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.

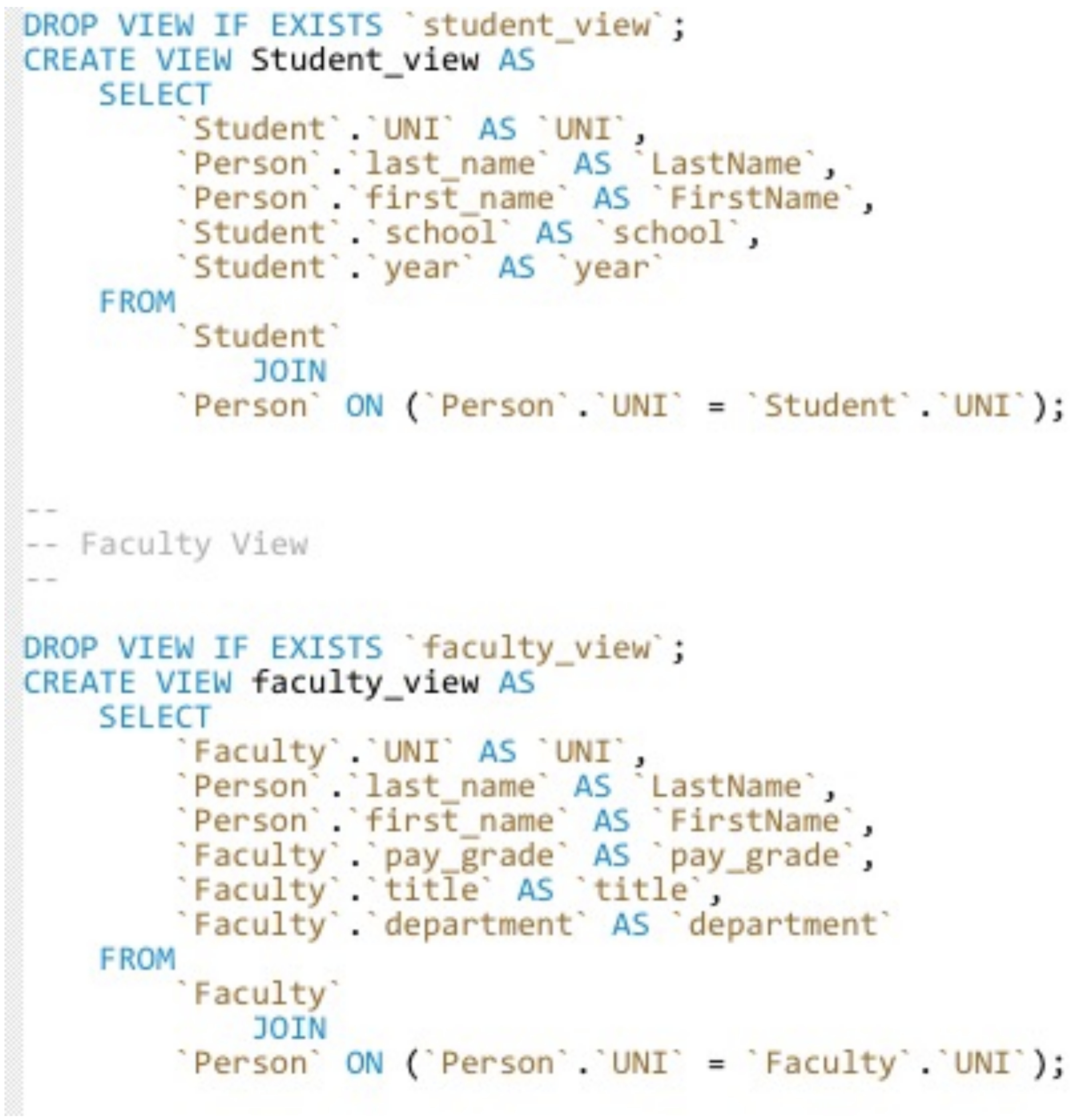
The Student table contains "uni", "school" and "year" column. "uni" is a foreign key that references to Person table's "uni".
The insert trigger on Student set two constraints. First, it checks that the new inserted uni must have a match in Person table. Second, the "year" column must be between 2000 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.



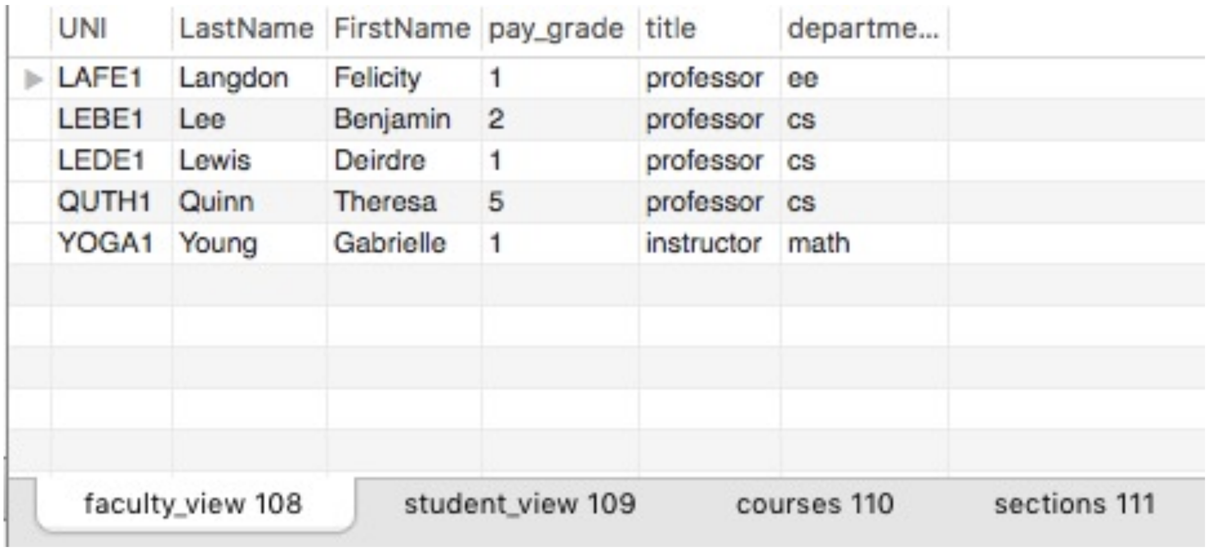
```
DROP VIEW IF EXISTS `student_view`;
CREATE VIEW Student_view AS
    SELECT
        `Student`.`UNI` AS `UNI`,
        `Person`.`last_name` AS `LastName`,
        `Person`.`first_name` AS `FirstName`,
        `Student`.`school` AS `school`,
        `Student`.`year` AS `year`
    FROM
        `Student`
            JOIN
        `Person` ON (`Person`.`UNI` = `Student`.`UNI`);


--
-- Faculty View
--

DROP VIEW IF EXISTS `faculty_view`;
CREATE VIEW faculty_view AS
    SELECT
        `Faculty`.`UNI` AS `UNI`,
        `Person`.`last_name` AS `LastName`,
        `Person`.`first_name` AS `FirstName`,
        `Faculty`.`pay_grade` AS `pay_grade`,
        `Faculty`.`title` AS `title`,
        `Faculty`.`department` AS `department`
    FROM
        `Faculty`
            JOIN
        `Person` ON (`Person`.`UNI` = `Faculty`.`UNI`);
```

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.

