Project 1 SQL
COMP9311 22T1

The deadline for project 1 is: April 1st 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.
- 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
- PLpgSQL functions are **not** allowed to use in this project
- For each question, you must output result within 120 seconds on Grieg server.

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/22T1/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/22T1/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/22T1/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 as defined in each problem (see details from the solution template we provided).

Question 1 (2 marks)

Define a SQL view `Q1(subject_name)`:
Give the names of subjects that mentions at least two COMP courses in its prerequisite description, where one of COMP course mentioned must be a level 3 course (i.e., with a course code of the format COMP3***).

- `subject_name` should be taken from `Subjects.name` field;

Question 2 (3 marks)

Define an SQL view `Q2(course_id)` that gives the id of the course that holds `Studio` in at least 3 different buildings.

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

Question 3 (3 marks)

Define a SQL view `Q3(course_id, use_rate)`:
Give the courses that used the `Central Lecture Block` the most in 2008.

- `course_id` should be taken from `Courses.id` field;
- Return `use_rate` as integer;
- For each course, the value `use_rate` is number of its classes using a room in `Central Lecture Block` building (refers to `buildings.name`).

Question 4 (3 marks)

Define a SQL view `Q4(facility)`:
Give a complete list of the facilities that do not exist on any of the buildings on UNSW Kensington Campus with a grid reference that begins with the letter ‘C’.

- `facility` should be taken from `Facilities.description` field.
- Each building on the Kensington campus has a grid reference number to denote its relative position on the map (e.g., CSE building has a grid reference of K17 beginning with the letter ‘K’). This information exists in the column `Buildings.gridref`.

Question 5 (4 marks)

Define a SQL view `Q5(unsw_id, student_name)`:
Give the UNSW id and name of all the distinct domestic 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(subject_name, non_null_mark_count, null_mark_count):`
Give the subject name of the course, the number of student marks that are not null, and the number of students marks that are null. We only consider courses offered in Semester 1 2006 that recorded more than 10 students with null marks.

- `subject_name` should be taken from `Subjects.name` field;
- Return `non_null_mark_count` and `null_mark_count` as integer.

Question 7 (4 marks)
Define a SQL view `Q7(school_name, stream_count):`
Give the schools that have offer more streams than the `School of Computer Science and 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 8 (4 marks)
The university is trying to pair students together for a special project.
Define a SQL view `Q8(student_name_local,student_name_intl):`
Give all possible combinations of student pairings. The students concerned are all from in course 2012 Semester 1, `Engineering Design` who scored higher than 98 marks. To encourage some semblance of international exchange, we want the pair to consist of one domestic student and one international student, and they both should have the same WAM. (We want each pair to be formatted as such <local>, <international> exactly)

- `student_name_local` and `student_name_intl` should be taken from `People.name` field;
- `Engineering Design` refers to `Subjects.name`.

Question 9 (4 marks)
Define a SQL view `Q9(ranking, course_id, subject_name, student_diversity_score):`
Give the ranking, course id, the corresponding subject name, and score of the courses with the highest diversity scores. We consider a student diversity metric; the value of which is determined by the number of known distinct countries of origins of all its student members (To help with comparisons, the university always assumes that there is at least one student with the origin country ‘Australia'). We also only consider scores with a Rank of 6 or above.

- `course_id` should be taken from `Courses.id` field;
- `subject_name` should be taken from `Subjects.name` field;
- Return `student_diversity_score` as integer.
- There may be courses with the same diversity score, your result should assign the same rank to courses with the same score. The Rank() function in PostgreSQL will be able to do this for you for the ranking column.

Question 10 (4 marks)
Define a SQL view `Q10(subject_code, avg_mark)`:
Give the subject codes (of the related courses) and average mark of all PG-career courses offered by `School of Computer Science and Engineering` in Semester 1 2010.

