COMP3311 22T1 Database Systems

Assignment 1
Queries and Functions on MyMyUNSW

Last updated: **Monday 7th March 2:57am**
Most recent changes are shown in red ... older changes are shown in brown.

[Assignment Spec] [Database Design] [Schema in SQL] [check1.sql] [Examples] [Fixes+Updates]

Aims

This assignment aims to give you practice in

- reading and understanding a moderately large relational schema (MyMyUNSW)
- implementing SQL queries and views to satisfy requests for information
- implementing SQL/PLpgSQL functions to aid in satisfying requests for information

The goal is to build some useful data access operations on the MyMyUNSW database, which contains a wealth of information about what happens at UNSW. One aim of this assignment is to use SQL queries (packaged as views) to extract such information. Another is to build SQL/PlpgSQL functions that can support higher-level activities, such as might be needed in a Web interface.

A theme of this assignment is "dirty data". As I was building the database, using a collection of reports from UNSW's information systems and the database for the academic proposal system (MAPPS), I discovered that there were some inconsistencies in parts of the data (e.g. duplicate entries in the table for UNSW buildings, or students who were mentioned in the student data, but had no enrolment records, and, worse, enrolment records with marks and grades for students who did not exist in the student data). I removed most of these problems as I discovered them, but no doubt missed some. Some of the exercises below aim to uncover such anomalies; please explore the database and let me know if you find other anomalies.

Summary

Submission:	Submit required files on moodle
Required Files:	`ass1.sql` (contains both SQL views and SQL/PLpgSQL functions)
Deadline:	21:00 Friday 18 March
Marks:	**15 marks** toward your total mark for this course
Late Penalty:	0.1 marks off the ceiling mark for each hour late, no submission is accepted 5 days (120h) after the deadline

How to do this assignment:

- read this specification carefully and completely
- create a directory for this assignment
- copy the supplied files into this directory
- login to `d.cse` and run your PostgreSQL server

- load the database and start exploring
- complete the tasks below by editing `ass1.sql`
- submit `ass1.sql` via moodle

Details of the above steps are given below. Note that you can put the files wherever you like; they do not have to be under your `/localstorage` directory. You also edit your SQL files on hosts other than `d.cse`. The only time that you need to use `d.cse` is to manipulate your database.

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.

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

- no easy way to swap classes once enrolled
- no representation for degree program structures
- poor integration with the UNSW Online Handbook

The first point is inconvenient, since it means that the only way for a student to change tute classes is to drop the course and re-enrol into the course, selecting th new tute. If the course is already full, students would be unwilling to drop the course in case someone else grabs their place before they can re-enrol.
The second 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

In this assignment, you will be working with two instances of a database to hold information about academic matters at UNSW. The first database instance (**mymy1**) contains data from 2000 to 2015. The second database instance (**mymy2**) will contain data from 2014 to 2019. Note that all `People` data about students, and much of the `People` data about staff is synthetic.

Doing this Assignment

The following sections describe how to carry out this assignment. Some of the instructions must be followed exactly; others require you to exercise some discretion. The instructions are targetted at people doing the assignment on d.cse. If you plan to work on this assignment at home on your own computer, you'll need to adapt the instructions to "local conditions".

If you're doing your assignment on the CSE machines, some commands must be carried out on `d.cse`, while others can (and probably should) be done on a CSE machine other than `d.cse`. In the examples below, we'll use `vxdb$` to indicate that the comand must be done on `d.cse` and `cse$` to indicate that it can be done elsewhere.

Setting Up

The first step in setting up this assignment is to set up a directory to hold your files for this assignment.

```
~~cse$ mkdir /my/dir/for/ass1~~
~~cse$ cd /my/dir/for/ass1~~
~~cse$ cp /home/cs3311/web/22T1/assignments/ass1/ass1.sql ass1.sql~~
... or ...
vxdb$ cd /localstorage/YourZid
vxdb$ cp /home/cs3311/web/22T1/assignments/ass1/ass1.sql ass1.sql
```

The next step is to set up your database:

```
... login to d.cse and source env as usual ...
vxdb$ dropdb mymy   ... if you already had such a database
vxdb$ createdb mymy
vxdb$ bzcat /home/cs3311/web/22T1/assignments/ass1/mymy1.dump.bz2 | psql mymy ...or my
vxdb$ psql mymy
... examine the database contents ...
```

Note that the database dump is quite large. It's not worth copying it into your assignment directory on the CSE servers, because you only need to read it once to build your database (see below). The database loading should take less than 60 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 will 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 `mymy1.dump` file is 50MB in size; copying it under your home directory is not a good idea).

