Project 2 SQL and PLpgSQL

Due: Sun 17 May, 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 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
- familiarise 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
- 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 proj2**
$ **psql proj2 -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 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're running PostgreSQL at home, the file proj2.zip 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.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 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 CourseEnrolments;**
... how many course enrolment records ...
proj2=# **select * from dbpop();**
... how many records in all tables ...
proj2=# **select * from transcript(3197893);**
... transcript for student with ID 3197893 ...
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 …/mymyunsw.dump**
$ **psql proj2**
... run some checks to make sure the database is ok
$ **mkdir *Project2Directory***
... make a working directory for Project 2
$ **cp ./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 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 10000 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 (7 marks)

Define an SQL function (SQL, not PLpgSQL) called Q1(integer) that takes as parameter either a People.id value (i.e. an internal database identifier) or a People.unswid value (i.e. a UNSW student ID), and returns the name of that person. If the id value is invalid, return an empty result. You can assume that People.id and People.unswid values come from disjoint sets, so you should never return multiple names. (It turns out that the sets aren't quite disjoint, but I won't test any of the cases where the id/unswid sets overlap)

The function must use the following interface:

create or replace function Q1(integer) returns text ...

Q2 (12 marks)

The transcript(integer) function from lectures is supplied in the database. Define a new PLpgSQL function Q2(integer), which is based on this function but returns a new kind of transcript record, called NewTranscriptRecord, that includes in each row (except the last) the 4-digit program code for the program being studied when the course was studied.

Use the following definition for the new-style transcript tuples:

```
create type NewTranscriptRecord as (
      code  char(8), -- UNSW-style course code (e.g. COMP1021)
      term  char(4), -- semester code (e.g. 98s1)
      prog  char(4), -- program being studied in this semester
      name  text,    -- short name of the course's subject
      mark  integer, -- numeric mark acheived
      grade char(2), -- grade code (e.g. FL,UF,PS,CR,DN,HD)
      uoc   integer  -- units of credit awarded for the course
);
```

Note that this type is already in the database so you won't need to define it. In order to get access to the source code for the transcript() function, use the following command in psql and save the function definition in a local file where you can edit it:

```
proj2=# \ef transcript(integer)
```

Q3 (16 marks)

An important part of defining academic rules in MyMyUNSW is the ability to define groups of academic objects (e.g. groups of subjects, streams or programs) In MyMyUNSW, groups can be defined in three different ways:

- enumerated by giving a list of objects in a X_members table
- pattern by giving a pattern that identifies all relevant objects
- query by storing an SQL query which returns a set of object ids

In all cases, the result is a set of academic objects of a particular type

Write a PLpgSQL function Q3(integer) that takes the internal ID of an academic object group and returns the codes for all members of the academic object group. Associated with each code should be the type of the corresponding object, either subject, stream or program. For this question, you only need to consider groups defined via a pattern. If the supplied object group ID refers to an enumerated or query type group, you may simply return an empty result. You should return distinct codes (i.e. ignore multiple versions of any object), and there is no need to check whether the academic object is still being offered.

The function is defined as follows:

```
create or replace function Q3(integer) returns setof AcObjRecord
```

where AcObjRecord is already defined in the database as follows:

```
create type AcObjRecord as (
            objtype text,  -- academic object's type e.g. subject, stream, program
            object  text,  -- academic object's code e.g. COMP3311, SENGA1, 3978
);
```

Groups of academic objects are defined in the tables:

- acad_object_groups(id, name, gtype, glogic, gdefby, negated, parent, definition)
  where the most important fields are:
  - gtype ... what kind of objects in the group
  - gdefby ... how the group is defined
  - definition ... where queries or patterns are given
- program_group_members(program, ao_group) ... for enumerated program groups
- stream_group_members(stream, ao_group) ... for enumerated stream groups
- subject_group_members(subject, ao_group) ... for enumerated subject groups

The ways of specifying object groups are quite flexible, and groups can be defined hierarchically. For this exercise, however, you can ignore groups defined in terms of child groups. You can also ignore negated groups, which would probably result in very large sets of objects. In these cases, simply return an empty result. You can also ignore the glogic field.

There are a wide variety of patterns. You should explore the acad_object_groups table yourself to see what's available. To give you a head start, here are some existing patterns and what they mean:

- COMP2### ... any level 2 computing course (e.g. COMP2911, COMP2041)
- COMP[34]### ... any level 3 or 4 computing course (e.g. COMP3311, COMP4181)
- FREE#### ... any free elective; for this case, simply return the pattern itself**
- GENG#### ... any Gen Ed course; for this case, simply return the pattern itself**
- ####1### ... any level 1 course at UNSW
- (COMP|SENG|BINF)2### ... any level 2 course from CSE
- COMP1917,COMP1927 ... core first year computing courses
- COMP1###,COMP2### ... same as COMP[12]###

Your function should be able to expand any pattern element from the above classes of patterns (i.e. pattern elements that include #, [...] and (...|...)), except for FREE#### and GENG#### as noted.

** Note that there are some variations on the FREE and GEN patterns that should also be treated specially. Any pattern element that begins with "FREE" should be returned unchanged. Similarly for the patterns "GEN#####" and "ZGEN####".

Patterns can be qualified by constraint clauses (e.g. /F=ENG) and alternatives can be specified (e.g. {MATH1131;MATH1141}), but you don't need to be able to handle these. If a pattern does contain one of these, simply return an empty result.

Note:

1) For the cases that are not clearly stated in Q3, just simply return an empty result. For example: ARTSxxxx returns empty result.

2) For the FREE#### or GENG###, you can return FREE#### according to Q3. However, if you just return empty result for this case, it is also accepted.

3) For the cases {{MATH1131;MATH1141}), } or {/F=ENG }, just simply return empty result if a pattern does contain one of these.

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 proj2 proj2.sql”

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** 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 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 ..../mymyunsw.dump** ... load the MyMyUNSW schema and data
$ **psql proj2 -f ..../check.sql** ... load the checking code
$ **psql proj2 -f proj2.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 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 I 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 a 2 mark penalty for each problem.

7. Late Submission Penalty

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