Project 1 SQL

The deadline for project 1 is:

Thu 1 Apr, 5:00 pm (Sydney time) for Q1-Q12

Fri 9 Apr, 5:00 pm (Sydney time) for Q13

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 PLpgSQL functions to aid in satisfying requests for information

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.

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 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 available 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 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 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.

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/21T1/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 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
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 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.

If you are running PostgreSQL at home, you can download the files: mymyunsw.dump, proj1.sql to get you started. You can grab the check.sql separately, once it becomes available.

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 proj1**
... PostgreSQL welcome stuff ...
proj1=# **\d**
... look at the schema ...
proj1=# **select * from Students;**
... look at 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;**
... how many course enrolments records ...
proj1=# **select * from dbpop();**
... how many records in all tables ...
proj1=# ... etc. etc. etc.
proj1=# **\q**

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

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/21T1/proj/proj1/mymyunsw.dump**
$ **psql proj1**
... run some checks to make sure the database is ok
$ **mkdir Project*1Directory***
... make a working directory for Project 1
$ **cp /home/cs9311/web/21T1/proj/proj1/proj1.sql Project*1Directory***

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.

Notes

Read these before you start on the exercises:

- the marks reflect the relative difficulty/length of each question
- use the supplied **proj1.sql** template file for your work
- you may define as many additional functions and 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
- make sure that your queries would work on any instance of the MyMyUNSW schema; don't customize them to work just on this database; we may test on a different 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
- 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
- You can define SQL views to answer the following questions.
- 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, take a look at the expected_qX tables supplied in the checking script.

5. 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).

Q1 (3 marks)

Define an SQL view `Q1(subject_longname, subject_num)` that gives the `longname` (refers to the `subjects.longname`) of the PhD related subject and the number of subjects having the same subject longname.

Note:

- Only consider the `subject_longname` that contains 'PhD' (not 'Ph.D').
- Only consider the `subject_longname` with more than one subjects (`subject_num>1`).

Q2 (3 marks)

Define an SQL view `Q2(OrgUnit_id, OrgUnit, OrgUnit_type, Program)` that gives the details of the OrgUnit which offers the programs having more than 300 total units of credit (refers to the `Programs.UOC`). The view should return the following details about each course:

- `OrgUnit_id` should be taken from `OrgUnits.id`
- `OrgUnit` should be taken from `OrgUnits.name`
- `OrgUnit_type` should be taken from `OrgUnit_types.name`
- `Program` should be taken from `Programs.name`

Q3 (3 marks)

Define an SQL view `Q3(course, student_num, avg_mark)` that gives the details of the courses which are only taught by 'Course Tutor' and have the average marks more than 70. The view should return the following details about each course:

- `course` should be taken from `courses.id`.
- `student_num` should be the number of students for the course.
- `avg_mark` should be computed from `Course_enrolments.mark`.

Note:

- When calculating the average, onl y consider students who have a **not null** mark.
- The mark of a course can be referred from `Course_enrolments.mark`.
- Only consider valid courses which have ~~at least~~ more than **10** students.
- The courses should be only taught by 'Course Tutor' rather than other types of staff.
- return `avg_mark` as `numeric(4,2)`.

Q4 (3 marks)

Define an SQL view `Q4(student_num)` that gives the number of students who enrolled the courses that having the `Lecture` classes held in both `Quadrangle` and `Red Centre` buildings.

Note:

- The buildings are considered based on the `Buildings.name` filed.
- The classes are considered based on the `Class_types.name` filed.

Q5 (3 marks)

Define an SQL view `Q5(staff, min_mark, course)` that gives all the `distinct staffs` who have taught the courses offered by the `School of Law` (refers to the `OrgUnits.Name`). We also want to know the minimum mark of the courses he taught and the corresponding course. The view should return the following details about each staff:

- `staff` should be taken from `staff.id` field.
- `min_mark` should be considered on the `Course_enrolments.mark` field.
- `course` should be considered on the `Courses.id` field.

Note:

- If a staff taught many courses with the same minimum mark, we return all these courses.
- When computing the marks, only consider students who have a **not null** mark.
- The results should be ordered by the descending `min_mark`.