If you have other large databases under your PostgreSQL server on d.cse or you have large files under your `/localstorage/`*YOU*`/` directory, it is possible that you will exhaust your d.cse disk quota. In particular, you will not be able to store two copies of the MyMyUNSW database under your d.cse server. The solution: remove any existing databases before loading your MyMyUNSW database.

If you're running PostgreSQL at home, the file `ass1.zip` contains copies of the files: `mymy1.dump.bz2`, `mymy2.dump.bz2`, `ass1.sql` to get you started. If you copy `ass1.zip` to your home computer, unzip it, and perform commands analogous to the above, you should have a copy of the MyMyUNSW database that you can use at home to do this assignment.

Think of some questions you could ask on the database (e.g. like the ones in Prac Exercises and Online Exercises) and work out SQL queries to answer them.

One useful query is

```
mymy=# select * from dbpop();
```

This will give you a list of tables and the number of tuples in each. Some tables are empty, and are not relevant to this assignment. They are included for the sake of "completeness", i.e. to show what kinds of data might be stored in a real database to replace MyUNSW/NSS.

Your Tasks

Answer each of the following questions by typing SQL or PLpgSQL into the `ass1.sql` file. You may find it convenient to work on each question in a temporary file, so that you don't have to keep loading *all* of the other views and functions each time you change the one you're working on. Note that you can add as many auxuliary views and functions to `ass1.sql` as you want. However, make sure that *everything* that's required to make all of your views and functions work ends up in the `ass1.sql` file before you submit.

Q1 (1 mark)

Define an SQL view `Q1(unswid,name)` that gives the student id and name of any student who has enrolled in more than 4 distinct programs at UNSW. The name should be take from the `People.name` field for the student, and the student id should be taken from `People.unswid`.

Q2 (1 marks)

Define an SQL view `Q2(unswid,name,course_cnt)` that gives the unswid and name of the person(s) who has been course tutor of the most courses at UNSW and the number of courses they have been course tutor for. In the database, the course tutor has the role of `Course Tutor`.

Q3 (1 marks)

Define an SQL view `Q3(unswid,name)` that gives all the distinct international students who have enrolled in the course offered by the `School of Law` (refers to the `OrgUnits.Name`) and got a mark higher than 85.

Q4 (1 marks)

Define a SQL view `Q4(unswid,name)` that gives all the distinct local students who enrolled in COMP9020 and COMP9331 (refer to `Subjects.code`) in the same term.

Q5 (2 marks)

For the mymy1 database instance ... Define an SQL view `Q5a(term,min_fail_rate)` that gives the term and the minimum fail rate of the course COMP3311 from year 2009 to year 2012.

For the mymy2 database instance ... Define an SQL view `Q5b(term,min_fail_rate)` that gives the term and the minimum fail rate of the course COMP3311 from year 2016 to year 2019.

Note:
The `term` should be taken from `Terms.name`;
Only count the students with valid marks (not null), fail rate = (number of students with mark less than 50 / number of students with mark);
Round `min_fail_rate` to the nearest 0.0001. (i.e. if minimum fail rate = 0.01, then return 0.0100; if

minimum fail rate = 0.01234, then return 0.0123; if minimum fail rate = 0.02345, then return 0.0235). This rounding behaviour is different from the IEEE 754 specification for floating point rounding which PostgreSQL uses for `float/real/double` precision types. PostgreSQL only performs this type of rounding for `numeric` and `decimal` types.

Q6 (1 mark)

Define an SQL function (SQL, *not* PLpgSQL) called `Q6(id integer, code text)` that takes as parameters: a `People.id` value (i.e. an internal database identifier) and a `Subjects.code` value (i.e. a subject code), and returns the student's mark for the course with the given subject code. If either id or code is invalid, return NULL as the result.