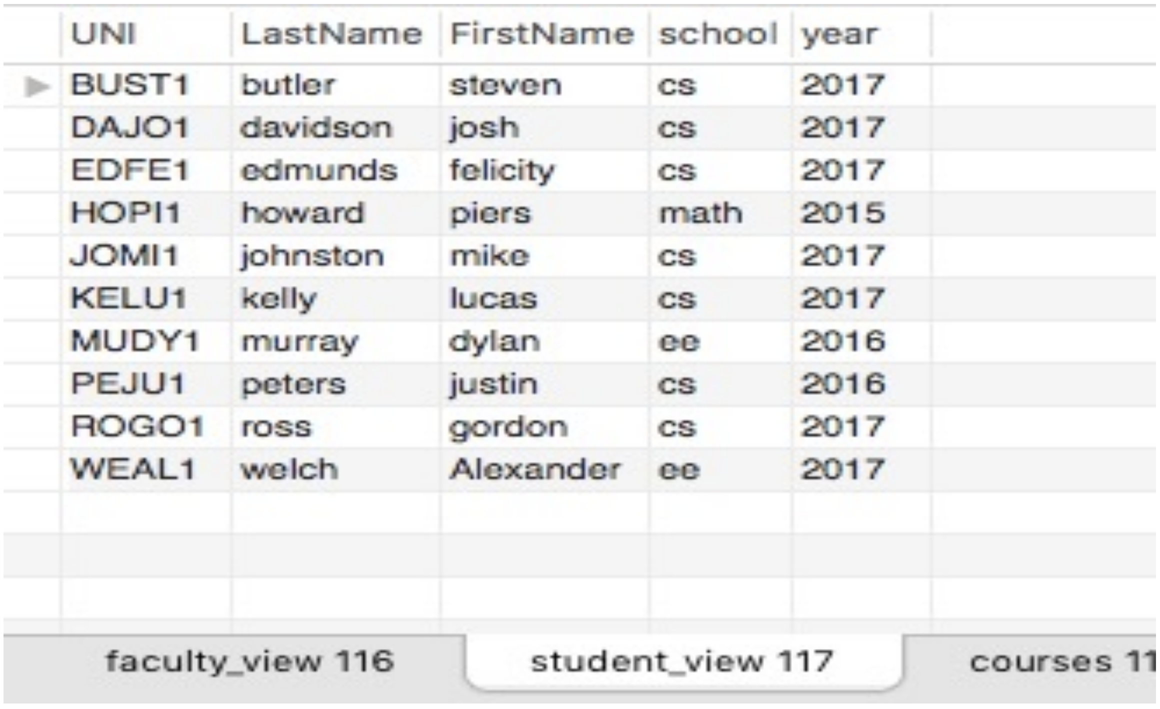
The insert statements for testing data are in “hw3_test”.



UNI	LastName	FirstName	pay_grade	title	departme...
LAFE1	Langdon	Felicity	1	professor	ee
LEBE1	Lee	Benjamin	2	professor	cs
LEDE1	Lewis	Deirdre	1	professor	cs
QUTH1	Quinn	Theresa	5	professor	cs
YOGA1	Young	Gabrielle	1	instructor	math

faculty_view 108 | student_view 109 | courses 110 | sections 111

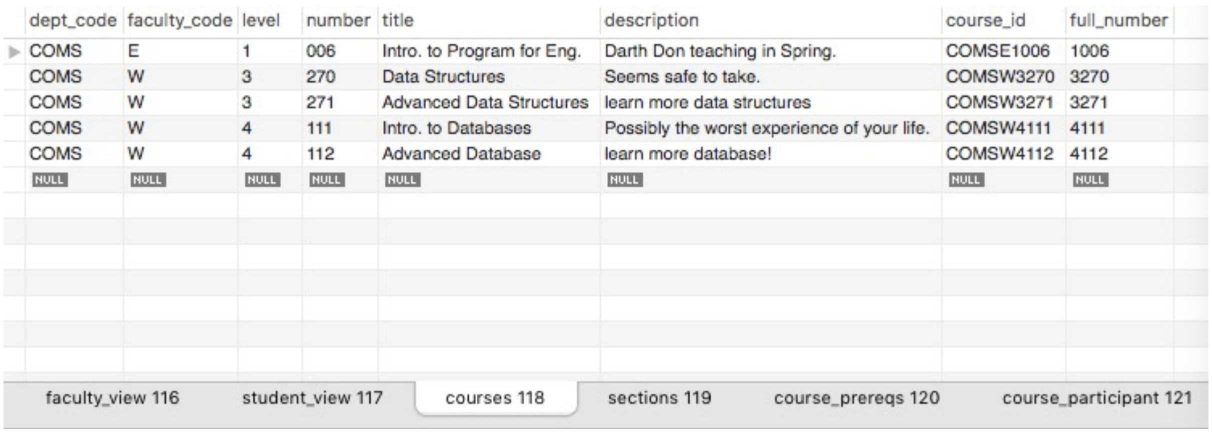
Figure: 5 faculty tuples



UNI	LastName	FirstName	school	year
BUST1	butler	steven	cs	2017
DAJO1	davidson	josh	cs	2017
EDFE1	edmunds	felicity	cs	2017
HOPI1	howard	piers	math	2015
JOMI1	johnston	mike	cs	2017
KELU1	kelly	lucas	cs	2017
MUDY1	murray	dylan	ee	2016
PEJU1	peters	justin	cs	2016
ROGO1	ross	gordon	cs	2017
WEAL1	welch	Alexander	ee	2017

