# Project 1 SQL Views and Functions

# Due: Sun 24 Apr. 23:59

## 1. Aims

This project 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 functions to aid in satisfying requests for information

The goal is to build some useful data access operations on the MyMyUNSW database. A theme of this project 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.

## 2. How to do this project:

- read this specification carefully and completely
- familiarise yourself with the database **schema** (description, SQL schema, summary)
- make a private directory for this project, and put a copy of the **proj1.sql** template there
- you **must** use the create statements in **proj1.sql** when defining your solutions
- look at the expected outputs in the expected_qX tables loaded as part of the **check.sql** file
- solve each of the problems below, and put your completed solutions into **proj1.sql**
- check that your solution is correct by verifying against the example outputs and by using the check_qX() functions
- test that your **proj1.sql** file will load *without error* into a database containing just the original MyMyUNSW data
- double-check that your **proj1.sql** file loads in a *single pass* into a database containing just the original MyMyUNSW data
- submit the project via give.

## 3. 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 often called NSS.

UNSW has spent a considerable amount of money ($80M+) 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 waiting lists for course or class enrolment
- no representation for degree program structures
- poor integration with the UNSW Online Handbook

The first point is inconvenient, since it means that enrolment into a full course or class becomes a sequence of trial-and-error attempts, hoping that somebody has dropped out just before you attempt to enrol and that no-one else has grabbed the available spot.

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

NSS contains data about student, 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.

## 4. Setting Up

To install the MyMyUNSW database under your Grieg server, simply run the following two commands:

$ **createdb proj1**
$ **psql proj1 –f /home/cs3311/web/16s1/proj/proj1/mymyunsw.dump**

If you've already set up PLpgSQL in your template1 database, you will get one error message as the database starts to load:

psql:mymyunsw.dump:*NN*: ERROR:  language "plpgsql" already exists

You can ignore this error message, but any other occurrence of ERROR during the load needs to be investigated.

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

```
SET
SET
SET
SET
SET
psql:mymyunsw.dump:NN: ERROR:  language "plpgsql" already exists
... if PLpgSQL is not already defined,
... the above ERROR will be replaced by CREATE LANGUAGE
SET
SET
SET
CREATE TABLE
CREATE TABLE
... a whole bunch of these
CREATE TABLE
ALTER TABLE
ALTER TABLE
... a whole bunch of these
ALTER TABLE
```

Apart from possible messages relating to plpgsql, you should get no error messages. The database loading should take less than 60 seconds on Grieg, assuming that Grieg is not under heavy load. (If you leave your project until the last minute, loading the database on Grieg 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 mymyunsw.dump file is 50MB in size; copying it under your home directory or your /srvr directory is not a good idea).

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

If you are running PostgreSQL at home, the file proj1.tar.gz contains copies of the files: mymyunsw.dump, proj1.sql to get you started. You can grab the check.sql separately, once it becomes available. If you copy proj1.tar.gz to your home computer, extract 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 project.

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 proj1**

... PostgreSQL welcome stuff ...
proj1=# **\d**
... look at the schema ...
proj1=# **select * from Students;**
... look at the Students table ...
proj1=# **select p.unswid,p.name from People p join Students s on (p.id=s.id);**
... look at the names and UNSW ids of all students ...
proj1=# **select p.unswid,p.name,s.phone from People p join Staff s on (p.id=s.id);**
... look at the names, staff ids, and phone #s of all staff ...
proj1=# **select count(*) from CourseEnrolments;**
... how many course enrolment records ...
proj1=# **select * from dbpop();**
... how many records in all tables ...
proj1=# **select * from transcript(3197893);**
... transcript for student with ID 3197893 ...
proj1=# ... etc. etc. etc.
proj1=# **\q**

You will find that some tables (e.g. Books, Requirements, etc.) are currently unpopulated; their contents are not needed for this project. You will also find that there are a number of views and functions defined in the database (e.g. dbpop() and transcript() from above), which may or may not be useful in this project.

**Summary on Getting Started**

To set up your database for this project, run the following commands in the order supplied:
$ **createdb proj1**
$ **psql proj1 -f /home/cs3311/web/16s1/proj/proj1/mymyunsw.dump**
$ **psql proj1**
... run some checks to make sure the database is ok
$ **mkdir *Project1Directory***
... make a working directory for Project 1
$ **cp /home/cs3311/web/16s1/proj/proj1/proj1.sql *Project1Directory***

The only error messages produced by these commands should be those noted above. If you omit any of the steps, then things will not work as planned.

