Huibo Zhao

hz2480

COMS4111 SEC003

HW3

1. Implement a 3 table solution for inheritance for Person, Student and Faculty

I adopt the given ddl and started from that. The details can be examined in “hw3\_ddl”.

The Person table serves as an abstract type class. It contains columns that are common to all the subclasses. More specifically in my design, the Person table contains “uni”, “last\_name”, and “first\_name”. The column “type” indicates which subclass it belongs to. In this design, the type column should be either student or faculty.

Ideally I don’t think we should directly insert data into Person table because it is an abstract class. We should only call insert student/faculty procedure and that procedure will insert corresponding data into Person table. However, in case users still want to directly insert or update Person table. I set insert and update trigger. The insert trigger set strictly the format of the uni according to the “generate\_uni” function so that users could not define their own desired uni. The update trigger prevents updating uni.

The Student table contains “uni”, “school” and “year” column. “uni” is a foreign key that references to Person table’s “uni”.

The insert trigger on Student set two constraints. First, it checks that the new inserted uni must have a match in Person table. Second, the “year” column must be between 2000 and 2020. The update trigger also sets two constraints. It prevents users from changing uni and check if updated “year” is valid.

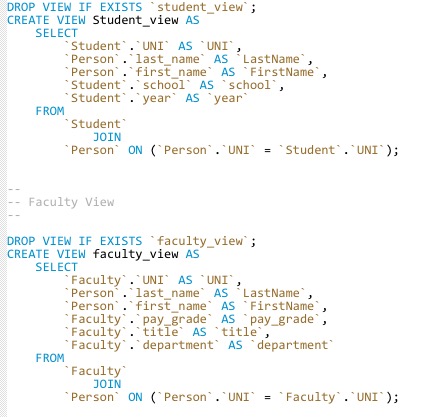
The Faculty table can be regarded as a parallel table to Student. It contains “uni”, “pay\_grade”, “title” and “department” columns. “uni” is a foreign key that references to Person table’s “uni”.

The insert trigger on Faculty checks that the new inserted uni must have a match in Person table and the pay\_grade must be in certain range. The update trigger prevents users from changing uni and check if updated pay\_grade is valid.

The above triggers test cases are trivial and pasting all test cases results is not necessary. Thus, please refer to “hw3\_ddl” for codes implementation.

In addition, a new column “enrollment\_limit” is added to the section table and it will be used later.

2.



The two views are implemented as above in “hw3\_ddl”. The student/faculty view contains column information from both subclass and abstract class (table).

The six procedures are in “hw3\_procedures” along with a “generate\_uni” function.

Each procedure would affect both the derived entity and base entity.

3.

The insert statements for testing data are in “hw3\_test”.

