Project 1 SQL

The deadline for project 1 is:

3rd April, 5:00 pm

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 give
- 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 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 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/20T1/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
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 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 enrolment 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/20T1/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/20T1/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.

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 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
- 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(courseid,code)` that gives the `distinct` course id and subject code of any course that is taught by `Course Tutor` (refers to the `staff_roles.name`). We only consider the courses with the subject code starting with 'LAWS', like 'LAWS1213'. The view should return the following details about each course:

- `courseid` should be taken from `Courses.id` field.
- `code` should be taken from `Subjects.code` field.

Q2 (3 marks)

Define an SQL view `Q2(unswid,name,class_num)` that gives the `distinct` building id, name and the number of lecture class (refers to the `class_type.name`) taken in this building. The view should return the following details about each course:

- `unswid` should be taken from `Buildings.unswid` field.
- `name` should be taken from `Buildings.name` field.

Note:

- only consider the buildings which have at least **one** lecture class.

Q3 (4 marks)

Define a SQL view `Q3 (classid,course,room)` that displays `classid`, `course` and `room` of all the `distinct` **classes** equipped with `Television monitor` (refers to the `facilities.description`). We only consider the classes of the courses enrolled by the student `Craig Conlon`. The view should return the following details about each class:

- `classid` should be taken from `Classes.id` field.
- `course` should be taken from `Classes.course` field.
- `room` should be taken from `Classes.room` field.

Q4 (4 marks)

Define a SQL view `Q4(unswid,name)` that gives all the `distinct local` students who have enrolled in `COMP9311` and `COMP9021,` and got `CR` (refers to the `Course_enrolments.grade`) in both courses. The view should return the following details about each student:

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

Q5 (4 marks)

Define a SQL view `Q5(num_student)` that gives the number of `distinct` students whose average mark is higher than the average of all the students' average marks. For example, if student A, B and C get 70, 80 and 90 respectively on average, the overall average mark is (70 + 80 + 90)/3 = 80 and we only count the student C who get an average mark higher than 80.
Note:

- when calculating the average, only consider students who have at least one **not null** mark.
- The mark of a course can be referred from `Course_enrolments.mark.`

Q6 (4 marks)
Define a SQL view `Q6(semester, max_num_student)` that displays the semester(s) with the lowest `max_num_student`, which is the number of the students of the course having the maximum enrolled students in that semester. The view should return the following details about each course:

- `semester` should be taken from `Semesters.longname` field.
- `max_num_student` should be the maximum number of students in the course of its corresponding semester.

Note:

- only consider valid courses which have at least **10** students.
- if several semesters have the same lowest maximum course student number, return all of them.
- skip the semester with no valid course.

Q7 (4 marks)
Define a SQL view `Q7(course, avgmark, semester)` that gives all the `distinct valid` courses with a high average mark (`avgmark` > 80) of the enrolled students in `each semester` of the years 2009 and 2010. The view should return the following details about each course:

- `course` should be taken from `Courses.id` field.
- `avgmark` should be calculated from `Course_enrolments.mark` field.
- `semester` should be taken from `Semesters.name` field.

Note:

- only consider valid courses which have at least **20 not null** mark records.
- the average mark of students enrolled in this course this semester as `numeric(4,2)`
- only consider the valid courses with an average mark more than 80.

Q8(4 marks)
Define SQL view `Q8(num)`, which gives the number of `distinct local` students enrolling in the year `2008` (refers to `Semesters.year`) in the `Law` stream (refers to `Streams.name`), but never enrolling in any course offered by `Accounting` **`or`** `Economics` (refers to `OrgUnits.name`).

Note:

- the student IDs are the UNSW ids (i.e. student numbers) defined in the `People.unswid` field.
- Do not count duplicate records.

Q9 (5 marks)
Define SQL view `Q9(unswid,name)`, which gives all the `distinct` students who had been enrolled in all the popular level-9 COMP subjects (i.e., subject code starting with 'COMP9') with

good performance. A subject was considered as popular if it was taught in every major semester (i.e., S1 and S2) from 2002 to 2013 (both inclusive). Good performance means that the student achieves either 'HD' or 'DN' in all the popular subjects (refers to the `Course_enrolments.grade`). A student will not be returned as a result if he/she got a poor grade on any popular subject. We only consider popular subjects in series 'COMP9%'. The view should return the following details about each student:

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

Note:

- it is possible that a student enrolled multiple times in a given subject. For example, one may fail the course in 03S1, and then be re-enrolled in the same course in 04S1. In this case we only consider the highest grade he/she got in that course.
- Do not count duplicate records.

