COMP9311 24T1: Project 1

Deadline: Fri 16:59:59 March 29th (Sydney 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 in 'tasks' section 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 (following the 'AutoTest Checking' section).
- 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 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 students, 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/24T1/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 need 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/24T1/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/24T1/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).

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 level-7 subjects that are offered by organizations whose type is school. And the school's name should contain '`Information`'.

- `subject_code` should be taken from `Subjects.code` field.
- School refers to the `Orgunit_types.name` field that contains '`School`'.
- The school name refers to the `orgunits.longname` field.
- Level-7 refers to the `subject_code` formatted as 'XXXX7***', where 'X' represents a letter and '*' represents a number.

Question 2 (3 marks)

Define an SQL view `Q2(course_id)` that gives the ID of the COMP course only offering 'Lecture' and 'Laboratory' classes (the course has only 2 different types of classes).

- `course_id` refers to the `Courses.id` field.
- Lecture and Laboratory refer to the `Class_types.name` field.
- COMP courses refer to the courses whose related `Subjects.code` starting with '`COMP`'.

Question 3 (4 marks)

Define a SQL view `Q3(unsw_id)` that gives the unswid of students who enrolled in at least five courses in year between 2008 and 2012. Only consider the course that has at least two professors as staff and the student whose unswid starting with `320`.

- `unsw_id` should be taken from `people.unswid` field.
- Professor denotes the '`Prof`' in `People.title`.
- Year refers to `Semesters.year` field.

Question 4 (5 marks)

Define an SQL view `Q4(course_id, avg_mark)` that gives the ID of course along with the average mark of students who achieved 'above distinction' for each course. The view only includes the courses that have the highest average 'above distinction' mark when compared to other courses offered by the same faculty and within the same semester in the year 2012. If there are multiple courses sharing the same maximum average mark, list all the `course_id` and its `avg_mark` of these courses.

Note: Round `avg_mark` to the nearest 0.01 in in `numeric` type (i.e., 85.014≈85.01, 85.016≈85.02, 85≈85.00).

- `course_id` should be taken from `Courses.id`.
- Faculty refer to the organization units where their `Orgunit_types.name` is '`Faculty`'.
- The 'above distinction' means the course grade of a student is '`DN`' or '`HD`'.

Question 5 (5 marks)

Define a SQL view `Q5(course_id, staff_name)` that gives the ID of course which enrolled more than 500 students and had at least two professors as staff in year between 2005 and 2015. Show the **ordered** given name of the professors in the `staff_name` section.

Note: Concatenating the given name with '`; `' (i.e., `Jack; Michele`).

- Professor refers to the '`Prof`' in `People.title` field.
- The given name refers to the `People.given` field.

Question 6 (5 marks)

Define SQL view `Q6(room_id, subject_code)` that gives the ID of room(s) that were used most frequently by different classes in the year 2012, along with the subject code(s) that occupied it the most. If there are multiple rooms or subjects sharing the identical maximum usage, listing all of them.

- `room_id` should be taken from `rooms.id` field.
- `subject_code` should be taken from `subject.code` field.

Question 7 (5 marks)

Define SQL view `Q7(student_id, program_id)` that gives the IDs of students who have completed two or more programs offered by the same organizations. Additionally, the completion of **all programs** from the same organization occurred within a duration of 1000 days. The `program_id` column should list the IDs of these programs.

- `student_id` should be taken from `people.unswid` field.
- Organization refers to `orgunits.id` field.
- `program_id` refers to `programs.id` field.
- Assuming students can register at most 1 program in each semester.

Note:

- A student will pass the course and earn the UOC if she/he receives the mark ≥ `50`.
- The student is valid for graduation if the total UOC earned in the program (hint: `subjects.uoc`) is no less than the required UOC of the program (refer to `programs.uoc`).
- If a student has enrolled in several different programs, you need to calculate the UOC separately according to different programs. A course is considered part of a program if the student enrolls in both the course and the program during the same semester.

- For each student, the duration of one program is the number of days between the earliest date (hint: `Semesters.starting`) and the latest date (hint: `Semesters.ending`) among all her/his course enrolments for the same program.
- '1000 days' refers to the duration from the start of the first program (hint: `Semesters.starting`) to the completion of the latest program (hint: `Semesters.ending`) among all the programs the student completed from the same organization does not exceed 1000 days. The `program_id` field should enumerate the IDs of the programs involved.

Question 8 (6 marks)

Define SQL view `Q8(staff_id, sum_roles, hdn_rate)` that gives the id of the staff, the total number of roles (affiliations) the staff held in the past or currently across all organizations, and the total '`above distinction`' rate of all the courses where the staff member serves as `course convenor` in the year `2012`. The view displays only staff who previously held three or more roles (affiliations) in same organizations, showcasing the **top 20** results based on `hdn_rate`.

Note: Round the rate to the nearest 0.01 with the same rule as Question 4.

- Above distinction means that `Course_enrolments.mark` is ≥ 75.
- A person is employed as a course convenor for a course if her/his `staff_roles.name` is '`Course Convenor`' in course_staff.
- The number of staff roles (affiliations) in one organization refers to her/his `staff_roles.id` in `affiliations.orgunit` field.
- `staff_id` should be taken from `People.unswid` field.
- `hdr_rate` should be in numeric type.

Question 9 (6 marks)

Define a PL/pgSQL function `Q9(unswid integer)` that takes an unswid of a student and returns the subject code of the course the given student enrolled in and finished with a valid mark, and the rank of the student within that course. Only consider the courses which have at least one same-prefix course as prerequisite.

- An unswid should be taken from `People.unswid` field.
- Prerequisite refers to `_prereq` in the related table.
- Same-prefix courses are the courses which have the same first four characters.
- 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.

