Project 1 SQL and PLpgSQL

Due: Sun 20 April, 23:59

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, I discovered that there were some inconsistencies in parts of the data (e.g. 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.

Note also that, as the database was constructed, I made some small changes to the schema, in order to deal with the data I had available. For example, there was no starting time for some of the staff affiliations, and so I had to remove the `not null` constraint on `Affiliation.starting` in order to load a whole bunch of staff affiliation data. You should make sure that you check the latest version of the schema and use that; if in doubt, check the meta-data in the database itself (using `psql`'s `\d` command).

2. How to do this project:

- read this specification carefully and completely
- familiarise 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.

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 COMP3311 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 /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 40 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 34MB 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're running PostgreSQL at home, the file proj1.zip contains copies of the files: mymyunsw.dump, proj1.sql to get you started. You can grab the check.sql separately, once it becomes available. If you copy proj1.zip to your home computer, unzip 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 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 CourseEnrolments;**
... how many course enrolment records ...
proj1=# **select * from dbpop();**
... how many records in all tables ...
proj1=# **select * from transcript(3231850);**
... transcript for student with ID 3231850 ...
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. 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 proj1**
$ **psql proj1 -f /mymyunsw.dump**
$ **psql proj1**
... run some checks to make sure the database is ok
$ **mkdir *Project1Directory***
... make a working directory for Project 1
$ **cp /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 customise 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
- do not use values of id fields if you can refer to tuples symbolically; e.g. if a question asks about lecture classes, do **not** use the fact the id of the lecture class type is 1 and check for `classtypes.id=1`; instead check for `classtypes.name='Lecture'`
- 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`
- 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

An important note related to marking:

- make sure that your queries are reasonably efficient (defined as taking < 15 seconds to run)
- use psql's \timing feature to check how long your queries are taking; they must each take less than 20000 ms
- queries that are too slow will be **penalised by half of the mark for that question**, even if they give the correct result

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. Exercises

Q1 (5 marks)

Define an SQL view `Q1(name, school, starting)` which gives details about all of the current Heads of School in UNSW. The view should return the following details about each Head:

- their name (use the `name` field from the `People` table)
- the school (use the `longname` field from the `OrgUnits` table)
- the date that they were appointed (use the `starting` date from the `Affiliation` table)

Since there is some dirty-looking data in the `Affiliation` table, you will need to ensure that you return only legitimate Heads of School. Apply the following filters:

- only choose people whose role is `'Head of School'`

- only choose people who have not finished in the role
- only choose people for whom this is their primary role
- only choose organisational units whose type is actually 'School'

Q2 (5marks)

Using the view from the previous question, write a new view Q2(status,name,school,starting) which returns a table containing only the longest-serving and most-recent current Heads of School. Each tuple in the view result should contain the following:

- the status: either the string 'Longest serving' or the string 'Most recent' (cast to type text)
- the same three values (name, school, starting-date) for each Head of School

You should *not assume* that there is only one longest-serving or one most-recent Head of School; several Heads may have started in the role on the same day.

You must also ensure that no warning messages are generated during the loading of your view.

Q3 (5 mark)

Dealing with Terms.id values is not ideal when referring to semesters. CSE typically uses shorthand symbolic names like "12s1" and "00s2" to refer to terms. Write an SQL *function* that takes as parameter aTerms.id value and returns a symbolic term name, consisting of the last two digits of the year, and the session code (with a lower-case letter).

The function should be defined as follows:

create or replace function Q3(integer) returns text
as $$ *SQL query* $$ language sql;

If the parameter value does not correspond to a known term, the function should simply return a NULL value.

Q4 (6 marks)

UNSW management occasionally expresses concern that the university is too heavily dependent on international student enrolments. Write a view Q4(term,percent) that will help them to monitor this by indicating the percentage of international students enrolled in each semester. Each tuple in the view result should contain the following:

- the name of the term in the format 09s1, 09s2, 10s1, etc.
- the fraction (internationals/total enrolments) as numeric(4,2)

Each student's status as local or international can be determined from the Students.stype field. You should compute the number of enrolled students by considering just the ProgramEnrolments for each semester. You can assume that no student is registered as both local and international in the same semester. Also, only consider the main semesters (S1 and S2) and

only consider years from 2005 onwards. Do the fraction calculation using floating point values to achieve the highest precision.

Q5 (7 marks)

In the previous question, we measured enrolments in terms of the number of students enrolled in programs. For funding purposes, the University measures enrolments in terms of full-time equivalent (FTE) students. A full-time enrolment is a student enrolled in 24UOC worth of courses. Thus, the FTE number for a given semester can be computed by looking at the total UOC for enrolled courses in that semester and dividing by 24.

Write a view `Q5(term, nstudes, fte)` that gives, for each main semester (S1 and S2) from 2000 S1 to 2010 S2, the number of students enrolled and the FTE for that semester. Each tuple in the view result should contain the following:

- the name of the term in the format `09s1`, `09s2`, `10s1`, etc.
- the total number of distinct students enrolled in courses in that semester
- the FTE (total enrolled UOC / 24) for that semester as `numeric(6,1)`

You should compute enrolment information from the `CourseEnrolments` table (i.e. no need to consider `ProgramEnrolments`). Do the FTE calculation using floating point values to achieve the highest precision.

Q6 (7 marks)

Some of the data in this database is a little sparse; for example, most of the courses have no `CourseStaff` specified. We want to find extreme examples where a course has been offered very many times, but has never had any staff allocated to it according to the database.

Write a view `Q6(subject,nOfferings)` that gives a list of subjects which have been offered more than 30 times, but have never had any staff associated with them (via the `CourseStaff` table). Each tuple in the view result should contain the following:

- the subject code and title, in the fromat e.g. COMP3311 Database Systems
- the number of times that this subject has been offered as a `Course`

6. Submission

Submit this project by doing the following:

- Ensure that you are in the directory containing the file to be submitted.
- Type “give cs3311 proj1 proj1.sql”

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** to 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 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 ..../mymyunsw.dump** ... load the MyMyUNSW schema and data
$ **psql proj1 -f ..../check.sql** ... load the checking code
$ **psql proj1 -f proj1.sql** ... load your solution

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 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 a 2 mark penalty for each problem.

7. Late Submission Penalty

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