Q10 (5 marks)

The university is interested in the `Lecture Theatre` usage status in 2010 S2. So please define SQL view `Q10(unswid,longname,num,rank)`, which gives the `distinct` room id and name of all the `Lecture Theatre` with the number of `distinct` classes that use this theatre and the rank. Theatre rankings are ordered by `num` from highest to lowest. If there are multiple theatres with the same `num`, they have the same rank. The view should return the following details about each theatre:

- `unswid` should be taken from `Rooms.unswid` field.
- `longname` should be taken from `Rooms.longname` field.
- `num` counts the total number of `distinct` classes that use this theatre.
- `rank` records the rank of the number of `distinct` classes that use this theatre.

Note:

- if there is no class using a theatre, the `num` would be 0.
- the ranking is with gaps. i.e., if there are 2 theatres ranked as first, the third theatre will be ranked as third.

Q11 (5 marks)

Define SQL view `Q11(unswid,name)` that gives all the `distinct` `BSc` students who are eligible to graduate with distinction in semester `2010 S2.` A valid student should satisfy the following conditions in one program:

- enroll a program in `BSc` (refer to `program_degrees.abbrev`)
- must pass at least one course in the program in semester `2010 S2.`
- **average mark** >= 80. **Average mark** means the average mark of all courses a student has passed before 2011(exclusive) in the program.
- the total UOC (refer to `subjects.uoc`) earned in the program before 2011 (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:

- `unswid` 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`).

Q12 (5 marks)

Below are the rules for postgraduate academic standings:

i. If a postgraduate student's cumulative (total number of) UOC (refer to `subjects.uoc`) failed during the entire postgraduate career is fewer than 12, his/her academic standing will be 'Good'.
ii. If a postgraduate student's cumulative (total number of) UOC failed during the entire postgraduate career is between 12(inclusive) and 18(inclusive), he/she will receive 'Probation'.
iii. If a postgraduate student's cumulative (total number of) UOC failed during the entire postgraduate career is more than 18, the academic standing will be 'Exclusion'.

Define SQL view `Q12(unswid,name,program,academic_standing)`, which gives the `unswid`, `name` and `academic standing` of `distinct` students enrolled in `MSc` (refer to `program_degrees.abbrev`). In this question we only consider students whose unswid starts with '329'. The view should return the following details about each student:

- `unswid` should be taken from `People.unswid` field.
- `name` should be taken from `People.name` field.
- `Program` should be taken from Programs`.name` field.
- `academic standing` should be defined as the rules given above. You will need to display 'Good', 'Probation', 'Exclusion' 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 and ignore students who do not get any mark in a program yet.
- The total number of failed UOC is calculated as the sum of the UOC of the failed courses. To fail a course, a student needs to get less than 50 in that course. (`Course_enrolments. mark` < 50)
- if a student has enrolled into several different programs, you need to calculate the UOC 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`).
- as these are the rules for postgraduate students, only consider the students who enrolled in `MSc` programs.

6. Submission

You can submit this project by doing the following:

- The file name should be proj1.sql.
- Log into the CSE server, ensure that you are in the directory containing the file to be submitted.
- Type "give cs9311 proj1 proj1.sql" to submit.
- You can also use the web give system to submit.
- 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 screen capture (including timestamp and the size of the submitted file) for your submissions as proof. If you are not sure how, please have a look at the guidelines.
- 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 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:

```
$ dropdb proj1              ... remove any existing DB
$ createdb proj1            ... create an empty database
$ psql proj1 -f /home/cs9311/web/20T1/proj/proj1/mymyunsw.dump     ... load the MyMyUNSW
schema and data
$ psql proj1 -f /home/cs9311/web/20T1/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 question5
…
proj1=# select check_q12();     … check your solution to question10
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, then 30% reduction per day.