The function must use the following interface:

```
create or replace function Q6(id integer, code text) returns integer
```

Q7 (1 marks)

Define an SQL function (SQL, *not* PLpgSQL) called `Q7(year integer, session text)` that takes as parameters the year (e.g. 2019) and a session (aka term e.g. 'T1'), and returns a list of all the postgraduate COMP courses (refers to `Subjects.code` starting with COMP) offered at the given year and session. An postgraduate course is the one whose `Subjects.career` is PG.

The function must use the following interface:

```
create or replace function Q7(year integer, session text)
  returns table (code text)
```

Q8 (2 marks)

Define a PLpgSQL function `Q8(zid integer)`, which takes a student zid (`People.unswid`) and produces a term transcript as a table of `TermTranscriptRecords`. Each transcript record should contain information about the student's attempt of a term. Records should appear ordered chronologically by term.

Use the following definition for the transcript tuples:

```
create type TermTranscriptRecord as (
        term            char(4), -- term code (e.g. 98s1)
        termwam         integer, -- numeric term WAM acheived
        termuocpassed   integer  -- units of credit passed this term
);
```

Note that this type is already defined in the database. `term` in a record can be obtained by `termName(terms.id)`. But it is text type. Use `CAST (termName(terms.id) AS char(4))` to convert it into char(4) type.

Only count a UOC value when calculating `termuocpassed` if the student actually passed the course (i.e. has a grade from the set {**SY, PT, PC, PS, CR, DN, HD, A, B, C**}) or has an **XE** grade (for credit from exchange) or a **T** grade (transferred credit) or a **PE** grade (professional experience) or a **RC** or **RS** grade (research courses)). A null grade or any grade other than those just mentioned should not be treated as a pass. (For simplicity, grade **GP** and **EC** are treated as unpassed)

Only results with associated marks are included in the `termwam` calculation. If no mark or grade is available for a course (a null mark or grade), do not consider it when calcualting `termwam`. (i.e. grade SY, PT, XE, T, PE, RC and RS...or other grade-only results, they normally have a null mark).

`termuocpassed` and `termwam` are computed as follows:

```
termuocpassed = ΣUOC for all courses passed in this term
termwam = Σ(M*U)/ΣU for all courses completed (marked) in this term
M = mark received in a course
U = units of credit for a course
```

Round the WAM value to the nearest integer and refer to the note of Q5. (i.e. the nearest integer of 10.49 is 10; the nearest integer of 10.50 is 11) If no `termwam` or `termwam` is 0 this term, set `termwam` as null; If no courses have been passed this term, set `termuocpassed` as null. Examples can be found in `Example` page.

At the end of the transcript, add an extra `TermTranscriptRecord` which contains

```
('OVAL', overallwam, overalluocpassed)
```

where the `overalluocpassed` and `overallwam` are computed as follows:

```
overalluocpassed = ΣUOC for all courses passed
overallwam = Σ(M*U)/ΣU for all courses completed (marked)
M = mark received in a course
U = units of credit for a course
```

The calculation rules are similar between the term and overall. Round the WAM value to the nearest integer. If no `overallwam` or `overallwam` is 0, just set `overallwam` as null; If no courses have been passed overall, set `overalluocpassed` as null.

For simplicity, ignore the situiation of degree changes. If a student zid is invalid, return an empty table. See the example in `Example` page.

How can you find interesting transcripts to look at? The answer is to think of some properties of a transcript that might make it interesting, and then ask a query to get information about any students who have these kinds of transcripts. I used the following query to find some students. Work out what it does and then try variations to find other kinds of "interesting" students:

```
select p.unswid,pr.code,termName(min(pe.term)),count(*)
from   People p
         join Program_enrolments pe on (pe.student=p.id)
         join Programs pr on (pe.program=pr.id)
where  pr.code like '3%'
group  by p.unswid,pr.code
having count(*) > 5;
```

Q9 (2 marks)

An important part of defining academic rules in MyMyUNSW is the ability to define groups of academic objects (e.g. groups of subjects, streams or programs) In MyMyUNSW, groups can be defined in three different ways:

- `enumerated` by giving a list of objects in a *`X_members`* table
- `pattern` by giving a pattern that identifies all relevant objects
- `query` by storing an SQL query which returns a set of object ids

In all cases, the result is a set of academic objects of a specific type (given in the `gtype` attribute).

