Project 2 PLpgSQL

Please make sure you have submitted all the questions before:

Fri 18 May, 5:00pm

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

The goal is to build some useful data access operations on the MyMyUNSW database. A theme of this project is "dirty data". As I was building the database, using a collection of reports from UNSW's information systems and the database for the academic proposal system (MAPPS), I discovered that there were some inconsistencies in parts of the data (e.g. duplicate entries in the table for UNSW buildings, or students who were mentioned in the student data, but had no enrolment records, and, worse, enrolment records with marks and grades for students who did not exist in the student data). I removed most of these problems as I discovered them, but no doubt missed some. Some of the exercises below aim to uncover such anomalies; please explore the database and let me know if you find other anomalies.

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 **proj2.sql** template there
- you **must** use the create statements in **proj2.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 **proj2.sql**
- check that your solution is correct by verifying against the example outputs and by using the check_qX() functions
- test that your **proj2.sql** file will load *without error* into a database containing just the original MyMyUNSW data
- double-check that your **proj2.sql** file loads in a *single pass* into a database containing just the original MyMyUNSW data

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, however, 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 proj2**
$ **psql proj2 -f /home/cs9311/web/18s1/proj/proj2/mymyunsw.dump**

If everything proceeds correctly, the load output should look something like:

```
SET
... a whole bunch of these
SET
```

CREATE EXTENSION
COMMENT
SET
CREATE DOMAIN
CREATE DOMAIN
CREATE DOMAIN
CREATE DOMAIN
CREATE TYPE
CREATE DOMAIN
... a whole bunch of these
CREATE DOMAIN
CREATE FUNCTION
CREATE FUNCTION
SET
SET
CREATE TABLE
... a whole bunch of these
CREATE TABLE
ALTER TABLE
ALTER TABLE
... a whole bunch of these
ALTER TABLE

Please carefully investigate any occurrence of ERROR during the load. 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, the file proj2.tar.gz contains copies of the files: mymyunsw.dump, proj2.sql to get you started. You can grab the check.sql separately, once it becomes available. If you copy proj2.tar.gz to your home computer, extract it, and perform commands analogous to the above, you should have a copy of the MyMyUNSW database that you can use at home to do this project.

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 proj2**
... PostgreSQL welcome stuff ...

proj2=# **\d**
... look at the schema ...
proj2=# **select * from Students;**
... look at the Students table ...
proj2=# **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 ...
proj2=# **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 ...
proj2=# **select count(*) from Course_Enrolments;**
... how many course enrolment records ...
proj2=# **select * from dbpop();**
... how many records in all tables ...
proj2=# ... etc. etc. etc.
proj2=# **\q**

You will find that some tables (e.g. Books, Requirements, etc.) are currently unpopulated; their contents are not needed for this project. You will also find that there are a number of views and functions defined in the database (e.g. dbpop() and transcript() from above), which may or may not be useful in this project.

Summary on Getting Started

To set up your database for this project, run the following commands in the order supplied:

$ **createdb proj2**
$ **psql proj2 -f /home/cs9311/web/18s1/proj/proj2/mymyunsw.dump**
$ **psql proj2**
... run some checks to make sure the database is ok
$ **mkdir *Project2Directory***
... make a working directory for Project 2
$ **cp /home/cs9311/web/18s1/proj/proj2/proj2.sql *Project2Directory***

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 **proj2.sql** template file for your work
- you may define as many additional functions and views as you need, provided that (a) the definitions in proj2.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
- you are not allowed to use ***limit*** in answering any of the queries in this project

- when queries ask for people's names, use the `People.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

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

Note that the mymyunsw.dump used in project 2 is different from that used in project 1, please confirm that you load the correct database when you start your work.

Q1 (8 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 `Q1`.

Please write a PLpgSQL function `Q1(course_id integer)` that takes a `course_id` as parameter and outputs two numbers: (1) among all the rooms in UNSW, the number of rooms which can carry all the students enrolled in this course (i.e., `rooms.capacity` >= the total number of students enrolled in this course); (2) among all the rooms in UNSW, the number of rooms which can carry all the students enrolled in this course and also carry all the students in the enrolment waitlist of this course (i.e., `rooms.capacity` >= the total number of students enrolled in this course + the total number of students in the enrolment waitlist of this course).