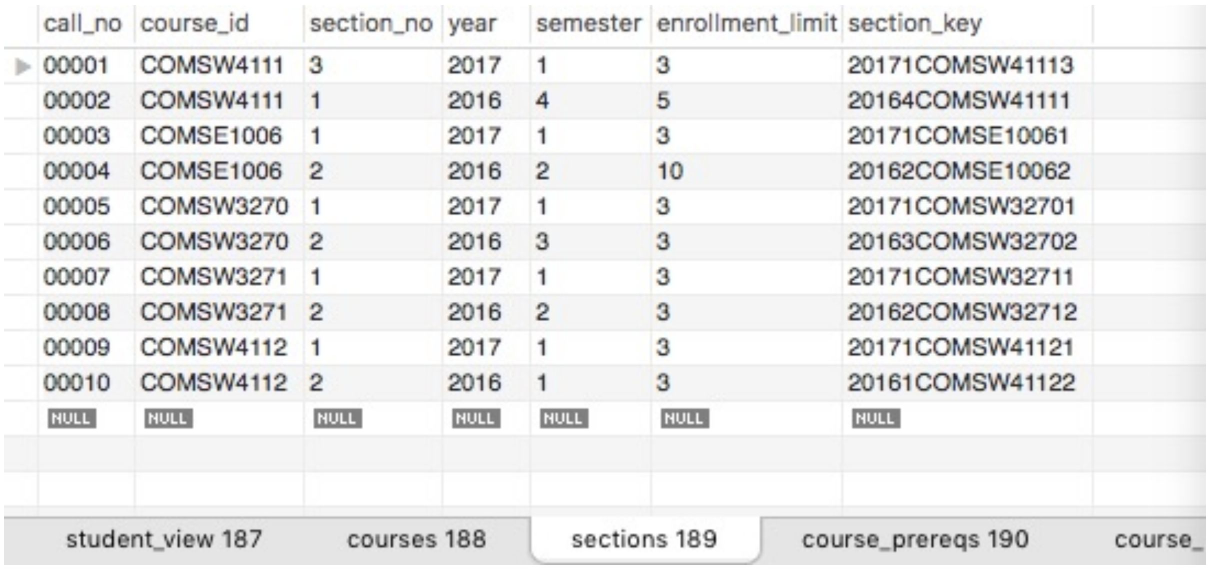
faculty_view 116 | student_view 117 | courses 11

Figure: 10 student tuples



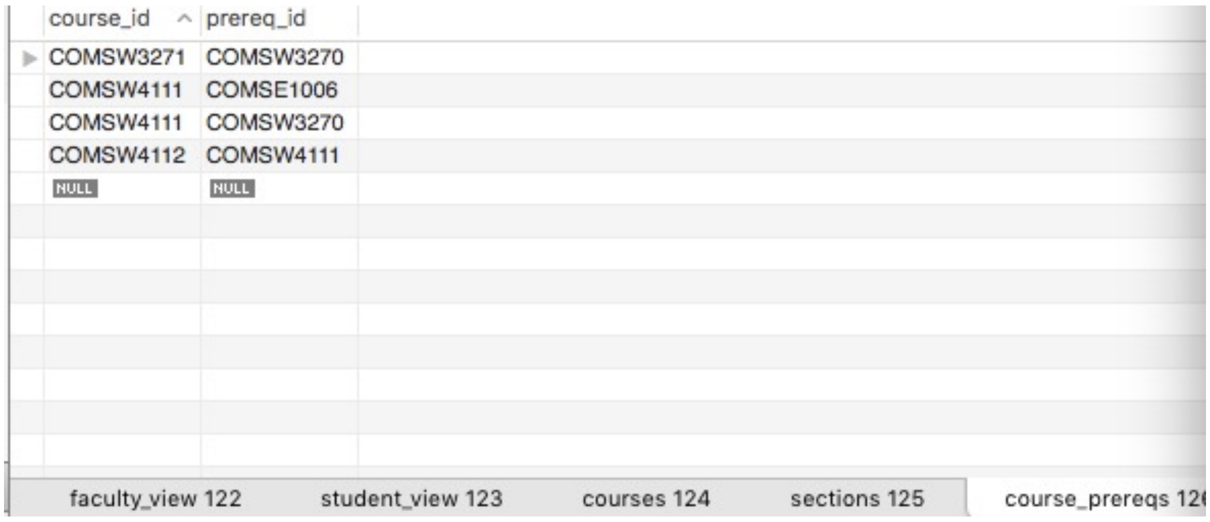
dept_code	faculty_code	level	number	title	description	course_id	full_number
COMS	E	1	006	Intro. to Program for Eng.	Darth Don teaching in Spring.	COMSE1006	1006
COMS	W	3	270	Data Structures	Seems safe to take.	COMSW3270	3270
COMS	W	3	271	Advanced Data Structures	learn more data structures	COMSW3271	3271
COMS	W	4	111	Intro. to Databases	Possibly the worst experience of your life.	COMSW4111	4111
COMS	W	4	112	Advanced Database	learn more database!	COMSW4112	4112
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figure: 5 courses tuples



call_no	course_id	section_no	year	semester	enrollment_limit	section_key
00001	COMSW4111	3	2017	1	3	20171COMSW41113
00002	COMSW4111	1	2016	4	5	20164COMSW41111
00003	COMSE1006	1	2017	1	3	20171COMSE10061
00004	COMSE1006	2	2016	2	10	20162COMSE10062
00005	COMSW3270	1	2017	1	3	20171COMSW32701
00006	COMSW3270	2	2016	3	3	20163COMSW32702
00007	COMSW3271	1	2017	1	3	20171COMSW32711
00008	COMSW3271	2	2016	2	3	20162COMSW32712
00009	COMSW4112	1	2017	1	3	20171COMSW41121
00010	COMSW4112	2	2016	1	3	20161COMSW41122
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figure: 10 section tuples
Note: the current semester is year 2017, semester 1
Therefore, I set some sections in the past



course_id	prereq_id
COMSW3271	COMSW3270
COMSW4111	COMSE1006
COMSW4111	COMSW3270
COMSW4112	COMSW4111
NULL	NULL

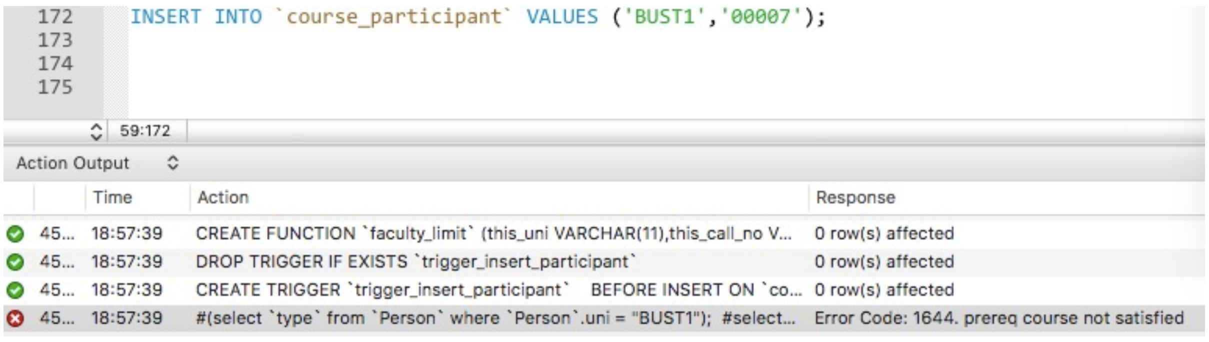
Figure: 4 prereq tuples

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.



