COMP9311 19T2 Database Systems

Assignment 2
SQL, Views, PLpgSQL, Functions

Last updated: **Monday 8th July 3:51pm**
Most recent changes are shown in red ... older changes are shown in brown.

[Assignment Spec] [Database] [Schema Summary] [Testing] [Samples]

Deadline: 1st August 9pm (Week 9, Thursday)

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 PLpgSQL functions to aid in satisfying requests for information

The goal is to build some useful data access operations on the MyMyUNSW database. You will find some empty tables. It is OK as they are not necessary for this assignment.

This database has been built over time by Dr. John Shepherd to capture the data behind many student record management systems at USNW. A theme of this assignment is "dirty data". As John was building the database, using a collection of reports from UNSW's information systems, he discovered that there were some inconsistencies in parts of the data (e.g. 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). He removed most of these problems as he discovered them, but no doubt missed some. Some of the exercises below aim to uncover such anomalies; please explore the database and see if you can find other anomalies.

Note also that, as the database was constructed, he made some small changes to the schema, in order to deal with the data he had available. For example, there was no starting time for some of the staff affiliations, and so I had to remove the `not null` constraint on `Affiliation.starting` in order to load a whole bunch of staff affiliation data. You should make sure that you check the latest version of the schema and use that; if in doubt, check the meta-data in the database itself (using `psql`'s **`\d`** command).

Obviously, for assignments and other learning exercises for the database course, John has altered and anonymise the data completely. This means the student IDs are not real and the transcript/enrolment records are not real. If you happen to find a student ID that exists, it will be an accidental match and the matching records have no link to the real person. But we doubt that a real ID exists in the database :-)

Summary

Submission: Login to Course Web Site > Assignments > ass2 > [Submit] > upload `ass2.sql`

Deadline: Thursday 1st August 9pm

Late Penalty: 0.09 *marks* off the ceiling mark for each hour late

This assignment contributes **25%** toward your total mark for this course.
The mark for each question indicates its level of difficulty.
The total marks for the questions sum to 25.
Your assignment *Mark* will be mapped into a mark out of 10 by *10*Mark/25*.

How to do this assignment:

- read this specification carefully and completely
- familiarise yourself with the database schema (description, SQL schema, summary)
- make a private directory for this assignment, and put a copy of the `ass2.sql` template there
- you **must** use the `create` statements in `ass2.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 `ass2.sql`
- check that your solution is correct by verifying against the example outputs and by using the `check_qX()` functions
- test that your `ass2.sql` file will load *without error* into a database containing just the original MyMyUNSW data
- double-check that your `ass2.sql` file loads in a *single pass* into a database containing just the original MyMyUNSW data
- submit the assignment via WebCMS as described above

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.

Setting Up

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

```
$ createdb ass2
$ zcat /home/cs9311/web/19T2/assignments/2/mymyunsw.dump.gz | psql ass2
```

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 40 seconds on Grieg, assuming that Grieg is not under heavy load. (If you leave your assignment 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 34MB 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.

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 ...
ass2=# 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 ...
ass2=# 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 ...
ass2=# select count(*) from CourseEnrolments;
... how many course enrolment records ...
ass2=# select * from dbpop();
... how many records in all tables ...
ass2=# select * from transcript(3231850);
... transcript for student with ID 3231850 ...
ass2=# ... etc. etc. etc.
ass2=# \q
```

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

Summary on Getting Started

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

