

Project 1 SQL

The deadline for project 1 is:

Sun 1 Nov, 5:00 pm (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.
- 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 available 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/20T3/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/20T3/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/20T3/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 (staff_role, course_num)` that gives the name of staff role (refers to the `staff_roles.name`) and the number of distinct courses taught by each type of staff in the year 2010 (refers to `Semesters.year`). The results should be ordered by `course_num` from lowest to highest.

Note:

- The distinct courses are considered based on the `courses.id` field.
- Only consider the staff roles that teach more than **one** courses in the year 2010.

Q2 (3 marks)

Define an SQL view `Q2 (course_id)` that gives the id of the course that the Studio classes of this course are held in **3** different buildings. The view should return the following details about each course:

- `course_id` should be taken from `Courses.id` field.

Note:

- The type of class, i.e. Studio, refers to the `Class_types.Name`).

Q3 (4 marks)

Define a SQL view `Q3 (course_num)` that the number of all the distinct **courses** that has the classrooms equipped with **both** Student wheelchair access and Teacher wheelchair access (refers to the `facilities.description`). We only consider the courses enrolled by at least one international student.

Note:

- The international students mean the students having the type 'intl' (refers to `students.stype`)
- Each returned course should have at least one classroom that equipped with **both** Student wheelchair access and Teacher wheelchair access.

Q4 (4 marks)

Define a SQL view `Q4 (unswid, name)` that gives all the distinct local students who have enrolled in the course offered by the School of Chemical Sciences (refers to the

OrgUnits.Name), and got a mark higher than 87 (refers to the Course_enrolments.mark). 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:

- The results should be ordered by the descending unswid.

Q5 (4 marks)

Define a SQL view Q5(course_id) that gives the courses, where more than 4/5 of the enrolled students in the course have higher marks than the average mark of the courses. For example, there are 20 students enrolled in the course A, and the average mark of course A is 50. If there are 17 students get the marks higher than 50, course A will be returned in the view Q5. The view should return the following details about each course:

- course_id should be taken from Courses.id field.

Note:

- When calculating the average, only consider students who have a **not null** mark.
- The mark of a course can be referred from Course_enrolments.mark.
- Only consider valid courses which have at least **10** students.

Q6 (4 marks)

Define a SQL view Q6(semester, course_num) that displays the semester(s) with the highest course_num, which is the number of the courses in that semester. The view should return the following details about each course:

- semester should be taken from Semesters.longname field.

Note:

- Only consider valid courses which have at least **10** students.
- If several semesters have the same highest maximum course number, return all of them.
- Skip the semester with no valid course.

Q7 (4 marks)

Define a SQL view Q7(course_id, avgmark, semester) that gives all the distinct selected courses with a medium average mark ($75 < \text{avgmark} < 80$) of the enrolled students in each semester of the years 2007 and 2008. The view should return the following details about each course:

- course_id 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 65 and less than 85.
- The results should be ordered by the descending course_id.

Q8(4 marks)

Define a SQL view `Q8(num)`, which gives the number of distinct intl students enrolling in the years 2009 and 2010 (refers to `Semesters.year`) in the Medicine stream (refers to `Streams.name`), but never enrolling in any course offered by the organizations having 'Engineering' in the name (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)

Database Systems course admins would like to know the average mark of each semester. Define a SQL view `Q9(year, term, average_mark)` to help them monitor the average mark each semester. Each tuple in the view should contain the following:

- the year (`Semesters.year`) of the semester
- the term (`Semesters.term`)
- the average mark of students enrolled in this course this semester as `numeric(4, 2)`

Database Systems has value 'Database Systems' in the `Subjects.name` field. You can find the information about all the course offerings for a given subject from `Courses`. You should calculate the average mark of enrolled students for a course offering from the table `Course_enrolments`.

Note:

- There are two subjects that share the same name "Database Systems", and we do not distinguish them in this question. In consequence, you may find more than one course for a single semester. In such case, there is no student enrolling in more than one course.
- When calculating the average marks, only consider **not null** mark records.
- Only consider the semesters which have 'Database Systems'.

Q10 (5 marks)

Define SQL view `Q10(year, num, unit)`, which gives, for each unit, the year with the greatest number of distinct international students enrolled, and the number of distinct international students enrolled for that year. The view should return the following details about each unit:

- year should be taken from `Semesters.year` field.
- num counts the total number of international students enrolled.
- unit should be taken from `OrgUnits.longname` field.

Note:

- You should ignore the units with no international student.
- In the case of ties, the same unit with different years should all be included in the result.
- We only count program enrolments when considering the number of student enrolments per unit for that year.

Q11 (5 marks)

Define SQL view `Q11(unswid, name, avg_mark)`, which gives the unswid and name of students with the top 10 highest rankings in semester 2011 S1. Students' rankings are ordered by `avg_mark` (i.e., the average mark of all the courses completed in 2011 S1) from highest to lowest. A student must be a full-time student (complete at least 3 courses in 2011 S1) in order to calculate his/her `avg_mark`. If there are multiple students with the same `avg_mark`, they have the same

rank, which means your view may contain more than 10 rows. 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.
- `avg_mark` as `numeric (4,2)`.

Note that for this problem:

- To complete a course, we only consider the students who receive a mark for the course taken. You may use `Course_enrolments.mark >= 0` to retrieve a list of valid students.
- To calculate `avg_mark`, only consider the courses taken in 2011 S1.
- `rank` is with gaps. i.e., if there are 2 students ranking first, the third student will be ranked as third.

Q12 (5 marks)

The university is interested in the `Lecture Theatre` usage status of the “Mathews Building” in 2010 S1. So please define SQL view `Q12(unswid, longname, rate)`, which gives the distinct room id and name of all the `Lecture Theatre` in `Matthews` building with the highest usage rate of the classes that use this theatre. 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.
- `rate` as `numeric (4,2)`

Note:

- If a lecture theatre is not used by any class, the usage rate of that theatre would be 0.00.
- The rate is calculated as dividing the total number of distinct students enrolled in a class by the total capacity of the theatre.

6. Submission

You can submit this project by doing the following:

- Students must submit an electronic copy of their answers to the above questions to the course website in Moodle.
- The file name should be `proj1_studentID.sql` (e.g., **`proj1_z5100000.sql`**).
- 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 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/20T3/proj/proj1/mymyunsw.dump ... load the MyMyUNSW
schema and data
$ psql proj1 -f /home/cs9311/web/20T3/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_q12();   ... check your solution to question12
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

20% reduction for each day late.