**Notes**

**Read these** before you start on the exercises:

- the marks reflect the relative difficulty/length of each question
- use the supplied **proj1.sql** template file for your work
- you may define as many additional functions and views as you need, provided that (a) the definitions in proj1.sql are preserved, (b) you follow the requirements in each question on what you are allowed to define
- make sure that your queries would work on any instance of the MyMyUNSW schema; don't customise them to work just on this database; we may test them on a different database instance

- do not assume that any query will return just a single result; even if it phrased as "most" or "biggest", there may be two or more equally "big" instances in the database
- you are not allowed to use ***limit*** in answering any of the queries in this project
- when queries ask for people's names, use the Person.name field; it's there precisely to produce displayable names
- when queries ask for student ID, use the `People.unswid` field; the `People.id` field is an internal numeric key and of no interest to anyone outside the database
- unless specifically mentioned in the exercise, the order of tuples in the result does not matter; it can always be adjusted using `order by`. In fact, our check.sql will order your results automatically for comparison.
- the precise formatting of fields within a result tuple **does** matter; e.g. if you convert a number to a string using to_char it may no longer match a numeric field containing the same value, even though the two fields may look similar
- develop queries in stages; make sure that any sub-queries or sub-joins that you're using actually work correctly before using them in the query for the final view/function
- You can define either SQL views OR SQL functions to answer the following questions.

An important note related to marking:

- make sure that your queries are reasonably efficient (defined as taking < 30 seconds to run)
- use psql's \timing feature to check how long your queries are taking; they must each take less than 30000 ms
- queries that are too slow will be **penalised by half of the mark for that question**, even if they give the correct result

Each question is presented with a brief description of what's required. If you want the full details of the expected output, take a look at the expected_qX tables supplied in the checking script.

## 5. Tasks

To facilitate the semi-auto marking, please pack all your SQL solutions into view or function as defined in each problem (see details from the solution template we provided).

### Q1(2 marks)

Define an SQL view `Q1(unswid,name)` that gives the student id and name of any student who has studied more than 55 courses at UNSW. The name should be taken from the `People.name` field for the student, and the student id should be taken from `People.unswid` field.

### Q2 (2 marks)

Define an SQL view `Q2(name,school,starting)` which gives the details about all of the current Heads of School at UNSW. The view should return the following details about each Head:

- their name (use the `name` field from the `People` table)
- the school (use the `longname` field from the `OrgUnits` table)
- the date that they were appointed (use the starting date from the `Affiliations` table)

Since there is some dirty-looking data in the `Affiliations` table, you will need to ensure that you return only legitimate Heads of School. Apply the following filters together:

- only choose people whose role is 'Head of School'
- only choose people who have not finished in the role
- only choose people for whom this is their primary role
- only choose organisational units whose type is actually 'School'

**Q3 (2 marks)**
It was claimed that having both `uoc` and `eftsload` in the `Subjects` table was unnecessary, since one could be determined from the other (e.g., `uoc/48 == eftsload`). We could test this by calculating the ratio of `uoc/eftsload` for all subjects to see whether they adhere to the `uoc/eftsload` ratio being constant. Since the `eftsload` values in data files are truncated to three decimal places, some of the ratios won't be exact; to allow for this, take only one decimal place in your ratio calculations.
Define an SQL view to produce a list of distinct `uoc/eftsload` values and the number of subjects which has each value, as follows:
```
create or replace view Q3(ratio, nsubjects) as ...
```
**Note:** use `numeric(4,1)` to restrict the precision of the ratios, and be careful about typecasting between integers and reals. You can ignore any subjects with `eftsload` values that are either `NULL` or `zero`.

**Q4 (2 marks)**
Define an SQL view `Q4(name,ncourses)` that prints the name of the person(s) who has been lecturer-in-charge (LIC) for the largest number of courses at UNSW and the number of courses that they have been LIC for.
**Note**: In the database, the LIC has the role of "Course Convenor". It is possible to return more than one lecturer since they may be in charge of the same number of courses.

**Q5 (3 marks)**
Define SQL views `Q5a(id)`, `Q5b(id)` and `Q5c(id)`, which give the student IDs of, respectively

- all students enrolled in 05S2 in the Computer Science (3978) degree
- all students enrolled in 05S2 in the Software Engineering (SENGA1) stream
- all students enrolled in 05S2 in degrees offered by School of Computer Science and Engineering (CSE)

Note: the student IDs are the UNSW id's (i.e. student numbers) defined in the `People.unswid` field.

