Project 1 SQL
COMP9311 23T2

The deadline for project 1 is: July 14th 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 Nw-syd-vxdb 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 Nw-syd-vxdb server, simply run the following two commands:

```
$ createdb proj1
$ psql proj1 -f /home/cs9311/web/23T2/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 Nw-syd-vxdb, assuming that Nw-syd-vxdb is not under heavy load. (If you leave your project until the last minute, loading the database on Nw-syd-vxdb 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 Nw-syd-vxdb or if you have large files under your /srvr/YOU/ directory, it is possible that you will exhaust your Nw-syd-vxdb disk quota. Regardless, it is certain that you will not be able to store two copies of the MyMyUNSW database under your Nw-syd-vxdb 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/23T2/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/23T2/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)` that gives all the subjects that are offered by organizations whose type is centre .

- `subject_code` should be taken from `Subjects.code` field;
- Centre refers to the `Orgunit_types.name` field that contains ‘`Centre`’.

Question 2 (3 marks)

Define an SQL view `Q2(course_id)` that gives the id of the course that has at least 4 different types of classes, where one of them should be seminar.

- `course_id` should be taken from `Courses.id` field;
- Seminar refers to the `Class_types.name` field that contains ‘`Seminar`’.

Question 3 (3 marks)

Define a SQL view `Q3(unsw_id)` that gives the unswid of students who enrolled in at least two courses in year 2010. Only consider the course that is equivalent to at least one JURD course and at least one LAWS course at the same time (i.e., with a course code of the format JURD**** and LAWS****).

- equivalent refers to `subject._equivalent` field;
- `unsw_id` should be taken from `people.unswid` field.

Question 4 (4 marks)

Define an SQL view `Q4(course_id, avg_mark)` that gives the course who has the maximum average mark among all the COMP courses in year `2010`. When calculating the average mark, we only count the students with valid marks (not null). Round avg_mark to the nearest 0.0001. (i.e., if avg_mark = 90.01 , then return 90.0100; if avg_mark = 85.01234, then return 85.0123; if avg_mark = 72.02345, then return 72.0235). 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. If there are multiple courses that have the maximum average mark, return all of them.

- COMP courses refer to the courses whose related `Subjects.code` contains '`COMP`'.
- `course_id` should be taken from `Courses.id`;
- `avg_mark` should be in `numeric` type.

Question 5 (4 marks)
Define a SQL view `Q5(faculty_id, room_id)` that gives the faculties and rooms that held the maximum number of tutorials offered by the corresponding faculties among all the rooms in the year `2005`. A tutorial is offered by a faculty if it belongs to a course whose related subject is offered by this faculty. If there are multiple rooms that held the maximum number of tutorials, return all of them.

- Faculty refers to an organization where its `Orgunit_types.name` contains `Faculty`';
- Tutorial refers to the class where `class_types.name` is '`Tutorial`';
- `Faculty_id` should be taken from `Orgunits.id` field;
- `room_id` should be taken from `Rooms.id` field.

Question 6 (4 marks)
Define a SQL view `Q6(program_id, stream_id)` that gives the programs offered by `Faculty of Arts and Social Sciences` in semester `2005 S1` and the streams with the highest number of enrolments in the corresponding program among all the streams that can be enrolled as part of this program in semester `2005 S1`.

- `program_id` should be taken from `Programs.id` field;
- `stream_id` should be taken from `Streams.id` field.

Question 7 (4 marks)
Define a SQL view `Q7(subject_id, staff_name)` that gives the subjects who are offered by Law organizations and provided at least two courses in year `2008`, and the corresponding course staffs who were associated with the teaching for all the courses related to those subjects in that year. The results should not contain the subjects with no course staff. If a subject associates with more than one course staff, return all of them. The view should return the following details:

- Law organization refers to an organization where its `Orgunits.name` contains '`Law`' (regardless of case).
- `subject_id` should be taken from `Subjects.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 students who enrolled in the programs offered by `Faculty of Science` and have ranked top 10 only in math-related courses. Math-related courses refer to the courses whose related subject code starts with '`MATH`'. We only consider the courses that have been enrolled by over than 100 students. 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:

- If a student has enrolled into several different programs, you only need to consider the course included by the program offered by `Faculty of Science`. A course is included in a program if this student enrolls into the course and the program in the same semester (refer to `Semesters.id`);
- To identify top 10 students in a course, the ranking is based on the marks received by the students(refer to `Course_enrolments.mark`) in this course, from highest to lowest. If multiple students have achieved the same mark, they should be assigned with the same ranking. The Rank() function in PostgreSQL will be able to do this for you to generate the ranking column.