- `subject_code` should be taken from `Subjects.code` field;
- Assume that if a student does not have a mark, they have a 0 mark instead.
- Only consider the courses having more than 10 students.
- Use `AVG()` function when calculating the average marks, save the results as `numeric(4,2)`

Question 11 (5 marks)
Give the subjects in 2008, where the average mark received by its students increased the most from semester 1 to semester 2. Define a SQL view `Q11(subject_code, inc_rate)`: Give the subject code and the increase rate of its average mark.

- `subject_code` should be taken from `Subjects.code` field;
- Only consider the subjects offered by `School of Chemistry` or `School of Accounting`.
- Only consider non-null marks.
- Increase rate in average mark = (avg_mark of S2 - avg_mark of S1) ÷ avg_mark of S1.
- Round the increase rate to the nearest 0.0001. (e.g., if inc_rate = 0.01, then return 0.0100; if inc_rate = 0.01234, then return 0.0123; if inc_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.

Question 12 (5 marks)
We want to find the total lab class time per week of COMP courses taught by the lecturer(s) who has a title of Dr. Define a view `Q12 (name, subject_code, year, term, lab_time_per_week)`: Gives lecturer's name, the subject code of the course, the year and term of the course provided, and the total lab class time (hours) per week of the course. Note:

- `name` should be taken from `People.name` field;
- `subject_code` should be taken from `Subjects.code` field;
- `year` and `term` should be taken from `Semesters.year` and `Semesters.term` fields, respectively;
- Return `lab_time_per_week` as integer; `lab_time_per_week` is the sum of all lab class time per week for a course;
- A lecturer refers to his/her `Staff_roles.name` contains "`Lecturer`";
- A lab class refers to its `Class_types.unswid` is `LAB`.

Question 13 (5 marks)
Define a view `Q13(subject_code, year, term, fail_rate)` that gives the fail rate for the semester with the largest number of enrolled students for each COMP course, only considering courses with more than 150 students enrolled. Note:

- `subject_code` should be taken from `Subjects.code` field.
- `year` should be taken from `Semesters.year` field.
- `term` should be taken from `Semesters.term` field.
- `fail rate` = number of students with mark less than 50 ÷ number of students with mark
- When counting the number of enrolled students, consider those with null mark; But when calculating fail rate, do not consider null mark records.
- Round `fail_rate` to the nearest 0.0001. (e.g., if fail_rate = 0.01, then return 0.0100; if fail_rate = 0.01234, then return 0.0123; if fail_rate = 0.02345, then return 0.0235).

- COMP course refers to the course whose subject code starts with “COMP”

To explain, the following table gives the information of each PHYS course whose number of enrolled students is more than 150. The red lines represent the semester with the largest number of enrolled students for each PHYS course (also # of students > 150).

```
 code        |  year  |term| stu_count
-------------+--------+-----+------------
 PHYS1121 |  2010 |  S2  |    167
 PHYS1231 |  2012 |  S2  |    169
 PHYS1231 |  2010 |  S2  |    183
 PHYS1121 |  2012 |  S2  |    186
 PHYS1231 |  2011 |  S2  |    188
 PHYS1160 |  2012 |  S2  |    196
 PHYS1131 |  2009 |  S1  |    201
 PHYS1160 |  2012 |  S1  |    213
 PHYS1160 |  2011 |  S2  |    217
 PHYS1121 |  2011 |  S2  |    238
 PHYS1121 |  2008 |  S1  |    239
 PHYS1131 |  2012 |  S1  |    258
 PHYS1131 |  2010 |  S1  |    318
 PHYS1131 |  2011 |  S1  |    344
 PHYS1121 |  2009 |  S1  |    380
 PHYS1121 |  2010 |  S1  |    453
 PHYS1121 |  2012 |  S1  |    474
 PHYS1121 |  2011 |  S1  |    524
```

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:

```
$ dropdb proj1              ... remove any existing DB
$ createdb proj1            ... create an empty database
$ psql proj1 -f /home/cs9311/web/22T1/proj/proj1/mymyunsw.dump      ... load the
MyMyUNSW schema and data
$ psql proj1 -f /home/cs9311/web/22T1/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_q12();       ... check your solution to question12
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

20% reduction (-10 out of 50) for each 24 hours after the deadline date and time.