COMP3311 23T3 Database Systems

Assignment 2
Python, PostgreSQL, MyMyUNSW

Last updated: **Thursday 9th November 9:27pm**

Most recent changes are shown in red ... older changes are shown in brown.

[Specification] [Database] [SQL Schema] [Grades+Rules] [Examples] [Testing] [Submitting] [Fixes+Updates]

Aims

This assignment aims to give you practice in

- reading, understanding and querying a moderately large relational schema (MyMyUNSW)
- implementing Python scripts to extract and display data from a database based on this schema
- implementing SQL views and PLpgSQL functions to aid in satisfying requests for information

The goal is to build some useful data access operations on the MyMyUNSW database.

A minor theme of this assignment is "dirty data". We built the database, using a collection of reports from UNSW's information systems. There may be a very occasional glitch in the data, but the data is not generally dirty. Just work with what's in the database; If it produces strange results, then write some queries to check whether those results fit with what's actually in the database, which is something you should be doing anyway.

You may bump into some residual data issues as you try to solve the problems below. Let us know via the Forum if you think you have found such data anomalies.

Summary

Marks:	This assignment contributes **16 marks** toward your total mark for this course.
Submission:	Login to Course Web Site > Assignments > Assignment 1 > [Submit] > upload `q1.py`, `q2.py`, `q3.py`, `q4.py`, `q5.py`, `helpers.py`, `helpers.sql` or, on a CSE server, give cs3311 ass2 `q1.py q2.py q3.py q4.py q5.py helpers.py helpers.sql`
Deadline:	Wednesday 15 November @ 23:59

Late Penalty:	Standard UNSW late penalty: 5% off the maximum achievable mark for each day late; marks deducted by the hour; after 5days late, mark is zero

How to do this assignment:

- read this specification carefully and completely
- familiarise yourself with the database schema
- create a database `ass2` on the host `vxdb2`
- explore the database to see how rules, etc. are represented
- make a private directory for this assignment
- put a copy of the template files there
- edit the files in this directory on a host other than `vxdb2`
- on `vxdb2`, test that your Python scripts produce the expected output
- submit the assignment via WebCMS3 or `give` (as described on the What to Submit page)

And, of course, if you have PostgreSQL installed on your home machine, you can do all of the `vxdb2` work there. BUT don't forget to test it on `vxdb2` before submitting.

The "template files" aim to save you some time in writing Python code. E.g. they handle the command-line arguments and let you focus on the database interaction. They are available in a file ass2.zip, which contains the following

- `helpers.sql` ... any views or PLpgSQL functions to assist your Python
- `helpers.py` ... any Python function to share between scripts
- `q1.py` ... Python script to track international student numbers over time
- `q2.py` ... Python script to track course satisfaction over time
- `q3.py` ... Python script to display program/stream rules
- `q4.py` ... Python script to produce a transcript
- `q5.py` ... Python script to do progression check

There are even a couple of functions in `helpers.py`. Freebies!

Introduction

All Universities require a significant information infrastructure in order to manage their affairs. This typically involves a large commercial DBMS installation. UNSW's student information system sits behind the MyUNSW web site. MyUNSW provides an interface to a PeopleSoft enterprise management system with an underlying Oracle database. This back-end system (Peoplesoft/Oracle) is sometimes called NSS. The specific version of PeopleSoft that we use is called Campus Solutions. There is also a system called SiMS, which can be used to access the data.

UNSW has spent a considerable amount of money ($100M+) on the MyUNSW/NSS system, and it handles much of the educational administration plausibly well. Most people gripe about the quality of the MyUNSW interface, but the system does allow you to carry out most basic enrolment tasks online.

Despite its successes, however, MyUNSW/NSS still has a number of deficiencies, including:

- no usable representation for degree program structures
- minimal integration with the UNSW Online Handbook

The first point prevents MyUNSW/NSS from being used for three important operations that would be extremely helpful to students in managing their enrolment:

- finding out how far they have progressed through their degree program, and what remains to be completed
- checking what are their enrolment options for next semester (e.g., get a list of “suggested” courses)
- determining when they have completed all of the requirements of their degree program and are eligible to graduate

The second point allows for inconsistencies between the Handbook and the system that manages enrolment.