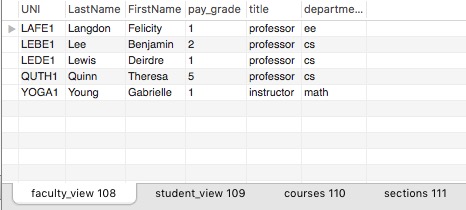


Figure: 5 faculty tuples

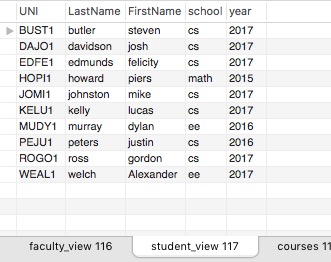


Figure: 10 student tuples

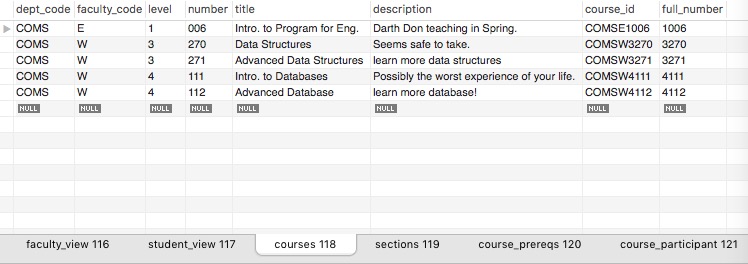


Figure: 5 courses tuples

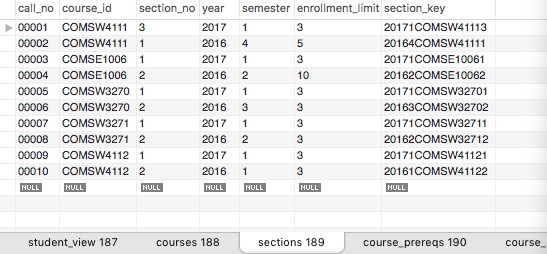


Figure: 10 section tuples

Note: the current semester is year 2017, semester 1

Therefore, I set some sections in the past

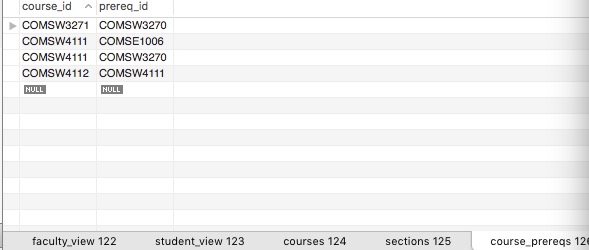


Figure: 4 prereq tuples

4.

I would like to address my test cases for testing those required constraints and then explain how I implement it.

1. A student may ONLY enroll in a section if the student has completed the course prereqs.

According to the tuple, we can find that COMSW3270 is a prerequisite course for COMS3271.

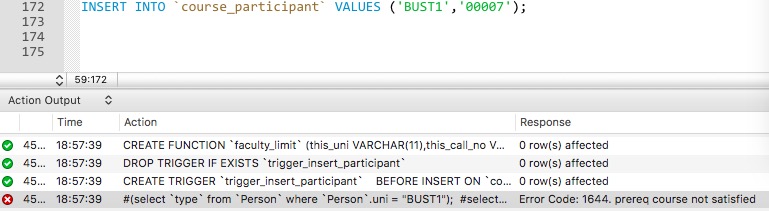


Figure: test prereq constraint

Section call # 00007 corresponds to course COMSW3271 and uni ‘BUST1’ is a random student’s uni #. As the error response suggested, the prereq course not satisfied.

Then we execute INSERT INTO `course\_participant` VALUES ('BUST1','00004');

00004 corresponds to COMSE1006, a very fundamental course. The insert operation is successful because that course does not have any prereq.

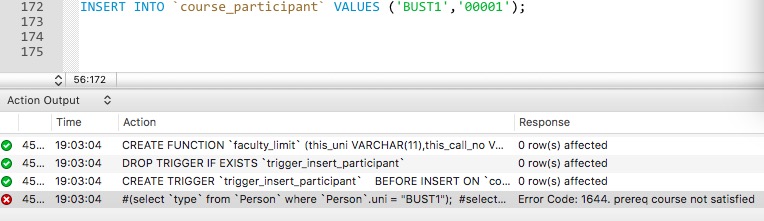


Figure: another test on prereq constraint

00001 corresponds to COMSW4111. Since this course requires two prereq courses. This student still doesn’t satisfy the prereq requirement as error response suggested.

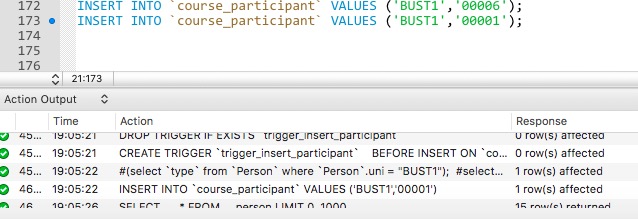


Figure: successful example

Lastly, we make student registered section 00006 (COMSW3270), which is another prereq for COMSW4111. Now, the insert operations are successful.

1. Sections have enrollment limits. A student enrollment should fail if the course is at then enrollment limit.

We take section 00003 as an example. It is course COMSE1006 and has enrollment limit size of 3.