Question 9 (5 marks)

Define SQL view `Q9(prof_id, fail_rate)` that gives all the professors from `School of Mechanical and Manufacturing Engineering` and the fail rate of all the `UG` courses where they are employed as course convenor. We only count the students with valid marks (not null).

- Failing a course means that `Course_enrolments.mark` is lower than 50.
- A person is identified as a professor if his/her `staff_roles.name` contains '`Professor`' (regardless of case) in an affiliation.
- A person is employed as a course convenor for a course if his/hers `staff_roles.name` is '`Course Convenor`' in course_staff.
- `UG` refers to `Subjects.career`.
- If a professor does not teach any `UG` course, then do not include he/she in the results.
- Round the fail rate to the nearest 0.0001. Use the same rule as Question 4.

The view should return the following details:

- `prof_id` should be taken from `People.unswid` field;
- `fail_rate` should be in numeric type.

Question 10 (5 marks)

Define a SQL view `Q10(student_id, program_id, remain_uoc)` that gives the `MA` students who may fail to graduate, the corresponding program and the remaining UOC they need to graduate. A student who fails to graduate should satisfy the following conditions in one program:

- Enroll a program in `MA` (refer to `Program_degrees.abbrev`);
- The total UOC (refer to `Subjects.uoc`) earned in the program (exclusive) is less than half of the required UOC of the program. A student can only earn the UOC of the courses he/she pass, i.e., the mark for the course (refers to `Course_enrolments.mark`) should be no less than 50.
- The duration he/she enrolled into this program exceeds 2000 days. For each student, the duration is the number of days between the earliest date(refer to `Semesters.starting`) and the latest date(refer to `Semesters.ending`) among all his/her enrolments for the same program.

The view should return the following details about each student:

- `Student_id` should be taken from `People.unswid` field;

- `Program_id` should be taken from `Programs.id` field;
- `remain_uoc` should be an integer.

Note:

- Remaining UOC = required UOC - earned UOC.
- If a student has enrolled into several different programs, you need to calculate the UOC separately according to different programs. A course is included in a program if this student enrolled into the course and the program in the same semester (refer to `Semesters.id`).

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, and returns the grade distribution ratio for all the students enrolled in the programs offered by the given organization.

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

For each student, the grade is based on the average mark of the courses enrolled in the given semester. The grading rules are as follows:

i. HD if 85 <= semester average mark.
ii. DN if 75 <= semester average mark < 85.
iii. CR if 65 <= semester average mark < 75.
iv. PS if 50 <= semester average mark < 65.
v. FL otherwise.

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

- The grade which is represented by 2 characters.
- The corresponding rate (which should be in `numeric` type).

The rate = number of students who received a particular grade ÷ total number of students enrolled in programs offered by the given organization. Round the rate to the nearest 0.0001. Use the same rule as Question 4. If a student did not receive any mark in the given semester, ignore this student during the calculation. See check file for examples. We will only test with valid inputs.

Question 12 (5 marks)

Define a PL/pgSQL function `Q12(subject_prefix character(4))` that takes the prefix of subject code, and returns the associated courses where the course staff is from at least 4 different organizations, and the corresponding organizations. See check file for examples. We will only test with valid inputs.

- Associated courses refer to the courses where the code of the related subjects starts with `subject_prefix`.

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

- A course id should be taken from `Courses.id` field;

- A list of id taken from `Orgunits.id` concatenated with slash '/'. Note that the id should be in ascending order (lowest to highest).

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 11, they are q11a to q11e). Testcases can be found from line228 in check.sql:

$ **dropdb proj1** ... remove any existing DB
$ **createdb proj1** ... create an empty database
$ **psql proj1 -f /home/cs9311/web/23T2/proj/proj1/mymyunsw.dump** ... load the MyMyUNSW schema and data
$ **psql proj1 -f /home/cs9311/web/23T2/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 Nw-syd-vxdb 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 of the max assessment mark (-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.