**Q6 (2 marks)**
Dealing with `Semesters.id` values is not ideal when referring to semesters. CSE typically uses shorthand symbolic names like "12s1" and "00s2" to refer to semesters. Write an SQL function that takes as parameter a `Semesters.id` value and returns a symbolic semester name, consisting of the last two digits of the year, and the term code (with a lower-case letter). The function should be defined as follows:
```
create or replace function Q6(integer) returns text
```
If the parameter value does not correspond to a known semester, the function should simply return empty. Note that you should return lower case for symbolic names, rather than upper case.

**Q7 (3 marks)**
UNSW management occasionally expresses concern that the university is too heavily dependent on international student enrolments. Define a view `Q7(semester,percent)` that will help them to monitor this by indicating the percentage of international students enrolled in each semester. Each tuple in the view should contain the following:

- the name of the semester in the format 09s1, 09s2, 10s1, etc (Note: you can re-use the function defined in Q6).
- the fraction (internationals/total enrolments) as `numeric(4,2)`

Each student's status as local or international can be determined from the `Students.stype` field.

You should compute the number of enrolled students by considering just the table `Program_enrolments` for each semester. You can assume that no student is registered as both local and international in the same semester. Also, only consider the main semesters (s1 and s2) and only consider years from 2005 onwards (inclusive). Do the fraction calculation using floating point values to achieve the highest precision.

**Q8 (2 marks)**
Some of the data in this database is a little sparse; for example, most of the courses have no course staff specified. We want to find extreme examples regarding a subject whose relevant courses has been offered quite many times, but has never had any staff allocated to it according to the database.
Define an SQL view `Q8(subject,nOfferings)` that gives a list of subjects which have been offered more than 25 times, but have never had any staff associated with them (via the `Course_staff` table). Each tuple in the view should contain the following:

- the subject code (`Subject.code` field) and name (`Subject.name` field), in the format e.g. COMP3311 Database Systems
- the number of times that this subject has been offered as a Course

**Q9 (2 marks)**
A professor wants to hire some research assistants who had studied all the courses with subject code starting with "COMP34". For example, both COMP3411 and COMP3431 belong to "COMP34" series, to list a few. Define an SQL view `Q9(unswid,name)` that gives a list of students which had enrolled in all the courses in the "COMP34" series.
The name should be taken from the `People.name` field for the student, and the student id should be taken from `People.unswid` field.

**Q10 (5 marks)**
Write an SQL view which gives all the students who had been enrolled in all the popular level-9 COMP subjects (i.e., subject code starting with 'COMP9') with good performance. A subject was considered as popular if it was taught in every major semester (i.e., S1 and S2) from 2002 to 2013 (both inclusive). Good performance means that the student achieves either 'HD' or 'DN' in all the popular subjects (only 'HD' and 'DN' are considered as good performance). A student will not be returned as a result if he/she got a poor grade on any popular subject. We only consider popular subjects in series 'COMP9%', since it was unlikely, for example, for a CSE student to be enrolled in subjects offered by other schools.
Define an SQL view `Q10(unswid,name)` that gives a list of students which satisfy the above constraints. The `name` should be taken from the `People.name` field, and `unswid` should be taken from `People.unswid` field.
Note: it is possible that a student was enrolled multiple times in a given subject. For example, one may fail the course in 03S1, and then be re-enrolled in the same course in 04S1.
Also, you may not be allowed to use `Semesters.id` values directly anywhere in your query (e.g., you may not refer to 07S1 as 146). Try to use symbolic name, e.g., '00S1', to refer to any semester in your SQL.

## 6. Submission

You can submit this project by doing the following:

- Ensure that you are in the directory containing the file to be submitted.

- Type “give cs3311 proj1 proj1.sql”
- If you submit your project more than once, the last submission will replace the previous one
- **To prove successful submission, please take a screenshot as assignment submission manual shows and keep it by yourself.**
- If you have any problems in submissions, please email to longyuan@cse.unsw.edu.au. You can also ask questions about this project in our project group, we will answer your question as soon as possible.

The proj1.sql file should contain answers to all of the exercises for this project. 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 proj1.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 proj1** ... remove any existing DB
$ **createdb proj1** ... create an empty database
$ **psql proj1 -f /home/cs3311/web/16s1/proj/proj1/mymyunsw.dump** ... load the MyMyUNSW schema and data
$ **psql proj1 -f /home/cs3311/web/16s1/proj/proj1/check.sql** ... load the checking code
$ **psql proj1 -f proj1.sql** ... load your solution

Note: if your database contains any views or functions that are not available in a file somewhere, you should put them into a 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 proj1.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 I need to manually fix problems with your proj1.sql file in order to test it (e.g. change the order of some definitions), you will be fined via half of the mark penalty for each problem.

## 7. Late Submission Penalty

10% reduction for the 1st day, then 30% reduction.