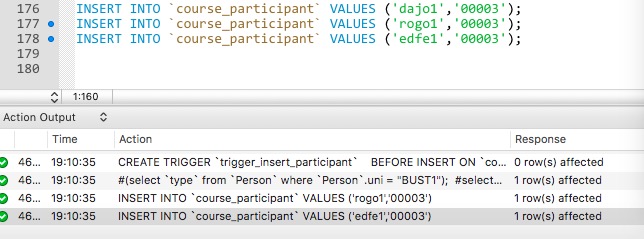


Figure: Example for testing enrollment size limit

Insert three students are all successful because this course does not have any prereq requirement and does not exceed the size limit.

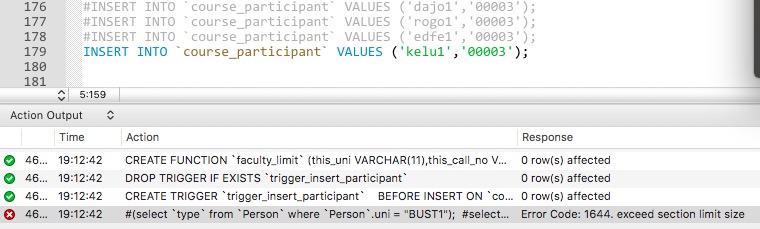


Figure: Example for testing enrollment size limit

When we try to insert another student into this section, it fails. As the error response suggested, this exceeds section limit size.

1. A faculty member may only teach 3 sections a semester.

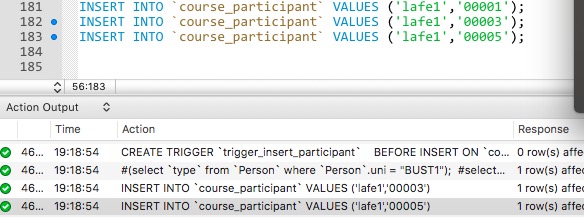


Figure: Example for testing faculty teaching limit

“lafe1” is a faculty’s uni and section 00001,00003 and 00005 are all sections in this semester. The insert are successful.

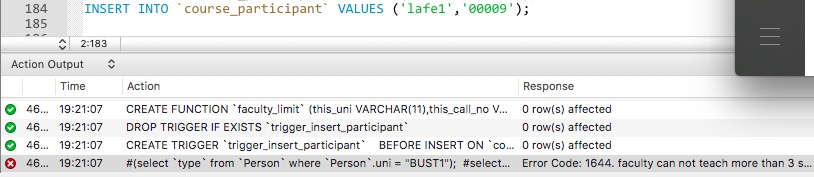


Figure: Example for testing faculty teaching limit

Section 00009 is another section in this semester. This time the insert fails. As the error response suggested, faculty cannot teach more than 3 sections a semester.

Then I would like to explain how I implement it. All the codes are in “hw3\_advanced.sql”

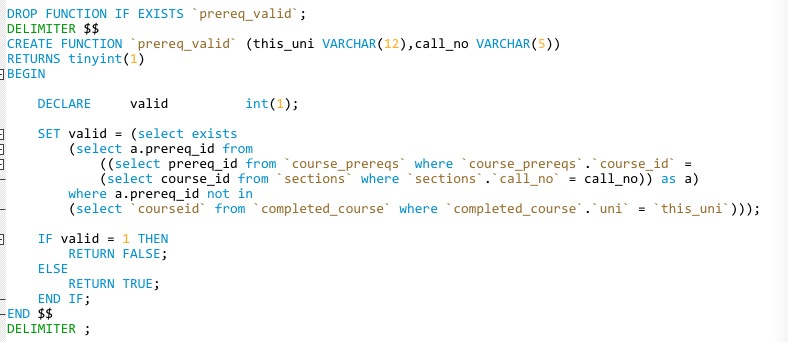


Figure: function for checking prereq course

The above function tests if students completed all required prereq courses according to another section(course). It involves using a view named “completed\_course” which I will explained later.



Figure: function for checking enrollment size limit of a section

The above function tests if a given section is full or not.

