the number of under-utilised rooms. If you take account of all possible cases of overlapping classes, you can score 5/5. Outputs for the two different cases are shown in the Expected Results page.

   Format of output:

   ```
   [percentage of underused rooms to 1 decimal place]%
   ```

   [Expected result] (5 marks)

8. **Produce a timetable for 19T3 for 1, 2 or 3 courses that aims to minimise the amount of hours being spent in a week on campus or commuting.**

   In cases where two timetables generate the same number of hours spent on campus or commuting, choose the timetable which has the fewer days on campus

   In cases where two timetables have the same number of days on campus, choose the timetable that has all of the classes done as early as possible in the week, i.e. Classes should occur as early in the week as possible.

   For a given course, you need to take exactly one of each of the class types for that course. For each class of a given type, you need to attend all of the meetings for that class (e.g. CSE tute/labs have two meetings in consecutive timeslots; some ARTS course require students to attend two tutorials on different days).

   To simplify things, you can assume that every class runs in all of weeks 1-10 (i.e. you can ignore the `weeks` and `weeks_binary` fields)

   - Your solution should run with `python3 q8.py` *`Courses...`*. If no courses are supplied, the default courses should be `COMP1511` and `MATH1131`.
   - Assume that you can only travel to and from campus once in a day. Assume that commuting is always 1 hour in each direction, regardless of the time of day.

- Assume you will not be given input that produces unresolvable lecture clashes
- Assume that all supplied course codes are valid and distinct (i.e. no repetitions).
- Assume that all courses we test with will only have classes on the Kensington campus.
- When automarked, this question will have a time limit imposed. Because the time limit depends on the input, we can't give you the number we'll end up using. However, if running `python3 q8.py COMP1511 MATH1141 MATH1081` takes less than 10 seconds, you should be fine. If it takes longer than that, make your code faster.

Format of output:

```
Total hours: [hours to 1 decimal place]
[2 spaces][Day]
[2 spaces][2 spaces][Course code] [Class type name]: [start time]-[end time]
[2 spaces][2 spaces][Course code] [Class type name]: [start time]-[end time]
[2 spaces][2 spaces][Course code] [Class type name]: [start time]-[end time]
[2 spaces][Day]
[2 spaces][2 spaces][Course code] [Class type name]: [start time]-[end time]
[2 spaces][2 spaces][Course code] [Class type name]: [start time]-[end time]
[2 spaces][Day]
[2 spaces][2 spaces][Course code] [Class type name]: [start time]-[end time]
[2 spaces][Day]
[2 spaces][2 spaces][Course code] [Class type name]: [start time]-[end time]
[2 spaces][2 spaces][Course code] [Class type name]: [start time]-[end time]
[2 spaces][2 spaces][Course code] [Class type name]: [start time]-[end time]
[2 spaces][2 spaces][Course code] [Class type name]: [start time]-[end time]
[2 spaces][2 spaces][Course code] [Class type name]: [start time]-[end time]
```

Outputs are printed in chunks per day. Days should be ordered in ascending order (Mon < Tue < Wed < Thu < Fri). Within each day, the order in which class times are printed is based on increasing order of start_time.

[Expected result] (7 marks)

Submission

Your submission should include the file `ass3.sql`, whether or not you put any SQL views or PLpgSQL functions in it. It should also include all of the Python scripts (`q1.py`, `q2.py`, ..., `q8.py`). Submit all Python scripts even if you didn't attempt some questions.

If you have all of your scripts in a single assignment directory, the following command would do the submission:

```
$ give cs3311 ass3 ass3.sql q?.py
```

Note that you do not submit the `cs3311.py` script; we'll put a copy of that in the directory where we test your code.

Make sure that all of your scripts that call `psycopg2.connect()` use the database name `a3`. If you use the `cs3311.py` database connector, it uses the correct database name.

Assessment

This assignment is worth a total of **29 marks**. It will later be scaled to 13 percent for the course as described in the course outline.

Note that questions are not worth equal marks; the mark for each question is intended to be indicative of how difficult the question is.

If we have to fix errors in your solution before it will load and run, you will incur a 6 (out of total 29) mark penalty. Python scripts that take too long to run will score only half the marks for that question, if they produce the correct answer.

Testing

To test your code, you can simply run the command

```
$ 3311 ass3check
```

on your CSE machine. Ensure that this is run while a psql server is currently running on `grieg`.

We will carry out the testing for marks as follows:

```
$ dropdb a3
$ createdb a3
$ psql a3 -f /home/cs3311/web/19T3/assignments/ass3/files/timetable.sql
... make a new copy of the timetable database ...
$ psql a3 -f ass3.sql
... load all of your views and functions (if any) ...
$ sh ass3check
... run the standard test set ...
$ sh ass3check2
... run additional (unseen) tests ...
```

Have fun, *jas* and Hayden