```
$ createdb  ass2
$ zcat /home/cs9311/web/19T2/assignments/2/mymyunsw.dump.gz | psql ass2
$ psql  ass2
... run some checks to make sure the database is ok
$ mkdir  Assignment2Directory
... make a working directory for Assignment 2
$ cp  /home/cs9311/web/19T2/assignments/2/ass2.sql  Assignment2Directory
```

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. If you subsequently ask questions on the MessageBoard, where it's clear that you have *not* done and checked the above steps, the questions will not be answered.

Notes

Read these before you start on the exercises:

- the marks reflect the relative difficulty/length of each question
- use the supplied `ass2.sql` template file for your work
- you may define as many additional functions and views as you need, provided that (a) the definitions in `ass2.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 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 assignment
- do not use values of `id` fields if you can refer to tuples symbolically; e.g. if a question asks about lecture classes, do **not** use the fact the `id` of the lecture class type is 1 and check for `classtypes.id=1`; instead check for `classtypes.name='Lecture'`
- 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`
- 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

An important note related to marking:

- make sure that your queries are reasonably efficient (defined as taking < 15 seconds to run)
- use `psql`'s `\timing` feature to check how long your queries are taking; they must each take less than 20000 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.

Exercises

Q1 (2 marks)

Define an SQL view `Q1(name,school,starting)` which gives details about all of the current Heads of School in 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 `Affiliation` table)

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

- 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'`

Q2 (2 marks)

Using the view from the previous question, write a new view `Q2(status,name,school,starting)` which returns a table containing only the longest-serving and most-recent current Heads of School. Each tuple in the view result should contain the following:

- the status: either the string `'Longest serving'` or the string `'Most recent'` (cast to type `text`)
- the same three values (name, school, starting-date) for each Head of School

You should *not* assume that there is only one longest-serving or one most-recent Head of School; several Heads may have started in the role on the same day.

You must also ensure that no warning messages are generated during the loading of your view.

Q3 (1 mark)

Dealing with `Terms.id` values is not ideal when referring to semesters. CSE typically uses shorthand symbolic names like "12s1" and "00s2" to refer to terms. Write an SQL *function* that takes as parameter a `Terms.id` value and returns a symbolic term name, consisting of the last two digits of the year, and the session code (with a lower-case letter).

The function should be defined as follows:

```
create or replace function Q3(integer) returns text
as $$ SQL query $$ language sql;
```

If the parameter value does not correspond to a known term, the the function should simply return a `NULL` value.

Q4 (3 marks)

UNSW management occasionally expresses concern that the university is too heavily dependent on international student enrolments. Write a view `Q4(term,percent)` that will help them to monitor this by indicating the percentage of international students enrolled in each semester. Each tuple in the view result should contain the following:

- the name of the term in the format `09s1`, `09s2`, `10s1`, etc.
- 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 `ProgramEnrolments` 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. Do the fraction calculation using floating point values to achieve the highest precision.

Q5 (3 marks)

In the previous question, we measured enrolments in terms of the number of students enrolled in programs. For funding purposes, the University measures enrolments in terms of full-time equivalent (FTE) students. A full-time enrolment is a student enrolled in 24UOC worth of courses. Thus, the FTE number for a given semester can be computed by looking at the total UOC for enrolled courses in that semester and dividing by 24.

Write a view `Q5(term, nstudes, fte)` that gives, for each main semester (S1 and S2) from 2000 S1 to 2010 S2, the number of students enrolled and the FTE for that semester. Each tuple in the view result should contain the following:

- the name of the term in the format `09s1`, `09s2`, `10s1`, etc.
- the total number of distinct students enrolled in courses in that semester
- the FTE (total enrolled UOC / 24) for that semester as `numeric(6,1)`

You should compute enrolment information from the `CourseEnrolments` table (i.e. no need to consider `ProgramEnrolments`). Do the FTE calculation using floating point values to achieve the highest precision.

Q6 (2 marks)

Some of the data in this database is a little sparse; for example, most of the courses have no `CourseStaff` specified. We want to find extreme examples where a course has been offered very many times, but has never had any staff allocated to it according to the database.

Write a view `Q6(subject,nOfferings)` that gives a list of subjects which have been offered more than 30 times, but have never had any staff associated with them (via the `CourseStaff` table). Each tuple in the view result should contain the following:

- the subject code and title, in the fromat e.g. `COMP3311 Database Systems`
- the number of times that this subject has been offered as a `Course`

Q7 (2 marks)

With the new UNSW timetabling system, the LIC of each course is required to specify what facilities they require in their classrooms as one of the inputs to the timetabling process. Write an SQL function to provide a list of rooms that contain a given facility, that might be used as part of this process. The function takes a parameter giving part of the name of a facility (e.g. "Lectern microphone" or "lectern mic" or ...) and returns the names of all `Rooms` that have a facility matching the parameter and the name of the actual facility. The function is defined as:

```
create function Q7(text) returns setof FacilityRecord
as $$ SQL statement $$ language sql;
```

Note that the `FacilityRecord` type is already included in the database as:

```
create type FacilityRecord as (room text, facility text);
```

The `room` field of each tuple should be a `Rooms.longname` value, while the `facility` filed should be a `Facilities.description` value.

To allow some of flexibility in search for facilities, the function parameter should be treated like a pattern, and should match all facilities whose description contains the string given as the parameter. For example, `'lectern mic'` would matche 'Lectern microphone', while `'microph'` would match 'Lectern microphone', 'Neck microphone' or 'Radio microphone'. Matching should be case insensitive.

Note that, according to the database, no rooms in UNSW have whiteboards. The database reflects reality to some extent, but is clearly not a 100% accurate and certainly not a 100% complete mapping of reality.

Q8 (4 marks)

A function returning the current semester is useful in many contexts (e.g. determining all currently enrolled students). We wish to write a generalisation of this function: one that takes a date and tells us which semester it falls in. This might seem simple enough, given that the `Terms` table contains `starting` and `ending` dates for each semester. However, we want to take a more liberal view of the extent of semesters. We will normally consider that each semester effectively starts one week before the starting date recorded in the `Terms` table. Also, we'll consider that the previous semester extends to the day before the *effective* starting date of the following semester. If the ending date of semester t_1 is less than one week before the starting date of the next semester t_2, then treat the starting date of t_2 as the day after the ending date of t_1.

Some examples to clarify. If `T1.starting='2005-02-28'` and `T2.ending='2005-01-31'`, then the effective starting date for `T1` is `'2005-02-21'` and the effective ending date for `T2` is `'2005-02-20'` (the day before the effective start of `T1`). If `T3.starting='2009-11-30'` and `T4.ending='2009-11-24'`, then the effective starting date for `T3` is `'2009-11-25'`, and the effective ending date for `T4` stays the same `'2009-11-24'`.

Write a PLpgSQL function called `Q7` to find which semester a particular date falls in. Use the following function header:

```
create or replace function Q8(_day date) returns text
as $$ ... PLpgSQL code ... $$ language plpgsql;
```

The parameter is a date (e.g. `'2005-12-25'`). The function returns the term name (e.g. `05s1`) of the semester containing the specified date. If a date is given that corresponds to no known term in the database (e.g. before the first term in the database or after the last term in the database), then `NULL` is returned.

Hint: A `date` value can be "wound backwards" by subtracting an `interval` value, or moved forward by adding an `interval` value.

Q9 (6 marks)

The **transcript** function supplied in the database assumes that the only way that a student can get credit towards their degree is by enrolling in a subject for which they have the pre-reqs and passing that subject. In fact, students can obtain various other kinds of "credit" towards their study to help them finish their degree:

- "advanced standing" gives students credit for some course at UNSW based on a similar course completed at another institution (or in an incomplete degree at UNSW); the student is allocated UOC for the UNSW subject, but it does not count towards their WAM; however, for purposes such as pre-requisites, it is as if the student took the UNSW course
- "substitution" allows a student to take one subject in place of a core subject in their program (e.g. if the original core subject is not available and it is the student's final semester of study); the student is given the UOC for the course actually taken and the course taken counts in their WAM; however, the course taken may be used as a "stand-in" for the substituted course in determining whether they have met their degree requirements
- "exemption" is where a student is deemed to have completed a course at UNSW based on a similar course at another institution, but is not awarded any UOC for the UNSW course; however, they can use it as a pre-requisite for further study at UNSW

Information about such enrolment variations is stored in the two tables:

```
Variations(student, program, subject, vtype, intequiv, extequiv, ...)
ExternalSubjects(id, extsubj, institution, yearoffered, equivto)
```

where each `Variations` tuple shows a variation for one student for a given subject towards a particular program. The `vtype` field indicates what kind of variation it is (`advstanding`, `substitution`, `eemption`). The `intequiv` field references a UNSW subject if the variation is based on a UNSW subject. The `extequiv` references a tuple in the `ExternalSubjects` table if the variation is based on a subject studied at another institution. Only one of `intequiv` or `extequiv` will be "not null". You should examine the contents of these two tables, as well as the file called "`variations.sql`" containing details of some of the variations in the database.

A transcript function has already been loaded into the database, along with a definition of the `TranscriptRecord` type. You can grab a copy of the `transcript()` function definition using PostgreSQL's `\ef` command (see the PostgreSQL manual for details).