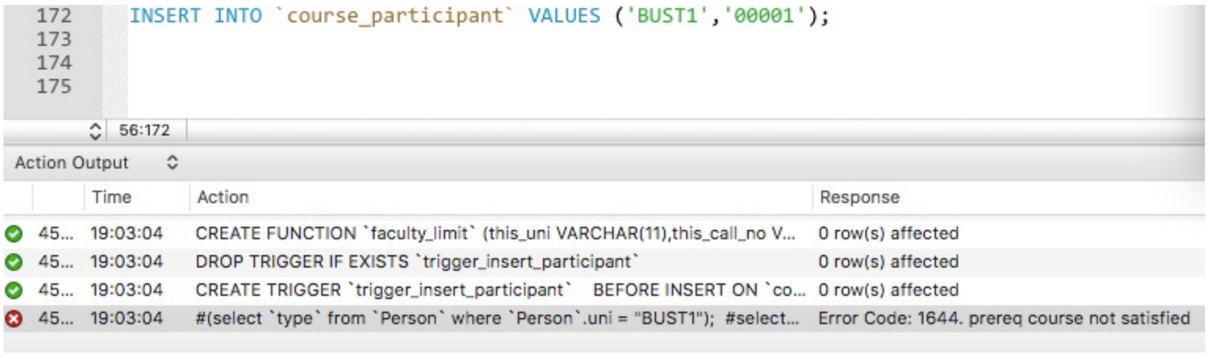
```
INSERT INTO `course_participant` VALUES ('BUST1','00007');
```

```
59:172
Action Output

   Time      Action                                                           Response
✓  45...  18:57:39  CREATE FUNCTION `faculty_limit` (this_uni VARCHAR(11),this_call_no V...  0 row(s) affected
✓  45...  18:57:39  DROP TRIGGER IF EXISTS `trigger_insert_participant`                 0 row(s) affected
✓  45...  18:57:39  CREATE TRIGGER `trigger_insert_participant`   BEFORE INSERT ON `co...  0 row(s) affected
✗  45...  18:57:39  #(select `type` from `Person` where `Person`.uni = "BUST1"); #select...  Error Code: 1644. prereq course not satisfied
```

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.

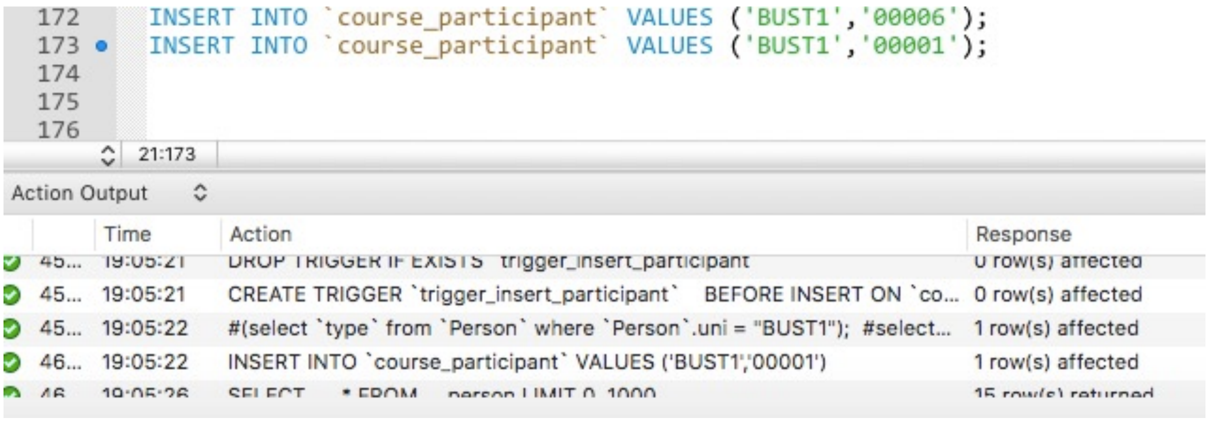


```
INSERT INTO `course_participant` VALUES ('BUST1','00001');
```

```
56:172
Action Output

   Time      Action                                                           Response
✓  45...  19:03:04  CREATE FUNCTION `faculty_limit` (this_uni VARCHAR(11),this_call_no V...  0 row(s) affected
✓  45...  19:03:04  DROP TRIGGER IF EXISTS `trigger_insert_participant`                 0 row(s) affected
✓  45...  19:03:04  CREATE TRIGGER `trigger_insert_participant`   BEFORE INSERT ON `co...  0 row(s) affected
✗  45...  19:03:04  #(select `type` from `Person` where `Person`.uni = "BUST1"); #select...  Error Code: 1644. prereq course not satisfied
```

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.



```
INSERT INTO `course_participant` VALUES ('BUST1','00006');
INSERT INTO `course_participant` VALUES ('BUST1','00001');
```

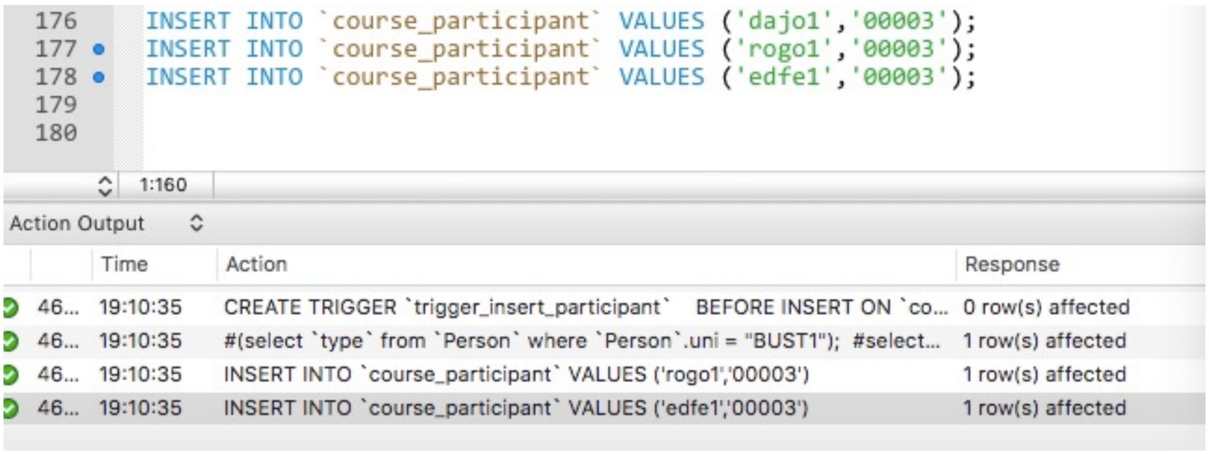
```
21:173

Action Output

     Time       Action                                                                    Response
45...  19:05:21   DROP TRIGGER IF EXISTS  trigger_insert_participant                          0 row(s) affected
45...  19:05:21   CREATE TRIGGER `trigger_insert_participant`   BEFORE INSERT ON `co...      0 row(s) affected
45...  19:05:22   #(select `type` from `Person` where `Person`.uni = "BUST1");  #select...   1 row(s) affected
46...  19:05:22   INSERT INTO `course_participant` VALUES ('BUST1','00001')                  1 row(s) affected
46...  19:05:26   SELECT     * FROM     person LIMIT 0, 1000                                15 row(s) returned
```

Figure: successful example

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

2. 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.

