Project 1 SQL
COMP9311 22T2

The deadline for project 1 is: July 15th 16:59:59 (Sydney Local Time)

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 PL/pgSQL functions to aid in satisfying requests for information
- The goal is to build some useful data access operations on the MyMyUNSW database. The data may contain some data inconsistencies; however, they won't affect your answers to the project.

2. How to do this project:

- Read this specification carefully and completely
- Familiarize 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 moodle
- For each question, you must output result within 120 seconds on Grieg server.
- **Hardcode is strictly forbidden.**

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, MyUNSW/NSS still has several 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 enroll 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 available courses)
- determining when they have completed all 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 COMP9311 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 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/cs9311/web/22T2/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 exist.
```

You can ignore the above error message, but all other occurrences 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 if you have large files under your /srvr/YOU/ directory, it is possible that you will exhaust your Grieg disk quota. Regardless, it is certain that 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.

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/cs9311/web/22T2/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/cs9311/web/22T2/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.

5. Important Advice Before You Start

The database instance you are given is not a small one. The first thing you should do is get a feeling for what data is there in the database. This will help you understand the schema better and will make the tasks easier to understand. *Tip: study the schema of each table to see how tables are related and try write some queries to explore/ understand what each table is storing.*

```
$ psql proj1
proj1=# \d
... study the schema ...
proj1=# select * from Students;
... look at the data in 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 Course_Enrolments;
... get an idea of the number of records each table has...
proj1=# select * from dbpop();
... how many records in all tables ...
proj1=# ... etc. etc. etc.
proj1=# \q
```

Read these before you start on the exercises:

- The marks reflect the relative difficulty/length of each question.
- Work on the project on the supplied **proj1.sql** template file.
- Make sure that your queries work on any instance of the MyMyUNSW schema; don't customize 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.
- 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.
- We advise developing queries in stages; make sure that any sub-queries or sub-joins that you're using works correctly before using them in the query for the final view/function
- You may define as many additional 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.
- If you meet with error saying something like “cannot change name of view column”, you can drop the view you just created by using command “**drop view VIEWNAME cascade;**” then create your new view again.

Each question is presented with a brief description of what's required. If you want the full details of the expected output, look at the expected_qX tables supplied in the checking script (check.sql) once we release it.

6. Tasks

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

Question 1 (3 marks)

Define a SQL view `Q1(subject_code)`:
Give the codes of `PG`-career subjects that are offered by `School of Computer Science and Engineering` and their `uoc` are more than 12. Where `PG` refers to Postgraduate career in `Subjects.career`. And use `longname` column in the related table to search the school's name. Only consider the subjects which are directly offered by the above school.

- `subject_code` should be taken from `Subjects.code` field.

Question 2 (3 marks)

Define an SQL view `Q2(course_id)` that gives the id of the course that has at least three different types of classes. Only consider courses in year `2002` and term `S1`.

- `course_id` should be taken from `Courses.id` field.

Question 3 (3 marks)

Define a SQL view `Q3(unsw_id, name)` that gives all the `distinct local` students who enrolled in `COMP9311` and `COMP9331` in the same semester. The view should return the following details about each student:

- `unsw_id` should be taken from `People.unswid` field;
- `name` should be taken from `People.name` field.

Question 4 (4 marks)

Define an SQL view `Q4(term, max_fail_rate)` that gives the term and the maximum fail rate of the course COMP9311 from year 2009 to year 2012. 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 max_fail_rate to the nearest 0.0001. (i.e., if maximum fail rate = 0.01 (i.e., 1%), then return 0.0100; if maximum fail rate = 0.01234, then return 0.0123; if maximum fail rate = 0.02345, then return 0.0235). This rounding behavior 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.

- `term` should be taken from `Semesters.name`;
- `max_fail_rate` should be in `numeric` type.

Question 5 (4 marks)
Define a SQL view `Q5(unsw_id, student_name)`:
Give the UNSW id and name of all the distinct international students who has only ever scored HD in their courses (refers to the `course_enrolments.grade`).

- `unsw_id` should be taken from `People.unswid` field;
- `student_name` should be taken from `People.name` field.

Question 6 (4 marks)
Define a SQL view `Q6(school_name, stream_count)`:
Give the schools that have offer more streams than the `School of Chemical Engineering`. Only consider the streams that are directly offered by an organization unit that is a `school`.

- `school_name` should be taken from `Orgunits.longname` field;
- Return `stream_count` as integer.

Question 7 (4 marks)
Define a SQL view `Q7(course_id, staff_name)` that gives the `id` of the courses whose classes are held at more than three buildings in semester 2010 S2, and the name of all the `course_staff` of these courses. The results should not contain the courses with no course_staff. Only consider the courses directly offered by `School of Computer Science and Engineering`. If a course has more than one course staff, return all of them. The view should return the following details:

- `course_id` should be taken from `Courses.id` field;
- `staff_name` should be taken from `People.name` field.