NSS contains data about students, courses, classes, pre-requisites, quotas, etc. but does not contain any representation of UNSW's degree program structures. Without such information in the NSS database, it is not possible to do any of the above three. So, in 2007 the COMP3311 class devised a data model that could represent program requirements and rules for UNSW degrees. This was built on top of an existing schema that represented all of the core NSS data (students, staff, courses, classes, etc.). The enhanced data model was named the “MyMyUNSW” schema.

The MyMyUNSW database includes information that encompasses the functionality of NSS, the UNSW Online Handbook, and the CATS (room allocation) database. In 2023, we trimmed the schema, removing all tables related to classes, rooms, textbooks, staff affiliations, etc. and revised how program requirements where represented. Note that there is also no information on subject pre-requisites in the new schema, so proper forward degree planning is not possible. The remaining data *does* allow us to check progression and graduation status.

The new MyMyUNSW data model, schema and database are described in a separate document. The data in the schema is drawn from 2019-2023 enrolment data, and the data has been anonymised; we are using only a tiny fraction of the available data. This results in courses having tiny (or no) enrolments, but still has enrolment data for 8000 students.

Setting Up

To install the MyMyUNSW database under your PostgreSQL server on `vxdb2`, simply run the following two commands (after ensuring that your server is running):

```
$ createdb ass2
$ psql ass2 -f /home/cs3311/web/23T3/assignments/ass2/files/a
```

If everything proceeds correctly, the load output should look something like:

```
SET
SET
...
SET
CREATE TYPE
... a few of these
CREATE DOMAIN
... a few of these
CREATE TABLE
... a whole bunch of these
COPY n
... a whole bunch of these
ALTER TABLE
... a whole bunch of these
ALTER TABLE
```

You should get no `ERROR` messages. The database loading should take less than 5 seconds on `vxdb2`, assuming that `vxdb2` is not under heavy load. (If you leave your assignment until the last minute, loading the database on `vxdb2` may be considerably slower, thus delaying your work even more. The solution: at least load the database *Right Now*, even if you don't start using it for a while.) (Note that the `ass2.dump` file is 3MB in size; copying it under your home directory on the CSE machines is not a good idea.)

Note that the database is called `ass2` (when you want to access it via `psql` or Python script). Throughout this document we refer to the database as "MyMyUNSW". Note also that this is **not** real data, although it was generated with much swizzling, from real data.

A useful thing to do initially is to get a feeling for what data is actually there. This may help you understand the schema better, and will make the descriptions of the exercises easier to understand. Look at the schema. Ask some queries. Do it now.

Examples ...

```
$ psql ass2
... PostgreSQL welcome stuff ...
ass2=# \d
... look at the schema ...
```

```
ass2=# select * from Students;
... look at the Students table; a list of zid's ...
ass2=# select p.id,p.fullname from People p join Students s c
... look at the names and UNSW ids of all students ...
ass2=# select p.id,p.fullname,s.phone from People p join Staf
... only one result because there's only one staff ...
ass2=# select count(*) from Course_enrolments;
... how many course enrolment records ...
ass2=# select * from dbpop();
... how many records in all tables ...
ass2=# ... etc. etc. etc.
ass2=# \q
```

You will find that many tables (e.g. `Books`, `Buildings`, etc.) are currently unpopulated; their contents are not needed for this assignment.

Summary on Getting Started

To set up your database for this assignment, run the following commands:

```
$ ssh vxdb2
... and then on vxdb2 ...
$ source /localhost/$USER/env
$ p1
... you shut down the server after your last session, didn't you?
$ createdb  ass2
$ psql  ass2  -f  /home/cs3311/web/23T3/assignments/ass2/file
$ psql  ass2
... run some checks to make sure the database is ok
$ mkdir  Assignment2Directory
... make a working directory for Assignment 2
$ cd Assignment2Directory
$ unzip /home/cs3311/web/23T3/assignments/ass2/files/ass2.zip
... puts the template files in your working directory
```

The only messages produced by these commands should be those noted above. If you omit any of the steps, then things will not work as planned. If you subsequently ask questions on the Forums, where it's clear that you have *not* done and checked the above steps, the questions will not be answered.

Exercises

Q0 (2 marks)

Style mark.

