Assignment 2
Python, PostgreSQL, MyMyUNSW

Last updated: **Saturday 13th November 8:45am**

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 and understanding a moderately large and complex 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 what's given, and 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 have found such data anomalies.

Summary

Marks:	This assignment contributes **20 marks** toward your total mark for this course.
Submission:	via Webcms3 or `give`, submit the files `rules`, `trans`, `prog`, `helpers.py`, `helpers.sql`
Deadline:	~~Friday 12~~ Monday 15 November 2021 @ 21:00
Late Penalty:	0.1 *marks* off the maximum achievable mark for each hour late (i.e., 2.4 marks per day).

How to do this assignment:

- read this specification carefully and completely
- familiarise yourself with the database schema
- create a database `mymyunsw` on the host `d.cse`
- 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 `d.cse`
- on `d.cse`, 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 `d.cse` work there. BUT don't forget to test it on `d.cse` 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
- `trans` ... Python script to produce a transcript
- `rules` ... Python script to display program/stream rules
- `prog` ... 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. The MyMyUNSW data model, schema and database are described in a separate document. Note that, while the *schema* is the complete one we developed in 2007, the *data* for this database has been trimmed to the absolute minimum to do this assignment; many tables and columns are empty.

Setting Up

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

```
$ createdb mymyunsw
$ psql mymyunsw -f /home/cs3311/web/21T3/assignments/ass2/mymyunsw.dump
```

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

```
SET
SET
...
SET
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 10 seconds on `d.cse`, assuming that `d.cse` is not under heavy load. (If you leave your assignment until the last minute, loading the database on `d.cse` 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 `mymyunsw.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 `mymyunsw` (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 mymyunsw
... PostgreSQL welcome stuff ...
mymyunsw=# \d
... look at the schema ...
mymyunsw=# select * from Students;
... look at the Students table; a list of zid's ...
mymyunsw=# select p.id,p.fullname from People p join Students s on p.id=s.id;
... look at the names and UNSW ids of all students ...
mymyunsw=# select p.id,p.fullname,s.phone from People p join Staff s on p.id=s.
... only one result because there's only one staff ...
mymyunsw=# select count(*) from Course_enrolments;
... how many course enrolment records ...
mymyunsw=# select * from dbpop();
... how many records in all tables ...
mymyunsw=# ... etc. etc. etc.
mymyunsw=# \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 d.cse
... and then on d.cse ...
$ source /localhost/$USER/env
$ p1
... you shut down the server after your last session, didn't you?
$ createdb  mymyunsw
$ psql  mymyunsw  -f  /home/cs3311/web/21T3/assignments/ass2/mymyunsw.dump
$ psql  mymyunsw
... 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/21T3/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

Q1 (6 marks)

Write a Python script called `trans`, 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.

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}  {UOC:2d}uoc"
```

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. The WAM is given by *WAM = Sum(Mark*UOC) / TotalAttemptedUOC*. 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.

Ii either of the mark or grad 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 of type `TranscriptRecord`. This type is already defined in the database and you can use it or not as you see fit.

Tuples of type `TranscriptRecord` contain:

```
create type TranscriptRecord as (
        code  char(8), -- UNSW-style course code (e.g. COMP1021)
        term  char(4), -- term code (e.g. 18s2, 20T3)
        name  text,    -- short name of the course's subject
        mark  integer, -- numeric mark acheived
        grade char(2), -- grade code (e.g. FL,UF,PS,CR,DN,HD)
        uoc   integer  -- units of credit awarded for the course
);
```

Note that this type is already in the database so you won't need to define it if you wish to use it.

Q2 (6 marks)

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

A number of different rule 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 dastabase are given in the Grades and Rules page.

The important tables involved in defining rules:

- `rules(id,name,type,min_req,max_req,ao_group,description)`
  where the most important fields are
  - `type` ... one of the types described in the Grades and Rules page
  - `min_req` ... min requirement to satisfy this rule (could be UOC or a count)
  - `max_req` ... max requirement for this rule (could be UOC or a count)
  - `ao_group` ... academic object group associated with the rule
- `acad_object_groups(id, name, gtype, glogic, gdefby, negated, parent, definition)`

where the most important fields are:
    - `gtype` ... what kind of objects in the group (all objects are the same type, either subjects or streams)
    - `gdefby` ... how the group is defined (by a list of codes and/or patterns)
    - `definition` ... where course/stream codes and/or patterns are given
- `program_rules(program,rule)` associates a rule to a program
- `stream_rules(stream,rule)` associates a rule to a stream

Q3 (8 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:

```
./prog  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  NameOfRule
```

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

The strategy for ordering the "to be completed" info

- do all CC rules first, stream CC's before program CC's
- then do all PE rules, stream PE's before programPE's
- then do GE rules, then FE rules

In other words, most specific to least specific.

Within groups (e.g stream CC's) order by `Rules.id`. For CC rules, simply print a list of 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 `mymyunsw.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 mymyunsw                    ... remove any existing DB
$ createdb mymyunsw                  ... create an empty database
$ psql mymyunsw -f ..../mymyunsw.dump     ... load the MyMyUNSW schema and data
$ psql mymyunsw -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).