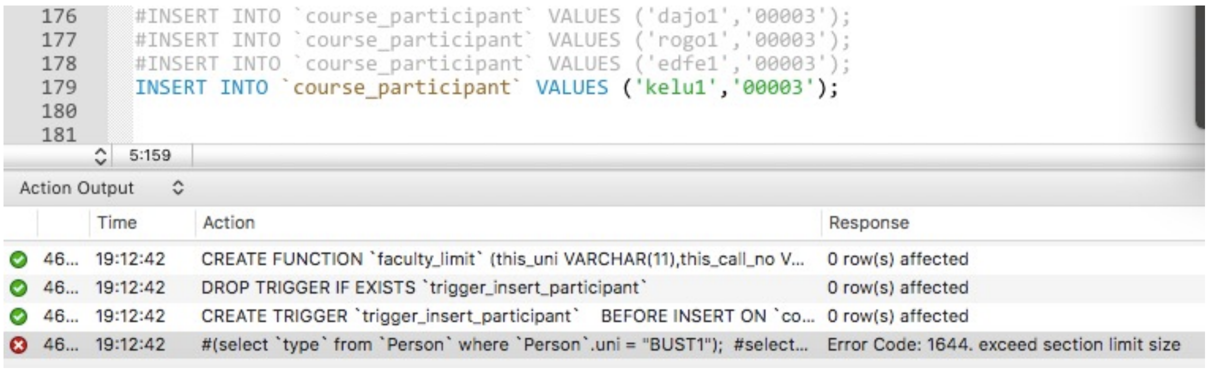


```
INSERT INTO `course_participant` VALUES ('dajo1','00003');
INSERT INTO `course_participant` VALUES ('rogo1','00003');
INSERT INTO `course_participant` VALUES ('edfe1','00003');
```

```
1:160

Action Output

     Time       Action                                                                    Response
46...  19:10:35   CREATE TRIGGER `trigger_insert_participant`   BEFORE INSERT ON `co...      0 row(s) affected
46...  19:10:35   #(select `type` from `Person` where `Person`.uni = "BUST1");  #select...   1 row(s) affected
46...  19:10:35   INSERT INTO `course_participant` VALUES ('rogo1','00003')                  1 row(s) affected
46...  19:10:35   INSERT INTO `course_participant` VALUES ('edfe1','00003')                  1 row(s) affected
```

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.



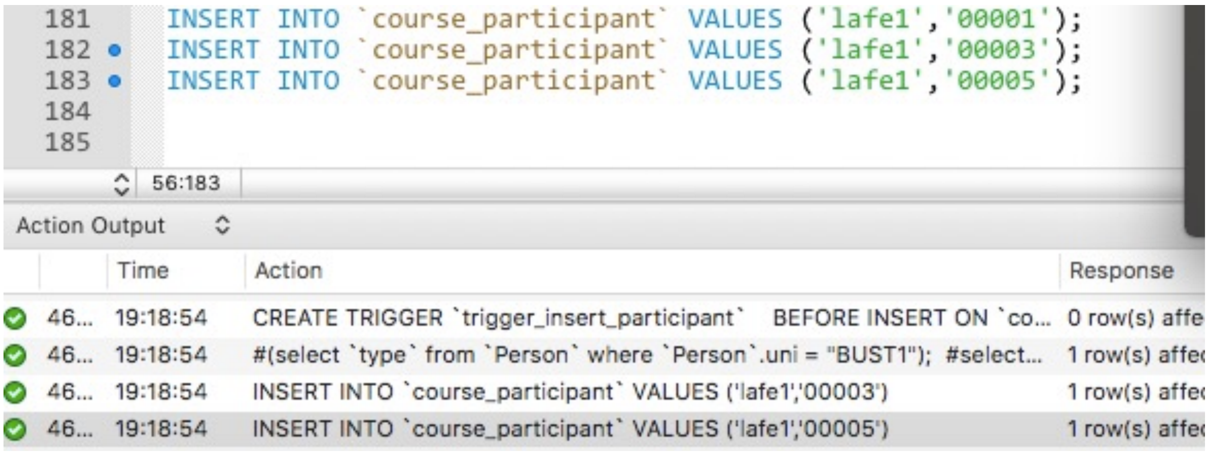
```
#INSERT INTO `course_participant` VALUES ('dajo1','00003');
#INSERT INTO `course_participant` VALUES ('rogo1','00003');
#INSERT INTO `course_participant` VALUES ('edfe1','00003');
INSERT INTO `course_participant` VALUES ('kelu1','00003');
```

```
5:159
Action Output
     Time      Action                                                                    Response
✓ 46... 19:12:42  CREATE FUNCTION `faculty_limit` (this_uni VARCHAR(11),this_call_no V...  0 row(s) affected
✓ 46... 19:12:42  DROP TRIGGER IF EXISTS `trigger_insert_participant`                      0 row(s) affected
✓ 46... 19:12:42  CREATE TRIGGER `trigger_insert_participant`  BEFORE INSERT ON `co...     0 row(s) affected
✗ 46... 19:12:42  #(select `type` from `Person` where `Person`.uni = "BUST1");  #select... Error Code: 1644. exceed section limit size
```

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.

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

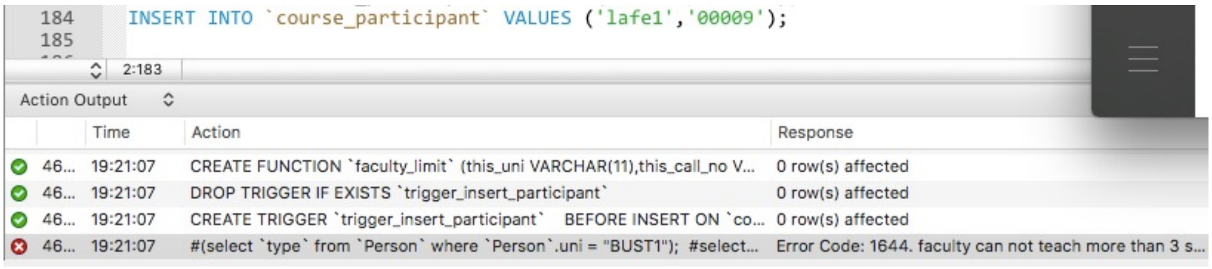


```
INSERT INTO `course_participant` VALUES ('lafe1','00001');
INSERT INTO `course_participant` VALUES ('lafe1','00003');
INSERT INTO `course_participant` VALUES ('lafe1','00005');
```

```
56:183
Action Output
     Time      Action                                                                    Response
✓ 46... 19:18:54  CREATE TRIGGER `trigger_insert_participant`  BEFORE INSERT ON `co...     0 row(s) affe
✓ 46... 19:18:54  #(select `type` from `Person` where `Person`.uni = "BUST1");  #select... 1 row(s) affe
✓ 46... 19:18:54  INSERT INTO `course_participant` VALUES ('lafe1','00003')                1 row(s) affe
✓ 46... 19:18:54  INSERT INTO `course_participant` VALUES ('lafe1','00005')                1 row(s) affe
```

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.