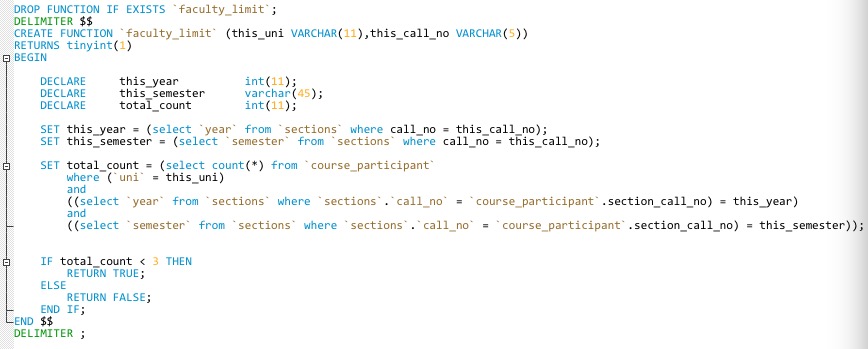


Figure: function for checking whether or not faculty can teach a new section

The above function tests whether or not faculty can teach a new section. I have considered the issue of sections in different semesters. Given a section, the function will look up how many sections that faculty teaches in that specific semester, then tells if a faculty could still teach a new section for that semester.

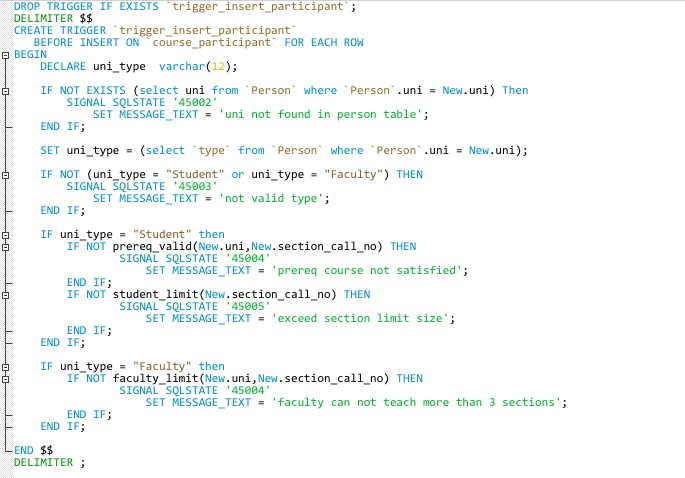


Figure: insert trigger on “course\_participant”

This is the trigger on insert “course\_participant”. It combines all the functions implemented above and constructed a trigger that covers all the constraints.

5.

The below functions and views codes can be found in “hw3\_advanced.sql”

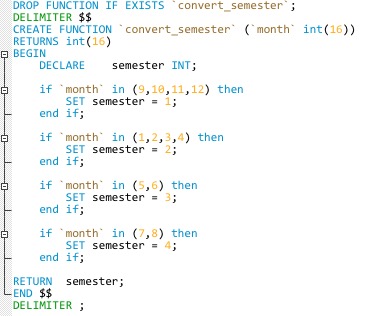


Figure: function convert\_semester

This is the function “convert\_semester”. It takes a specific month and returns what semester that month corresponds to. This function is used for creating view for “completed\_course”.



Figure: view “completed\_courses”

This view generates a tuple for each student of the form (UNI, completed course, year and semester completed). Basically, it compares the current date and the date for completing a course. If that course is completed before the current date, we add it to the view. This view is used for checking if a student satisfies the prereq requirement of a course.

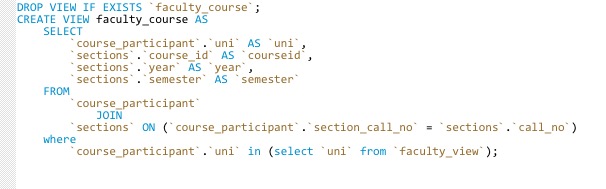


Figure: view “faculty\_course”

This view generates a tuple for each faculty member of the form (uni, course\_id, semester, year) representing each course a faculty member is teaching or has taught. This view can be used for “faculty\_limit” function that checks if faculty can still teach a new section on a semester although I do not use this view in my implementation.