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

- Subject code: the subject code of the course should be taken from `subjects.code` field.

- Rank: the course rank should be an integer. If she/he ranked 2nd in the course, the result is 2.

Special output:

- If the student has not finished any required course with a valid mark, return a line in the format of '`WARNING: Invalid Student Input [X]`', where 'X' denotes the provided `unswid`.

Question 10 (8 marks)

Define a PL/pgSQL function `Q10(unswid integer)` that takes the unswid of a student. Output the students WAM for all the programs that the student enrolled.

- An unswid should be taken from People.unswid field.
- Unlike to the Question 7, in this question, a student passes a course if she/he obtains a grade in $\text{set}_{\text{pass}} = \{\text{SY}, \text{PT}, \text{PC}, \text{PS}, \text{CR}, \text{DN}, \text{HD}, \text{A}, \text{B}, \text{C}, \text{XE}, \text{T}, \text{PE}, \text{RC}, \text{RS}\}$ refers to `course_enrolments.grade`.
- All the grades in $\text{set}_{\text{pass}} \cap \text{set}_{\text{sy}} = \{\text{SY}, \text{XE}, \text{T}, \text{PE}\}$ or in $\text{set}_{\text{pass}} \cap (\text{course mark} = \text{null})$, means the courses the student passed but won't be included in WAM calculation.
- All the grade not in set_{pass}, excluding the course $\text{mark} = \text{null}$, means a fail. These failed courses are still included in the WAM calculation, while courses with null marks are excluded.
- If a student has enrolled in several different programs, you need to calculate the WAM separately according to different programs. A course is considered part of a program if the student enrolls in both the course and the program during the same semester. Assuming students can register at most 1 program in each semester.

WAM is calculated according to the following formula:

$$\text{WAM} = \frac{\sum(\text{M} \times \text{U})}{\sum \text{U}}$$

Where: M = mark received in a course, U = units of credit for a course.

- For example, a student receives the following results for her/his courses: 80, 81, 82, 83, 84. The first three of these courses are 6 UOC and the last two are 3 UOC. The WAM is calculated as: $\frac{[(80\times6)+(81\times6)+(82\times6)+(83\times3)+(84\times3)]}{(6+6+6+3+3)} = 81.625 \approx 81.63$.

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

- Unswid: a unswid of student which is taken from `People.unswid` field.
- Program name: a program name which is taken from `programs.name` field.
- WAM: the WAM result and round it to the nearest 0.01. Use the same rule as Question 4.

Special output:

- If the student has enrolled in one program, but she/he did not register for any course or all the courses registered are not included in WAM calculation for that program, i.e., the divisor is zero, return '`No WAM Available`' in the WAM section of the return line.
- If the student has not enrolled in any programs (cannot find any programs for the student), return a line in the format of '`WARNING: Invalid Student Input [X]`', where 'X' denotes the provided `unswid`.

7. AutoTest Checking

Before you submit your solution, you should check and test its correctness by using the following operations. **For each PostgreSQL question, we provide six testcases** (E.g., for question 9, they are from q9a to q9f). The testcases can be found from the Line 222 in check.sql file:

```
$ dropdb proj1                                  ... remove any existing DB
$ createdb proj1                                ... create an empty database
$ psql proj1 -f /home/cs9311/web/24T1/proj/proj1/mymyunsw.dump
                                                ... load the MyMyUNSW schema & data
$ psql proj1 -f /home/cs9311/web/24T1/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_q9a();                     ... check your solution to question9(a)
...
proj1=# select check_q10e();                    ... check your solution to question10(e)
...
proj1=# select check_all();                     ... check all your solutions
```

Notes:

1. You must ensure that your submitted proj1.sql file will be loaded and run 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. For all the submission files, you must make sure there is no error occurring when using the autotest provided above. If we need to manually fix problems in your proj1.sql file to test it (e.g., change the order of some definitions), you will be fined half of the mark as penalty for each problem.
2. In addition, write queries that are reasonably efficient.
   a. For each question, the result must be produced within 120 seconds on the nw-syd-vxdb server. Failure to do so will incur a penalty, deducting half of the mark. This time constraint applies to executing the command 'select * from check_Qn()'.

8. Project Submission

You can submit this project by doing the following:

- You are required to submit an electronic version of your answers via **Moodle**.
- We only accept the **.sql** format. Please name your files in the following format to submit: proj1_**zID**.sql (e.g., **proj1_z5000000.sql**).
- Only the **Latest Submission** is marked. The Latest Submission after the deadline will result in a late penalty.
- In case the system is not working properly, please ensure to take these steps: keep a copy of your submitted file on the CSE server without any post-deadline modifications. If you're uncertain about how to do this, refer to the guidelines provided on Taggi.

The proj1_zID.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_zID.sql file will be loaded.
- Each checking function will be executed, and the results will be recorded.

9. Late Submission Penalty

- 5% of the total mark (50 marks) will be deducted for each additional day.
- Submissions that are more than five days late will not be marked.

10. Plagiarism

The work you submit must be your own work. Submission of work partially or completely derived from any other person or jointly written with any other person is not permitted. The penalties for such an offence may include negative marks, automatic failure of the course and possibly other academic discipline.

All submissions will be checked for plagiarism. The university regards plagiarism as a form of academic misconduct and has very strict rules. Not knowing the rules will not be considered a valid excuse when you are caught.

- For UNSW policies, penalties, and information to help avoid plagiarism, please see: https://student.unsw.edu.au/plagiarism.
- For guidelines in the online ELISE tutorials for all new UNSW students: https://subjectguides.library.unsw.edu.au/elise/plagiarism.