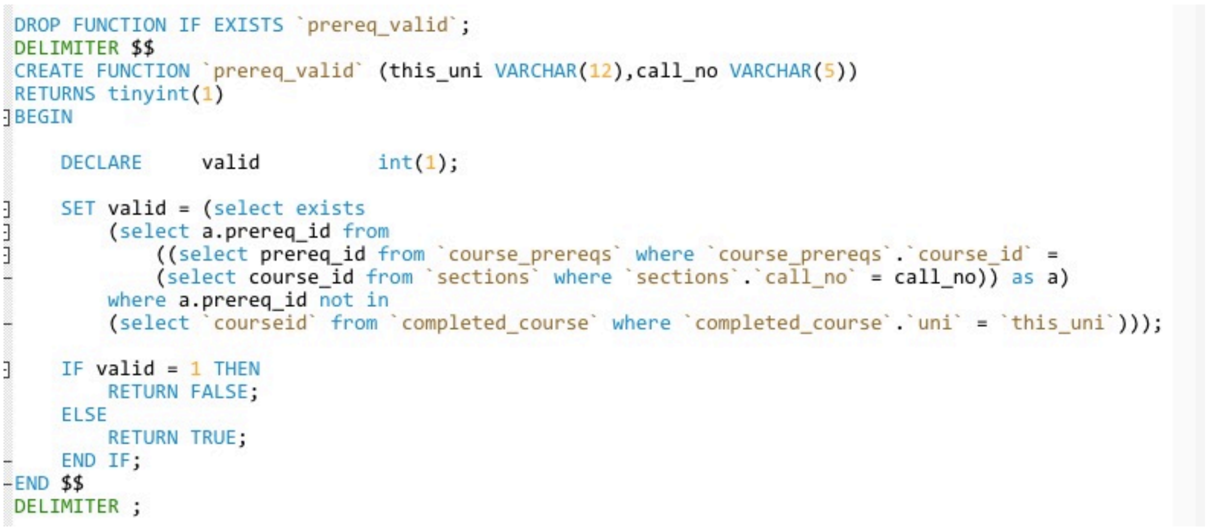
```
INSERT INTO `course_participant` VALUES ('lafe1','00009');
```

```
2:183
Action Output
   Time      Action                                                        Response
✓  46...  19:21:07  CREATE FUNCTION `faculty_limit` (this_uni VARCHAR(11),this_call_no V...  0 row(s) affected
✓  46...  19:21:07  DROP TRIGGER IF EXISTS `trigger_insert_participant`               0 row(s) affected
✓  46...  19:21:07  CREATE TRIGGER `trigger_insert_participant`  BEFORE INSERT ON `co...  0 row(s) affected
✗  46...  19:21:07  #(select `type` from `Person` where `Person`.uni = "BUST1"); #select...  Error Code: 1644. faculty can not teach more than 3 s...
```

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.

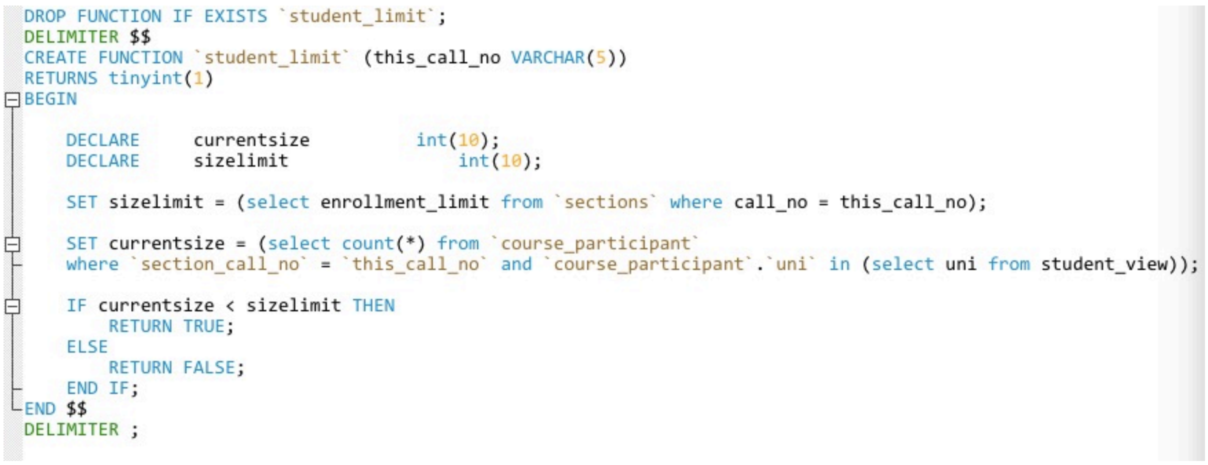
Then I would like to explain how I implement it. All the codes are in "hw3_advanced.sql"



```
DROP FUNCTION IF EXISTS `prereq_valid`;
DELIMITER $$
CREATE FUNCTION `prereq_valid` (this_uni VARCHAR(12),call_no VARCHAR(5))
RETURNS tinyint(1)
BEGIN

    DECLARE     valid          int(1);

    SET valid = (select exists
        (select a.prereq_id from
            ((select prereq_id from `course_prereqs` where `course_prereqs`.`course_id` =
            (select course_id from `sections` where `sections`.`call_no` = call_no)) as a)
        where a.prereq_id not in
        (select `courseid` from `completed_course` where `completed_course`.`uni` = `this_uni`)));

    IF valid = 1 THEN
        RETURN FALSE;
    ELSE
        RETURN TRUE;
    END IF;
END $$
DELIMITER ;
```