Q6 (3 marks)

Define an SQL view `Q6(course_id)` that gives the courses, where less than 2/5 of the enrolled students in the course have higher marks than the average mark of the courses. For example, there are 50 students enrolled in course A, and the average mark of thecourse A is 60. If there are 17 students who get marks higher than 60, course A will be returned in the view Q6. The view should return the following details about each course:

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

Note:

- Only consider the courses are enrolled by more than **10** students (who have **not null** marks) of the years 2009 and 2010 (refers to the `Semesters.year`).
- When calculating the average, only consider students who have **not null** markss.
- The mark of a course can be referred from `Course_enrolments.mark`

Q7 (3 marks)

Define an SQL view `Q7(staff_name, semester, course_num)` that displays the staff with the highest `course_num` in each semester between 2005 and 2007 (refers to the `Semesters.year`), and `course_num` is the number of the courses a staff taught. The view should return the following details about each course:

- `Staff_name` should be taken from `People.name` field.
- `semester` should be taken from `Semesters.longname` field.

Note:

- Only consider valid courses with at least **20** students.
- If several staffs have the same highest course number, return all of them.
- Skip the semester with no valid course.

Q8 (3 marks)

Define an SQL view `Q8(role_id, role_name, num_students)` that gives the numbers of distinct students taught by each role (refers to the `Course_Staff.role`) of staff from `School of Computer Science and Engineering` (refers to the `Orgunits.longname`) in 2010 (refers to the `Semesters.year`). The view should return the following details about each course:

- `role_id` should be taken from `Staff_roles.id` field.
- `role_name` should be taken from `Staff_roles.name` field.

Note:

- Only consider distinct students taught by each role of staff. For example, if a student enrolled 5 courses which are taught by Course Lecturer, we only count this student once for Course Lecturer.
- The affiliation of each staff can refer to the table `Affiliations`.

Q9 (4 marks)
Data Management course admins would like to know the average mark of `intl` and `local` students on each semester separately. Define an SQL view `Q9(year, term, stype, average_mark)` to help them monitor the average mark each semester. Each tuple in the view should contain the following:

- the year (`Semesters.year`) of the semester
- the term (`Semesters.term`)
- the type (`Students.stype`)
- the average mark of students enrolled in this course this semester as `numeric(4,2)`

Database Systems has value 'Data Management' in the `Subjects.name` field. You can find the information about all the course offerings for a given subject from `Courses`. You should calculate the average mark of enrolled students for a course offering from the table `Course_enrolments`.
Note:

- When calculating the average marks, only consider **not null** mark records.
- Only consider the semesters which have 'Data Management'.

Q10 (4 marks)
Define SQL view `Q10(room, capacity, num)`, which gives, for each `capacity`, the `room` with the greatest number of `distinct` facilities. The view should return the following details about each room:

- `room` should be taken from `Rooms.longname` field.
- `capacity` should be taken from `Rooms.capacity` field.
- `num` counts the total number of distinct facilities in that room.

Note:

- Ignore the rooms with no facility or no capacity.
- Ignore the rooms with capacity less than 100 (< 100).
- In the case of ties, the different rooms of same capacity with same number of facilities should all be included in the result.

Q11 (4 marks)
Define SQL view `Q11(staff, subject, num)`, which gives the **maximum** enrolment `number` of the courses in each `subject` a `staff` taught. We only consider the subjects a same staff has taught for at least **three** consecutive years (e.g., 2000, 2001, 2002). The view should return the following details:

- `staff` should be taken from `People.name` field.
- `subject` should be taken from `Subjects.longname` field.
- `num` counts the maximum enrolment number of all valid courses in this subject taught by this staff.

Note:

- Only consider the enrolments with not null mark. i.e., `Course_enrolments.mark` >= 0.
- Only consider major semesters (i.e., S1 and S2).
- Only consider staff with `staff.id` with 50354 (i.e., between 5035400 and 5035499 inclusive).
- Ignore the courses with no valid enrolment and the subjects with no valid course.