Ugly, inconsistent layout of SQL queries and PLpgSQL functions will be penalised. It's hard to layout Python3 code wrong, given that indentation replaces brackets, but if you manage to make your Python code ugly, that will also be penalised. You should ensure that your Python variable names are understandable and consistent.

Q1 (3 marks)

Write a Python script `q1.py` to track the proportion of international students over terms from 19T1 to 23T3.

Use the `Students` table to determine who's local/international. Treat any student whose status is `INTL` as international; treat everyone else as local (yes, even Kiwis). Use the `Program_enrolments` table to determine which students are enrolled in a given term.

Display each term code on a separate line, using the following formatting:

```
f"{TermCode} {#Locals:6d} {#Internationals:6d} {Proportion:6.
```

The `Proportion` is simply `#Locals/#Internationals`.

Add a heading at the start of the output in the same format as on the Examples page).

Warning: some students appear to be enrolled in two programs in one term.

Q2 (3 marks)

Write a Python script `q2.py` to track the satisfaction in a specified subject over terms from 19T1 to 23T3.

The script takes one command line parameter (a subject code)

```
python3  q2.py  SubjectCode
```

Display results for each term on a separate line, using the following formatting:

```
f"{TermCode} {Satifaction:6d} {#Responses:6d} {#Students:6d}
```

If any of the attributes have a NULL value, use a `"?"` instead of the number or convenor name (and in this case you will need to use a different format string than the above). If a subject wasn't offered in a particular term, don't print any line for that term.

Add a heading at the start of the output in the same format as on the Examples page).

Q3 (5 marks)

Write a Python script `q3.py` that takes a program code or a stream code and produces a readable list of rules for that program or stream.

The script takes on command-line parameter:

```
python3  q3.py  (Stream code or Program code)
```

A number of different requirement types are given in the Grades and Rules page. More details on the precise output format for rules will be available in the Examples page. All of the rules stored in the database are given in the Grades

and Rules page.

The requirements should be displayed in a specific order:

- display any `uoc` requirements first
- then display any `stream` requirements
- then display any `core` requirements
- then display any `elective` requirements
- then display any `gened` requirements
- then display any `free` elective requirements

If there are multiple requirements of the same type, display them in order of `Requirements.id`.

Display the name of the requirement after the min/max output described below e.g.

```
all courses from Level 1 Core
at least 6 UOC courses from Database Electives
between 12 and 18 UOC courses from Comp Sci electives
1 stream from Engineering majors
```

The brown text is the requirement name.

Total UOC rules are an exception to this. The "Total UOC" is displyed *before* the min/max output.

When displaying rules, use the following for min and max

- min and max are null ... nothing to be displayed
- min is not null, max is null ... "at least *min*"
- min is null, max is not null ... "up to *max*"
- both are not null and min < max ... "between *min* and *max*"
- both are not null and min = max ... display value of "*min*"

If the requirement is based on courses, add "UOC". If the requirement is based on streams, add the word "stream".

In the database, all of the rules are stored in the `Requirements` table. More details on the fields in this table are given in the Database page.

Q4 (5 marks)

Write a Python script `q4.py` that takes a command-line argument giving a student ID, and prints a transcript for that student. The template script already checks the command line argument.

The transcript should start with a two-line heading.

```
zID FamilyName, GivenNames
```

```
ProgramCode StreamCode ProgramName
```

The program and stream should be the ones most recently enrolled by the student.

Each line of the transcript should contain

```
CourseCode  Term  SubjectTitle  Mark  Grade  UOC
```

Use the following f-string to get the formatting right

```
f"{CourseCode} {Term} {SubjectTitle:<32s}{Mark:>3} {Grade:>2s
```

Entries should be ordered by term, and within the same term, by course code. You should also calculate a WAM and display this at the end of the course lines. How to use the grades and marks to determine the WAM is given in the Grades + Rules page. The precise format of the output will be available in the Examples page.

Some subject names are longer than 32 characters. To keep the table neat simply truncate any longer names to exactly 32 characters.