Figure: function for checking prereq course

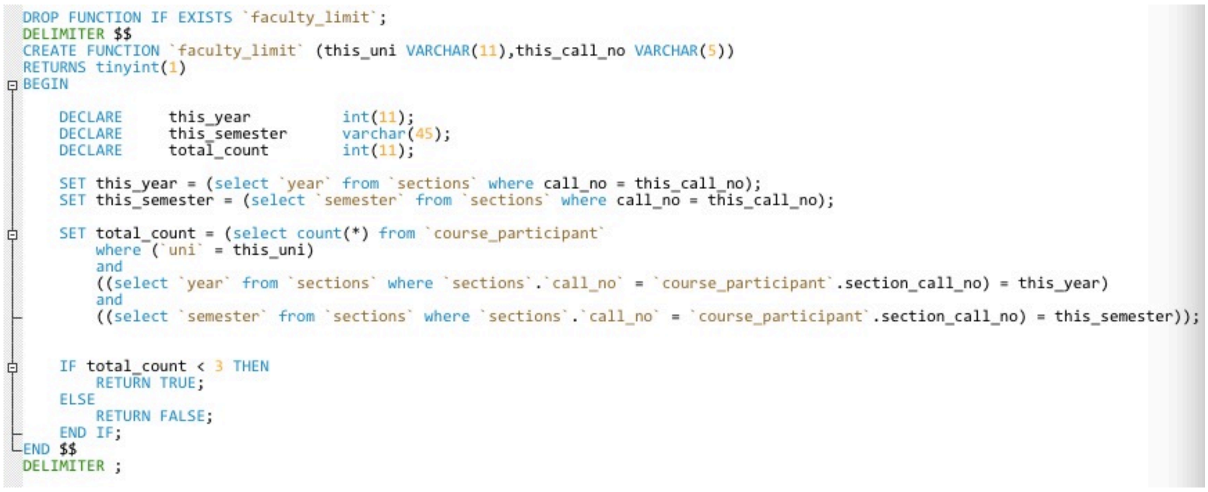
The above function tests if students completed all required prereq courses according to another section(course). It involves using a view named "completed_course" which I will explained later.



```sql
DROP FUNCTION IF EXISTS `student_limit`;
DELIMITER $$
CREATE FUNCTION `student_limit` (this_call_no VARCHAR(5))
RETURNS tinyint(1)
BEGIN

    DECLARE     currentsize         int(10);
    DECLARE     sizelimit               int(10);

    SET sizelimit = (select enrollment_limit from `sections` where call_no = this_call_no);

    SET currentsize = (select count(*) from `course_participant`
    where `section_call_no` = `this_call_no` and `course_participant`.`uni` in (select uni from student_view));

    IF currentsize < sizelimit THEN
        RETURN TRUE;
    ELSE
        RETURN FALSE;
    END IF;
END $$
DELIMITER ;
```

Figure: function for checking enrollment size limit of a section

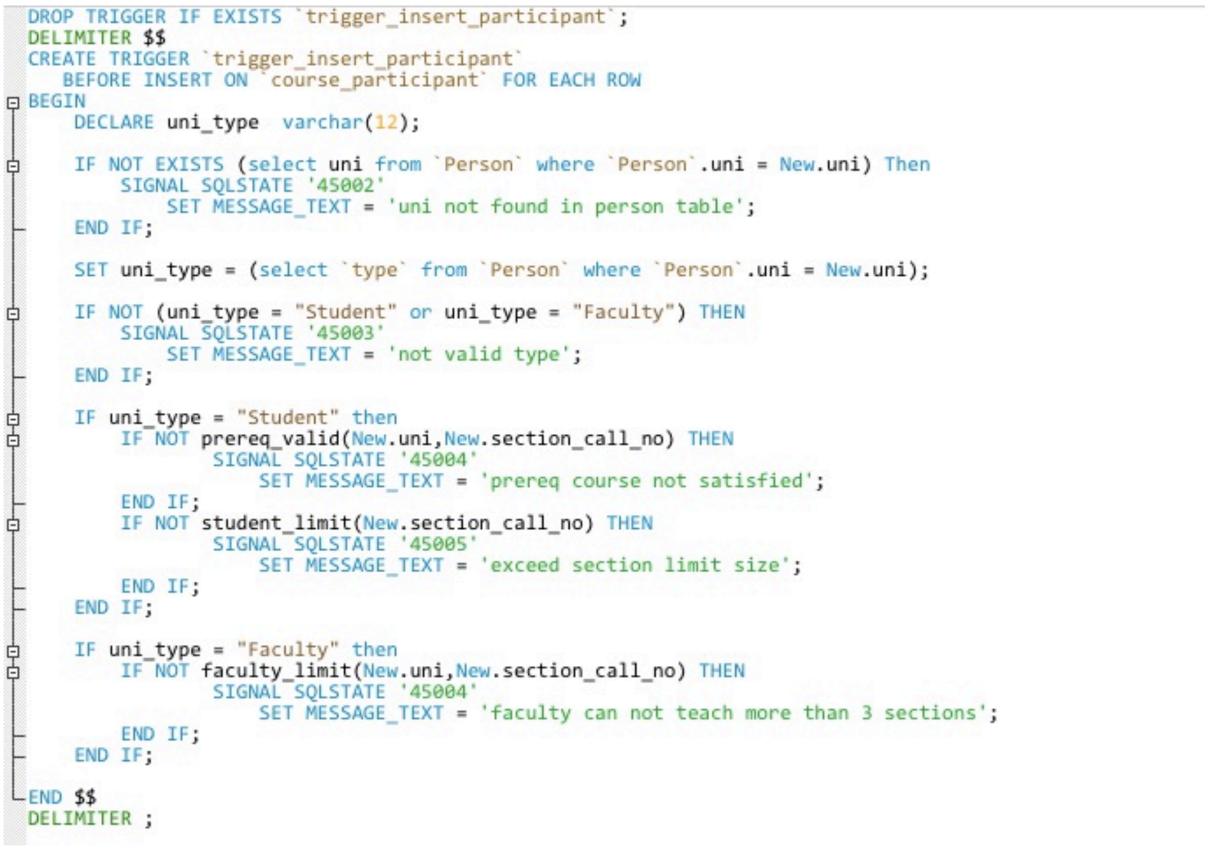
The above function tests if a given section is full or not.



```sql
DROP FUNCTION IF EXISTS `faculty_limit`;
DELIMITER $$
CREATE FUNCTION `faculty_limit` (this_uni VARCHAR(11),this_call_no VARCHAR(5))
RETURNS tinyint(1)
BEGIN

    DECLARE     this_year           int(11);
    DECLARE     this_semester       varchar(45);
    DECLARE     total_count         int(11);

    SET this_year = (select `year` from `sections` where call_no = this_call_no);
    SET this_semester = (select `semester` from `sections` where call_no = this_call_no);

    SET total_count = (select count(*) from `course_participant`
        where (`uni` = this_uni)
        and
        ((select `year` from `sections` where `sections`.`call_no` = `course_participant`.section_call_no) = this_year)
        and
        ((select `semester` from `sections` where `sections`.`call_no` = `course_participant`.section_call_no) = this_semester));


    IF total_count < 3 THEN
        RETURN TRUE;
    ELSE
        RETURN FALSE;
    END IF;
END $$
DELIMITER ;
```

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.



