Assignment 1
SQL, Views, PLpgSQL, Functions

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

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.

A theme of this assignment is "dirty data". As we were building the database, using a collection of reports from UNSW's information systems and the database for the academic proposal system (MAPPS/AIMS), we 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). We removed most of these problems as we discovered them, but no doubt we missed some. Some of the exercises below aim to uncover such anomalies; please explore the database and let us know if you find other anomalies.

Summary

Submission: Login to Course Web Site > Assignments > Assignment 1 > Assignment 1 Specification > Make Submission > upload `a1.sql` > [Submit]

Deadline: Friday 13 April 2018 @ 23.59

Late Penalty: 0.075 *marks* of the total mark (i.e., 15 marks) for each hour late (i.e., 1.8 marks per day).

This assignment contributes **15 marks** 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 15.

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 `a1.sql` template there
- you **must** use the `create` statements in `a1.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 `a1.sql`
- check that your solution is correct by verifying it against the example outputs and by using the `check_qX()` functions
- test that your `a1.sql` file will load *without error* into a database containing just the original MyMyUNSW data
- double-check that your `a1.sql` file loads in a *single pass* into a database containing just the original MyMyUNSW data
- submit the assignment via WebCMS3 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 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, if not with a particularly convenient interface.

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 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.

Setting Up

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

```
$ createdb a1
$ psql a1 -f /home/cs3311/web/18s1/assignments/a1/mymyunsw.dump
```

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

```
SET
SET
SET
SET
SET
SET
CREATE EXTENSION
COMMENT
SET
CREATE DOMAIN
... a few of these
CREATE TABLE
... a whole bunch of these
CREATE TABLE
ALTER TABLE
... a whole bunch of these
ALTER TABLE
```

You should get no ERROR messages. The database loading should take less than 30 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 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 may 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.

Note that the database is called `a1` (when you want to access it via `psql`), while the dump file containing the data is called `mymyunsw.dump`, and 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. And, finally, note that the data was derived from old NSS data, and so may refer to courses, programs and streams that no longer exist, and may not refer to relatively new courses (like COMP1511).

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 a1
... PostgreSQL welcome stuff ...
a1=# \d
... look at the schema ...
a1=# select * from Students;
... look at the Students table ...
a1=# 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 ...
a1=# 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 ...
a1=# select count(*) from Course_enrolments;
... how many course enrolment records ...
a1=# select * from dbpop();
... how many records in all tables ...
a1=# select * from transcript(3197893);
... transcript for student with ID 3231850 ...
a1=# select * from transcript(3227052);
... transcript for student from ADFA ...
a1=# ... etc. etc. etc.
a1=# \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:

```
$ ssh grieg
... and then on Grieg ...
$ priv srvr
$ source /srvr/YOU/env
$ pgs start
... you shut down the server after your last session, didn't you?
$ createdb  a1
$ psql  a1  -f  /home/cs3311/web/18s1/assignments/a1/mymyunsw.dump
$ psql  a1
... run some checks to make sure the database is ok
$ mkdir  Assignment1Directory
... make a working directory for Assignment 1
$ cp  /home/cs3311/web/18s1/assignments/a1/a1.sql  Assignment1Directory
```

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 Forums, 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 `a1.sql` template file for your work
- you may define as many additional functions and views as you need, provided that (a) the definitions in `a1.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 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
- use `psql`'s `\timing` feature to check how long your queries are taking; they must each take less than 10000 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_q`*X* tables supplied in the checking script.

Exercises

Q1 (1 mark)

Define an SQL view `Q1(unswid,name)` that gives the student id and name of any student who has studied more than 65 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`.

Q2 (1.5 marks)

Define an SQL view `Q2(nstudents,nstaff,nboth)` which produces a table with a single row containing counts of:

- the total number of students (who are not also staff)
- the total number of staff (who are not also students)
- the total number of people who are both staff and student

Q3 (1.5 marks)

Define an SQL view `Q3(name,ncourses)` that prints the name of the person (or people) who has (have) been lecturer-in-charge (LIC) of the most courses at UNSW and the number of courses they have been LIC for. In the database, the LIC has the role of "Course Convenor".

Q4 (1.5 marks)

Define SQL views `Q4a(id)`, `Q4b(id)`, and `Q4c(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 CSE

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

The definitions could be used as definitions for these groups in the `student_groups` table, although you do not need to add them into the `Student_groups` table.