Write a PLpgSQL function `Q9(gid integer)` that takes the internal ID of an academic object group and returns the codes for all members of the academic object group, including any child groups. Associated with each code should be the type of the corresponding object, either `subject`, `stream` or `program`.

You should return distinct codes (i.e. ignore multiple versions of any object), and there is no need to check whether the academic object is still being offered.

The function is defined as follows:

```
create or replace function Q9(gid integer) returns setof AcObjRecord
```

where `AcObjRecord` is already defined in the database as follows:

```
create type AcObjRecord as (
        objtype text,  -- academic object's type e.g. subject, stream, program
        objcode text   -- academic object's code e.g. COMP3311, SENGA1, 3978
);
```

Groups of academic objects are defined in the tables:

- `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
  - `gdefby` ... how the group is defined
  - `definition` ... where queries or patterns are given
- `program_group_members(program, ao_group)` ... for enumerated program groups
- `stream_group_members(stream, ao_group)` ... for enumerated stream groups
- `subject_group_members(subject, ao_group)` ... for enumerated subject groups

There are a wide variety of patterns. You should explore the `acad_object_groups` table yourself to see what's available. To give you a head start, here are some existing patterns and what they mean:

- `COMP2###` ... any level 2 computing course (e.g. COMP2911, COMP2041)
- `COMP[34]###` ... any level 3 or 4 computing course (e.g. COMP3311, COMP4181)
- `####1###` ... any level 1 course at UNSW
- `(COMP|SENG|BINF)2###` ... any level 2 course from CSE
- `COMP1917,COMP1927` ... core first year computing courses
- `COMP1###,COMP2###` ... same as `COMP[12]###`

You do *not* need to handle any of the following types of academic object groups:

- any groups defined using a query (`gdefby='query'`)
- any groups defined using negation (`negated=true`)
- any groups defined by a pattern which includes `'FREE'`, `'GEN'` or `'F='` as a substring

For any group like the above, simply return no reults (zero rows). You can also ignore the `glogic` field; treat them all as `or` groups.

If any group has a child group containing FREE, ignore just the child group. For pattern groups, you do not need to check whether codes used in the pattern correspond to real objects in the relevant table (e.g. a pattern string may contain a subject code which does not exist in the `Subjects` table)

Your function should be able to expand any pattern element from the above classes of patterns (i.e. pattern elements that include #, `[...]` and `(...|...)`). If patterns include `{xxx;yyy;zzz}` alternatives, include all of the alternatives as group members (i.e. as if they were `xxx,yyy,zzz`). If patterns have child patterns, include all of the acdemic objects in the child patterns. You can recognise that a group with `id=X` has child groups, by the existnce of other academic groups with `parent=X`.

Hint: In order to solve this, you'll probably need to look in the PostgreSQL manual at Section 9.4 "String Functions and Operators" and Section 42.5.4 "Executing Dynamic Commands".

Q10 (1 marks)

Define a PLpgSQL function that takes a subject code and returns the set of all subjects that include this subject in their pre-reqs. This kind of function could help in planning what courses you could study in subsequent semesters.

The function is defined as follows:

```
create or replace function Q10(code text) returns setof text ...
```

You only need to consider literal subject codes (e.g. COMP1234) in the pre-reqs. If a pre-req object group contains a pattern, ignore the pattern.

Hint: This function can probably make use of (a variation of) the `Q9()` function.

Submission and Testing

We will test your submission as follows:

- create a testing subdirectory
- create a new database *`TestingDB`* and initialise it with `mymy1.dump`
- run the command: `psql TestingDB -f ass1.sql` (using your `ass1.sql`)
- load and run the tests in the `check1.sql` script
- repeat the above for `mymy2.dump` and `check2.sql`

Note that there is a time-limit on the execution of queries. If any query takes longer than 60 seconds to run (you can check this using the **\timing** flag) your mark for that query will be reduced.

Your submitted code must be *complete* so that when we do the above, your `ass1.sql` will load without errors, which accounts for 2 marks out of 15. If your code does not work when installed for testing as described above and the reason for the failure is that your `ass1.sql` did not contain all of the required definitions, you will lose these 2 marks.

Before you submit, it would be useful to test out whether the files you submit will work by following a similar sequence of steps to those noted above.

Have fun