You should use the following type definition and function header:

```
create type RoomRecord as (valid_room_number integer,
bigger_room_number integer);
create or replace function Q1(course_id integer) returns setof
RoomRecord...
```

Note:
(1). If a given `course_id` is **not a valid** UNSW course id, please throw an exception '`INVALID COURSEID`'.
(2). All the students enrolled (in a given course) should be counted even though his/her mark is null.

Sample results (details can be found in check.sql):
proj2=#select * from q1(52491);

```
 valid_room_number | bigger_room_number
-------------------+--------------------
                29 |                 29
(1 row)
```

Q2 (8 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 `Q2`.

Please write a PLpgSQL function `Q2(staff_id integer)` that takes a staff's `id` as parameter and returns all teaching records of the given staff. Each transcript tuple should contain the following information: cid, term, code, name, uoc, avg_mark, highest_mark, median_mark, totalEnrols.

You should use the following type definition for the transcript tuples:

```
create type TeachingRecord as (
      cid integer,     -- course ID
      term char(4),    -- semester code (e.g. 98s1)
      code char(8),    -- UNSW-style course code (e.g. COMP1021)
      name text, -- corresponding subject name of the course
      uoc integer,     -- units of credit of this course
      average_mark integer, -- the average of marks of this course
      highest_mark integer, -- the highest mark of this course
      median_mark integer,  -- the median of marks of this course
      totalEnrols integer,  -- the total number of students
                            enrolled in this course with non-null
                            mark
);
```

Note:
(1). If a given `staff_id` is **not a valid** UNSW staff id, please throw an exception '`INVALID STAFFID`'.
(2). The `average_mark` and `median_mark` should be rounded to the nearest integer.
(3). A student whose mark is null should not be counted when you calculate the `average_mark`, `highest_mark`, `median_mark` and `totalEnrols`.
(4). Ignore the teaching record where `totalEnrols` = 0.
(5). Return null value if the `uoc` is null.

You should use the following and function header:

```
create or replace function Q2(staff_id integer) returns setof
TeachingRecord...
```

Sample results (details can be found in check.sql):
proj2=#select * from q2(50413833);

cid	term	code	name	uoc	average_mark	highest_mark	median_mark	totalenrols
48548	10s1	JURD7281	Property, Equity & Trusts 1	6	72	82	73	6
48642	10s1	LAWS2381	Property, Equity & Trusts 1	6	69	88	69	93
65702	12s2	JURD7281	Property, Equity & Trusts 1	6	69	75	70	5
65703	12s2	JURD7282	Property and Equity 2	6	70	90	72	44
65837	12s2	LAWS2382	Property and Equity 2	6	70	88	71	84

(5 rows)

Q3 (9 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 `Q3`.

Given the id of an organizational unit, please write a PLpgSQL function `Q3 (org_id integer, num_courses integer, min_score integer)` to help the UNSW administrative

officers to find out all the students satisfying the following: (1) he/she has taken more than `num_courses` courses offered by the given organization (note that a student may fail a course and take it again, thus we need to **count duplicate courses**); (2) he/she has got score no less than `min_score` for at least one course offered by the given organization. Each tuple should include:

- His/Her `unswid` (should be taken from `People.unswid`)
- His/Her `student_name` (should be taken from `People.name`)
- His/Her `course_records`

`course_records` of a student is a concatenation of several records with ‘,’. Each record is about a course he/she has taken and is offered by the given organization. Each record should include the code of the course (`Subjects.code`), the name of the course (`Subjects.name`), the semester that he/she has learned this course (`Semesters.name`), the name of the organization that offers the course (`OrgUnits.name`), and the score he/she got (`Course_enrolments.mark`).

Note:
(1). If a given `OrgUnits.id` is **not a valid** UNSW organization id, please throw an exception ‘`INVALID ORGID`’.
(2). A given organization may have lots of sub-organizations. For example, the faculty of engineering has 9 schools, such as biomedical engineering and CSE, and each school may have several divisions or departments. You need **to include all the sub-organizations recursively**.
(3). Course records of a student should be sorted in **descending order** of score (**null score value should always be sorted to the end and displayed as ‘null’**). If two or more course records have the same score, they should be sorted in **ascending order** of `Courses.id`. Records should be concatenated by ‘\n’, so that each record will be displayed in a separate line.
(4). Please note that, for `course_records`, there is a **space** after each ‘,’. All text fields are verified with exact text matching.
(5). Only the **first five** course records sorted in above order should be displayed. If the course records of a student are less than five, all the course records should be displayed.