Note that there are two UOC totals in this question:

- total_achieved_uoc = sum (uoc_i) where course_i is a "pass"
  this includes obvious ones like HD,DN,... but also ones like XE
  basically any couse with "yes" in the UOC column in the Grades table
- total_attempted_uoc = sum(uoc_i) for any course_i attempted
  even if the course was failed, we count its UOC; it was attempted
  any course which has "yes" in the WAM column in the Grades table
- weighted_mark_sum = sum(uoc_i * mark_i) for any course_i attempted
  any course which has "yes" in the WAM column in the Grades table
  if the course has no mark, treat the mark as zero

WAM = weighted_mark_sum / total_attempted_uoc

If either of the mark or grade is null, print a "-", right-aligned, where grade or mark would normally go.

To simplify this task, it would be useful to write a `transcript(integer)` function to extract transcript data in the correct order as a sequence of tuples containing the above fields. The argument to `transcript()` is the student's zID (i.e. the command-line argument).

Q5 (6 marks)

Write a Python script to show a student's progression through their program/stream, and what they still need to do to complete their degree. The script takes three command line parameters:

```
python3  q5.py  StudentID  [ ProgramCode  StreamCode ]
```

If no program/stream is given, use the program/stream for the student's most recent enrolment term. The script already checks the validity of the command-line arguments.

The output should look like a transcript, but with additional information to indicate which rule each course satisfies

```
CourseCode  Term  CourseTitle  Mark  Grade  UOC  NameOfRequir
```

The order should be the same as for the transcript script (i.e. order by term, then by course code within the term).

You should keep track of which courses and how many UOC in which requirements have been completed. After the line for each of the courses taken, you should display a sequence of lines indicating which core courses have not been completed, and how many UOC from each group of electives remains to be done.

If you consider each requirement as a bucket, then the process of determining which requirement a course satisfies, is a process of determining which bucket a particular course belongs in. If the bucket for the most appropriate requirement is full, the course cannot be allocated to that requirement, and a new requirement must be sought. In the "worst" case, the course will end up in the free electives bucket. If the free electives bucket is full, and if all of the other buckets that the course potentially be allocated to are also full, then the course cannot be allocated to any requirement and does not count toward the degree. Such courses should have 0UC against them and have a note "Cannot be allocated".

The strategy for ordering the "to be completed" info

- do all Core requirements first, stream Core's before program Core's
- then do all Elective requirements, stream Elective before program Electives
- then do GenEd requirements, then Free (elective) requirements

In other words, most specific to least specific.

Within groups (e.g stream Core's) order by `Requirements.id`. For Core requirements, print remaining UOC and the course codes of any not yet completed courses, in the order they appear in the group definition. For all other rule types, print remaining UOC and the name of the group. If a student has completed all UOCs for a rule, then no information on this rule needs to be printed.

If a student has satisfied all rules and enough UOC for the program, you should print

```
Eligible to graduate
```

instead of the "to be completed" text.

More details on the precise output format for rules will be available in the Examples page.

Submission

Submit this assignment by doing the following:

Login to Course Web Site > Assignments > Assignment 2 > Submit Your Work > Make Submission >
upload `helpers.sql`, `helpers.py`, `trans`, `rules`, `prog` > [Submit]

The `helpers.sql` file should contain all the views and functions that you've written to make your Python code simpler. It should be completely self-contained and able to load in a single pass, so that it can be auto-tested as follows:

- a fresh copy of the MyMyUNSW database will be created (using the schema from `ass2.dump`)
- the data in this database may be **different** to the database that you're using for testing
- a new `check.sql` file will be loaded (with expected results appropriate for the database)
- the contents of your `helpers.sql` file will be loaded
- each checking function will be executed and the results recorded

Before you submit your solution, you should check that it will load correctly for testing by using something like the following operations:

```
$ dropdb ass2                    ... remove any existing DB
$ createdb ass2                  ... create an empty database
$ psql ass2 -f ..../ass2.dump     ... load the MyMyUNSW schema ar
$ psql ass2 -f helpers.sql         ... load your SQL code
```

Note: if your database contains any views or functions that are not available in the `helpers.sql` file, you should add them to that file before you drop the database.

If your code does not load without errors, fix it and repeat the above until it does.

You must ensure that your `helpers.sql` file will load correctly (i.e., it has no syntax errors and it contains all of your view definitions in the correct order). If we need to manually fix problems with your `helpers.sql` file in order to test it (e.g., change the order of some definitions), **you will be "fined" via a 1 mark penalty on your ceiling mark** (i.e., the maximum you can score is 19 out of 20 marks).