Question 8 (5 marks)
Define SQL view `Q8(unsw_id, name)` that gives all the distinct `MSc` students who are eligible to graduate with distinction in semester 2012 S2. A valid student should satisfy the following conditions in one program:

- enroll a program in `MSc` (refer to `program_degrees.abbrev`);
- must pass at least one course in the program in semester `2012 S2`.
- **average mark** >= 80. **Average mark** means the average mark of all courses a student has passed before 2013 (exclusive) in the program.
- the total UOC (refer to `subjects.uoc`) earned in the program before 2013 (exclusive) should be no less than the required UOC of the program (refer to `programs.uoc`). A student can only earn the UOC of the courses he/she passed.

The view should return the following details about each student:

- `unsw_id` should be taken from `People.unswid` field;
- `name` should be taken from `People.name` field.

Note:

- to pass a course, a student must get at least 50 in that course `(Course_enrolments.mark >= 50).`

- if a student has enrolled into several different programs, you need to calculate the UOC and average mark separately according to different programs. A course belongs to a program if this student enrolls into course and program in a same semester (refer to `Semesters.id`).

Question 9 (5 marks)
Below are the rules for students' academic standings:

i. If a student takes more than one course in a semester, his/her academic standing would be 'Probation' if none of them passed, 'Referral' if 50% or less of the taken courses are passed, and 'Good' otherwise.
ii. If a student takes only one course in a semester, he/she will receive 'Good' if he/she passes it, and 'Referral' if he/she fails it.

Define SQL view `Q9(unsw_id, name, academic_standing)`, which gives the `unswid`, `name` and `academic standing` of students with the `unswid` beginning with '313' in 2012 S1. The view should return the following details about each student:

- `Unsw_id` should be taken from `People.unswid` field;
- `name` should be taken from `People.name` field;
- `academic standing` should be defined as the rules given above. You will need to display 'Good', 'Probation', 'Referral' for each student you found.

Note:

- for each student, we only consider the courses in which he/she receives a not `null` mark. You may use `Course_enrolments.mark` >= 0 to retrieve a list of valid students;
- to pass a course, a student needs to get 50 or more in that course. (`Course_enrolments.mark` >= 50).
- don't consider the students who never get any mark from any course in that semester.

Question 10 (5 marks)
Define a PL/pgSQL function `Q10(staff_id integer)` that takes a staff id (`People.id`) and returns a list of all roles the staff had. Use table `affiliations` to find the information. The output should include the orgunit names, his/her role names in corresponding orgunit, and the starting dates of the roles. Output should appear ordered chronologically by the starting date, in case of the same starting date, ordered alphabetically by the role names, then the orgunit name. See check file for examples. We will only test with valid inputs.

- `staff_id` is taken from `People.id` field;

Each line of the output (in `text` type) should contain the following three elements and they are concatenated with a slash '/':

- The orgunit name should be taken from `Orgunits.longname` field;
- A role name should be taken from `Staff_roles.name` field;
- Starting date should be taken from `Affiliations.starting` field.

Question 11 (5 marks)

Define a PL/pgSQL function `Q11(year courseyeartype, term character(2), orgunit_id integer)` that takes a year, a term and an orgunit id. Output the pass rate of each course offered by the given orgunit in the given term. A course is offered by an orgunit if its related subject is offered by the orgunit.

- `year` is taken from `Semesters.year` field;
- `term` is taken from `Semesters.term` field;
- `orgunit_id` is taken from `Orgunits.id` field

Each line of the output (in `text` type) should contain the following two elements and they are concatenated with a space:

- a subject code (related to the course) should be taken from `Subjects.code` field;
- the pass rate (it is better to process under `numeric` type) of the related course.

A student passes a course if he/she obtains a grade in {SY, PC, PS, CR, DN, HD} for this course. The pass rate of a course = number of students who pass this course ÷ number of students enrolled in this course. Round the pass rate to the nearest 0.0001. Use the same rule as Question 4. If no students are enrolled in the course, i.e., the divisor is zero, ignore this course in the output. See check file for examples. We will only test with valid inputs.

Question 12 (5 marks)

Define a PL/pgSQL function `Q12(code character(8))` that takes a subject code and returns the set of all same-prefix subjects that include this subject (the input code) in their pre-reqs. You only need to consider literal subject codes (e.g., COMP9020) in the pre-reqs. Same-prefix subjects are the ones share the same prefix codes. E.g., COMP9020 and COMP9021. But COMP9020 and MATH1000 are not. See check file for examples. We will only test with valid inputs.

- `code` is taken from `Subjects.code` field;

Each line of output (in `text` type) should contain an element:

- A subject code should be taken from `Subjects.code` field which has the `input code` in its pre-reqs info (`Subjects._prereq`);

7. Submission

You can submit this project by doing the following:

- Students must submit an electronic copy of their answers to the above questions to the course website in Moodle.
- The file name should be proj1_**studentID**.sql (e.g., **proj1_z5100000.sql**).
- If you submit your project more than once, the last submission will replace the previous one
- In case that the system is not working properly, you must take the following actions:
- Please keep a copy of your submitted file on the CSE server. If you are not sure how, please have a look at taggi.

The proj1.sql file should contain answers to all 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** from the database that you're using for testing
- A new check.sql file may 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

8. Check your Answers

Before you submit your solution, you should check that it loads correctly for testing by using something like the following operations. For function questions, we provide five testcases for each question (E.g., for question 10, they are q10a to q10e). Testcases can be found from line220 in check.sql:

```
$ dropdb proj1              ... remove any existing DB
$ createdb proj1             ... create an empty database
$ psql proj1 -f /home/cs9311/web/22T2/proj/proj1/mymyunsw.dump      ... load the
MyMyUNSW schema and data
$ psql proj1 -f /home/cs9311/web/22T2/proj/proj1/check.sql  ... load the checking code
$ psql proj1 -f proj1.sql       ... load your solution
$ psql proj1
proj1=# select check_q1();          … check your solution to question1
…
proj1=# select check_q6();          … check your solution to question6
…
proj1=# select check_q12a();        … check your solution to question12 testcase (a)
proj1=# select check_all();         … check all your solutions
```

Notes:

1. You must ensure that your proj1.sql file will load and runs correctly (i.e., it has no syntax errors, and it contains all your view definitions in the correct order).
    a. If your database contains any views that are not available in a file somewhere, you should put them into a file before you drop the database.
    b. If we 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.
    c. If your code loads with errors, fix it and repeat the above until it does not.

2. In addition, write queries that are reasonably efficient.
    a. For each question, you must output result within 120 seconds on Grieg server. This time restriction applies to the execution of the ‘select * from check_Qn()’ calls.
    b. For each question, you will be fined via half of the mark penalty if your solution cannot output results within 120 seconds.

9. Late Submission Penalty

5% reduction (-2.5 out of 50) for each 24 hours after the deadline date and time. Submissions that are more than five days late will not be marked.