Q5 (1.5 marks)

Define an SQL view `Q5(name)` which gives the faculty with the maximum number of committees. Include in the count for each faculty, both faculty-level committees and also the committees under the schools within the faculty. You can use the `facultyOf()` function defined in lectures to assist with this (the function is already available in the database). You can assume that committees are defined only at the faculty and school level. Use the `OrgUnits.name` field as the faculty name.

Q6 (1 mark)

Define an SQL function (SQL, *not* PLpgSQL) called `Q6(integer)` that takes as parameter either a `People.id` value (i.e. an internal database identifier) or a `People.unswid` value (i.e. a UNSW student ID), and returns the name of that person. If the id value is invalid, return an empty result.

The function must use the following interface:

```
create or replace function Q6(integer) returns text ...
```

Q7 (1.5 marks)

Define an SQL function (*not* PLpgSQL) called `Q7(text)` that takes as parameter a UNSW course code (e.g. COMP1917) and returns a list of all offerings of the course for which a Course Convenor is known. Note that this means just the Course Convenor role, not Course Lecturer or any other role associated with the course. Also, if there happen to be several Course Convenors, they should all be returned (in separate tuples). If there is no Course Convenor registered for a particular offering of the course, then that course offering should not appear in the result.

The function must use the following interface:

```
create or replace function Q7(text)
  returns table (course text, year integer, term text, convenor text)
```

The `course` field in the result tuples should be the UNSW course code (i.e. that same thing that was used as the parameter). If the parameter does not correspond to a known UNSW course, then simply return an empty result set.

Q8 (2 marks)

The `transcript(integer)` function from lectures is supplied in the database. Define a new PLpgSQL function `Q8(integer)`, which is based on this function but returns a new kind of transcript record, called `NewTranscriptRecord`, that includes in each row (except the last) the 4-digit program code for the program being studied when the course was studied.

Use the following definition for the new-style transcript tuples:

```
create type NewTranscriptRecord as (
        code  char(8), -- UNSW-style course code (e.g. COMP1021)
        term  char(4), -- semester code (e.g. 98s1)
        prog  char(4), -- program being studied in this semester
        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. In order to get access to the source code for the `transcript()` function, use the following command in `psql` and save the function definition in a local file where you can edit it:

```
a1=# \ef transcript(integer)
```

Q9 (3.5 marks)

An important part of defining academic rules in MyMyUNSW (although we won't look at the rules themselves until a later assignment) 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 particular type

Write a PLpgSQL function `Q9(integer)` that takes the internal ID of an academic object group and returns the codes for all members of the academic object group. Associated with each code should be the type of the corresponding object, either `subject`, `stream` or `program`. For this question, you only need to consider groups defined via a `pattern`. If the supplied object group ID refers to an `enumerated` or `query` type group, you may simply return an empty result. 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(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
        object  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

The ways of specifying object groups are quite flexible, and groups can be defined hierarchically. For this exercise, however, you can ignore groups defined in terms of child groups. You can also ignore negated groups, which would probably result in very large sets of objects. In these cases, simply return an empty result. You can also ignore the `glogic` field.

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)
- `FREE####` ... any free elective; for this case, simply return the pattern itself**
- `GENG####` ... any Gen Ed course; for this case, simply return the pattern itself**
- `####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]###`

Your function should be able to expand any pattern element from the above classes of patterns (i.e. pattern elements that include `#`, `[...]` and `(...|...)`), except for `FREE####` and `GENG####` as noted.

** Note that there are some variations on the FREE and GEN patterns that should also be treated specially. Any pattern element that begins with "FREE" should be returned unchanged. Similarly for the patterns "GEN#####" and "ZGEN####".

Patterns can be qualified by constraint clauses (e.g. /F=ENG) and alternatives can be specified (e.g. {MATH1131;MATH1141}), but you don't need to be able to handle these. If a pattern *does* contain one of these, simply return an empty result.

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

Submission

Submit this assignment by doing the following:

> Login to Course Web Site > Assignments > Assignment 1 > Assignment 1 Specification > Make Submission > upload a1.sql > [Submit]

The a1.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 a1.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 a1                    ... remove any existing DB
$ createdb a1                  ... create an empty database
$ psql a1 -f ..../mymyunsw.dump     ... load the MyMyUNSW schema and data
$ psql a1 -f ..../check.sql  ... load the checking code
$ psql a1 -f a1.sql         ... load your solution
```

Note: if your database contains any views or functions that are not available in the a1.sql file, you should put 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 a1.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 a1.sql file in order to test it (e.g., change the order of some definitions), **you will be "fined" via a 5 mark penalty on your ceiling mark** (i.e., the maximum you can score is 10 out of 15 marks).