Q12 (4 marks)

Define SQL view `Q12(staff, role, hd_rate)`, which gives the overall course high distinction rate of the target staff for each specific role. A valid staff should satisfy the following conditions:

- he/she has taught at least **two** different subjects, in each of which he/she had exactly **three** different staff roles.
- he/she can teach some other subjects, in which he/she did not have three staff roles.

The view should return the following details about each staff:

- `staff` should be taken from `People.name` field.
- `role` should be taken from `Staff_roles.name` field.
- `hd_rate` as `numeric (4,2)`.

Note:

- Only consider enrolments with **not null** marks.
- To get a high distinction in a course, a student needs to get at least **85** in that course. i.e., `Course_enrolments.mark` >= 85.
- To calculate the overall course pass rate, you need to find all enrolments of the courses with each staff being each staff role. Then you can divide the total number of the passed students by the total number to valid enrolments.

Q13 (10 marks)

(You may use any combination of views, SQL functions and PLpgSQL functions in this question. However, you must define at least a PLpgSQL function called Q13.)

UNSW staff members hold different roles at different periods of time. This information is recorded in `Affiliations` table. Given the id of an organization, we want to find all the staff members who have had at least two non-concurrent roles in the given organization (i.e., they must have two roles where role1's ending date ⩽ role 2's starting date). Note that you need to consider the given organization may have lots of sub-organizations. For example, the faculty of engineering has many schools, such as biomedical engineering and CSE.

Give the id of an organization, write a PLpgSQL function to (a) find all the staff members who have had several non-concurrent roles in the given organization over time and (b) produce a list showing, for each staff member, their unswid, name, and the roles they've had in the given organization (includes the name of staff role, the name of organization, and starting date and ending date). Note that the roles should be ordered by their starting date and bedisplayed as a single string with the details for each role on a separate line (terminated by '\n').

Use the following type definition and function header:
create type EmploymentRecord as (unswid integer, staff_name text, roles text);
create or replace function Q13(integer) returns setof EmploymentRecord ...

Note that PostgreSQL uses a + character to indicate an end-of-line in its output (as well as printing '\n'). You may need to use array for this question, and its documentation can be found at: here.

Sample results (details can be found in check.sql):

proj2=#select * from q13(661);

```
proj1=# select * from q13(661) ;
 unswid  |   staff_name   |                                roles
---------+----------------+-----------------------------------------------------------------
 3140956 | Thomas Loveday | Senior Lecturer, Interior Architecture Program (2011-11-25..)   +
         |                | Senior Lecturer, Architecture Program (2011-09-05..2011-09-05)
 9226425 | Stephen Ward   | Lecturer, Industrial Design Program (2011-09-27..)              +
         |                | Program Head, Industrial Design Program (2001-01-01..2011-09-27)
(2 rows)
```

6. 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 names should be proj1.sql and proj2.sql.
- If you submit your project more than once, the latest submission will replace the previous one
- In case that the system is not working properly, you must take the following action:
    - Please keep a copy of your submitted file on the CSE account. If you are not sure how, please have a look at taggi.

The proj1.sql and proj2.sql files should contain answers to all of 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

Before you submit your solution, you should check that it will load correctly for testing by using something like the following operations (same for proj2):

$ **dropdb proj1** ... remove any existing DB
$ **createdb proj1** ... create an empty database
$ **psql proj1 -f /home/cs9311/web/21T1/proj/proj1/mymyunsw.dump** ... load the MyMyUNSW schema and data
$ **psql proj1 -f /home/cs9311/web/21T1/proj/proj1/check.sql** ... load the checking code
$ **psql proj1 -f proj1.sql** ... load your solution (replace by proj2.sql here)
$ **psql proj1**
proj1=# **select check_q1();** … check your solution to question1 (replace by check_q13() here)
…
proj1=# **select check_q6();** … check your solution to question6
…
proj1=# **select check_q12();** … check your solution to question13
proj1=# **select check_all();** … check all your solutions
Note: 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.

If your code loads with errors, fix it and repeat the above until it does not.

You must ensure that your proj1.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 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. In addition, make sure that your queries are reasonably efficient. 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. For each question, you will be fined via half of the mark penalty if your solution cannot output results within 120 seconds.

7. Late Submission Penalty

10% reduction for the 1st day late, then 30% reduction per day.