You should use the following type definition and function header:

```
create type CourseRecord as (unswid integer, student_name text,
course_records text);
create or replace function Q3(org_id integer, num_courses integer,
min_score integer) returns setof CourseRecord...
```

Sample results (details can be found in check.sql):
proj2=# select * from q3(52, 35, 100);

```
 unswid  |  student_name   |                                                                                                  course_records
---------+-----------------+-------------------------------------------------------------------------------------------------------------------------------------------------
 3206256 | Nye Cavallo     | OPTM5271, Research Project 5B, Sem2 2012, Optometry and Vision Science, School of, 100                                                          +
         |                 | BABS1201, Molecules, Cells and Genes, Sem1 2008, Biotechnology and Biomolecular Sciences, School of, 95+
         |                 | PSYC4111, Psych and Stats for Optometry, Sem1 2011, Psychology, School of, 94                                                                   +
         |                 | VISN1211, Vision Science 1, Sem2 2008, Optometry and Vision Science, School of, 93                                                              +
         |                 | OPTM4151, Ocular Therapeutics 4A, Sem1 2011, Optometry and Vision Science, School of, 91                                                        +
         |                 |
 3219926 | Ningbo Su       | OPTM5171, Research Project 5A, Sem1 2012, Optometry and Vision Science, School of, 100                                                          +
         |                 | PSYC4111, Psych and Stats for Optometry, Sem1 2011, Psychology, School of, 95                                                                   +
         |                 | VISN2231, Introduction to Ocular Disease, Sem2 2009, Optometry and Vision Science, School of, 87                                                +
         |                 | OPTM3131, Ocular Disease 3A, Sem1 2010, Optometry and Vision Science, School of, 86                                                             +
         |                 | OPTM4291, Optom Med, Sem1 2011, Optometry and Vision Science, School of, 85                                                                     +
         |                 |
 3227561 | Nicholas Barry  | VISN2111, Physiology of the Ocular Syste, Sem1 2009, Optometry and Vision Science, School of, 100                                               +
         |                 | OPTM5171, Research Project 5A, Sem1 2012, Optometry and Vision Science, School of, 100                                                          +
         |                 | VISN3111, Aging of the Visual System, Sem1 2010, Optometry and Vision Science, School of, 99                                                    +
         |                 | VISN2131, Optics and the Eye, Sem1 2009, Optometry and Vision Science, School of, 98                                                            +
         |                 | OPTM3131, Ocular Disease 3A, Sem1 2010, Optometry and Vision Science, School of, 98                                                             +
         |                 |
(3 rows)
```

Note that PostgreSQL uses a ‘+’ character to indicate an end-of-line in its output (as well as printing '\n').

6. Submission

You can submit this project by doing the following:

- The file name should be proj2.sql.
- Ensure that you are in the directory containing the file to be submitted.
- Type "give cs9311 proj2 proj2.sql"
- If you submit your project more than once, the last submission will replace the previous one
- **To prove successful submission, please take a screenshot as assignment submission manual shows and keep it by yourself.**
- If you have any problems in submissions, please email to kai.wang@unsw.edu.au. You can also ask questions about this project in our project online Q&A group, we will answer your questions as soon as possible.

The proj2.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 will be loaded (with expected results appropriate for the database)
- the contents of your proj2.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 proj2** ... remove any existing DB
$ **createdb proj2** ... create an empty database
$ **psql proj2 -f /home/cs9311/web/18s1/proj/proj2/mymyunsw.dump** ... load the MyMyUNSW schema and data
$ **psql proj2 -f /home/cs9311/web/18s1/proj/proj2/check.sql** ... load the checking code
$ **psql proj2 -f proj2.sql** ... load your solution
$ **psql proj2**
proj2=# **select check_q1();** … check your solution to question1
proj2=# **select check_q2();** … check your solution to question2
proj2=# **select check_q3();** … check your solution to question3
proj2=# **select check_all();** … check all your solutions to q1-q3
Note: if your database contains any views or functions that are not available in a file somewhere, you should put them into a file before you drop the database.

If your code does not load without errors, fix it and repeat the above until it does.

You must ensure that your proj2.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 we need to manually fix problems with your proj2.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, please output results within 60 seconds. This time restriction applies to the execution of the 'select * from check_Qn()' calls. For each question, half of the total mark will be deducted if your solution cannot output results within 60 seconds.

7. Late Submission Penalty

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