```
create type TranscriptRecord as (
        code   char(8),  -- e.g. 'COMP3311'
        term   char(4),  -- e.g. '12s1'
        name   text,     -- e.g. 'Database Systems'
        mark   integer,  -- e.g. 75
        grade  char(2),  -- e.g. 'DN'
        uoc    integer   -- e.g. 6
);

create function
        transcript(_sid integer) returns setof TranscriptRecord
as $$
        ... PLpgSQL code ...
$$ language plpgsql;
```

You should write a new version of the `transcript()` function called `Q9()` which includes variations as well as regular course enrolments. You can use any or all of the code from the supplied `transcript()` function in developing your `Q9()` function. Any variations are displayed at the end of the transcript, after the regular courses, but before the WAM calculation. It should still produce the WAM and UOC count, like the original `transcript` function did, but they will be computed slightly differently (see below). Note that the `Q9()` function has exactly the same type signature as that noted above for the `transcript()` function.

Each variation produces two `TranscriptRecord` tuples. The first tuple gives details of which UNSW subject is being "varied", while the second tuple gives details of the equivalent subject that is used as the basis for the variation.

The details for what first `TranscriptRecord` tuple contains, for the different types of variation is as follows:

advanced standing

first tuple: `(`*`CourseCode`*`, null, 'Advanced standing, based on ...', null, null, `*`UOC`*`)`

- *`CourseCode`* is the course code of the UNSW course for which advanced standing is being granted
- *`UOC`* is the UOC for that particular course

Note that the UOC value should be added into the total UOC displayed in the last `TranscriptRecord` tuple, but should not be included in the WAM calculation.

substitution

first tuple: `(`*`CourseCode`*`, null, 'Substitution, based on ...', null, null, null)`

- *`CourseCode`* is the course code of the UNSW course for which advanced standing is being granted

exemption

first tuple: `(`*`CourseCode`*`, null, 'Exemption, based on ...', null, null, null)`

- *`CourseCode`* is the course code of the UNSW course for which advanced standing is being granted

The second `TranscriptRecord` tuple for each variation will be different depending on whether the substitution is based on an internal (UNSW) or external subject (from another institution).

variation based on a UNSW subject (`intequiv`)

second tuple: `(null, null, 'studying` *`CourseCode`* `at UNSW', null, null, null)`

- *`CourseCode`* is code of the UNSW subject referenced by `Variations.intequiv`

variation based on an external subject (`extequiv`)

second tuple: `(null, null, 'study at` *`Institution`*`', null, null, null)`

- *`Institution`* is name of the institution in the `ExternalSubjects` tuple referenced by `Variations.extequiv`

For the purposes of this exercise, we'll ignore whatever programs the student was enrolled in when they completed the courses. Assume that all courses were part of a single program.

Only a few students in the database have variations. Some example student IDs for testing: 3169329, 3118617, 3270322. You can find more in the "`variations.sql`" file.

Submission

Submit this assignment by doing the following:

Login to Course Web Site > Assignments > ass2 > [Submit] > upload `ass2.sql`

The `ass2.sql` file should contain answers to all of the exercises for this assignment. 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 `ass2.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
$ zcat /home/cs9311/web/19T2/assignments/2/mymyunsw.dump.gz | psql ass2    ... load the MyMyUNSW schema and data
$ psql ass2 -f ..../check.sql   ... load the checking code
$ psql ass2 -f ass2.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 `ass2.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 `ass2.sql` file in order to test it (e.g. change the order of some definitions), **you will be "fined" via a 6 mark penalty on your ceiling mark** (i.e. the maximum you can score is half marks).

Have fun, *Helen*