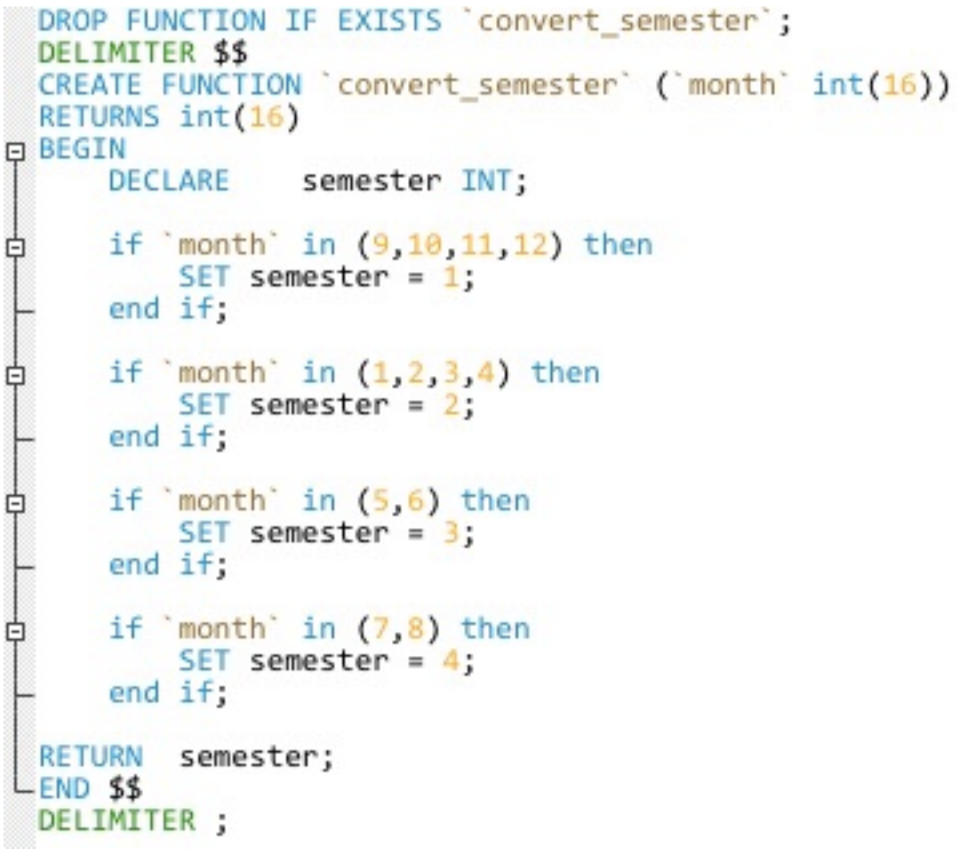
```
DROP TRIGGER IF EXISTS `trigger_insert_participant`;
DELIMITER $$
CREATE TRIGGER `trigger_insert_participant`
   BEFORE INSERT ON `course_participant` FOR EACH ROW
BEGIN
    DECLARE uni_type  varchar(12);

    IF NOT EXISTS (select uni from `Person` where `Person`.uni = New.uni) Then
        SIGNAL SQLSTATE '45002'
            SET MESSAGE_TEXT = 'uni not found in person table';
    END IF;

    SET uni_type = (select `type` from `Person` where `Person`.uni = New.uni);

    IF NOT (uni_type = "Student" or uni_type = "Faculty") THEN
        SIGNAL SQLSTATE '45003'
            SET MESSAGE_TEXT = 'not valid type';
    END IF;

    IF uni_type = "Student" then
        IF NOT prereq_valid(New.uni,New.section_call_no) THEN
                SIGNAL SQLSTATE '45004'
                    SET MESSAGE_TEXT = 'prereq course not satisfied';
        END IF;
        IF NOT student_limit(New.section_call_no) THEN
                SIGNAL SQLSTATE '45005'
                    SET MESSAGE_TEXT = 'exceed section limit size';
        END IF;
    END IF;

    IF uni_type = "Faculty" then
        IF NOT faculty_limit(New.uni,New.section_call_no) THEN
                SIGNAL SQLSTATE '45004'
                    SET MESSAGE_TEXT = 'faculty can not teach more than 3 sections';
        END IF;
    END IF;

END $$
DELIMITER ;
```

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.

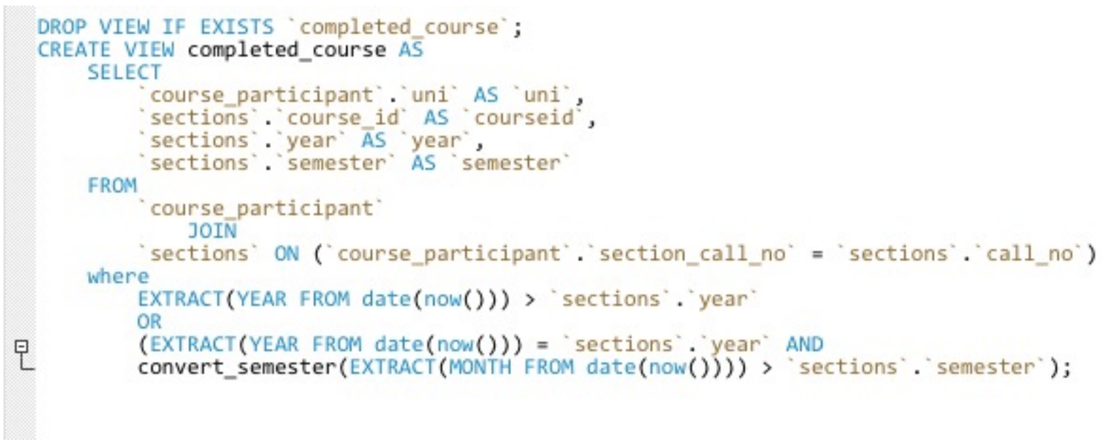
The below functions and views codes can be found in “hw3_advanced.sql”



```sql
DROP FUNCTION IF EXISTS `convert_semester`;
DELIMITER $$
CREATE FUNCTION `convert_semester` (`month` int(16))
RETURNS int(16)
BEGIN
    DECLARE    semester INT;

    if `month` in (9,10,11,12) then
        SET semester = 1;
    end if;

    if `month` in (1,2,3,4) then
        SET semester = 2;
    end if;

    if `month` in (5,6) then
        SET semester = 3;
    end if;

    if `month` in (7,8) then
        SET semester = 4;
    end if;

RETURN  semester;
END $$
DELIMITER ;
```

Figure: function convert_semester

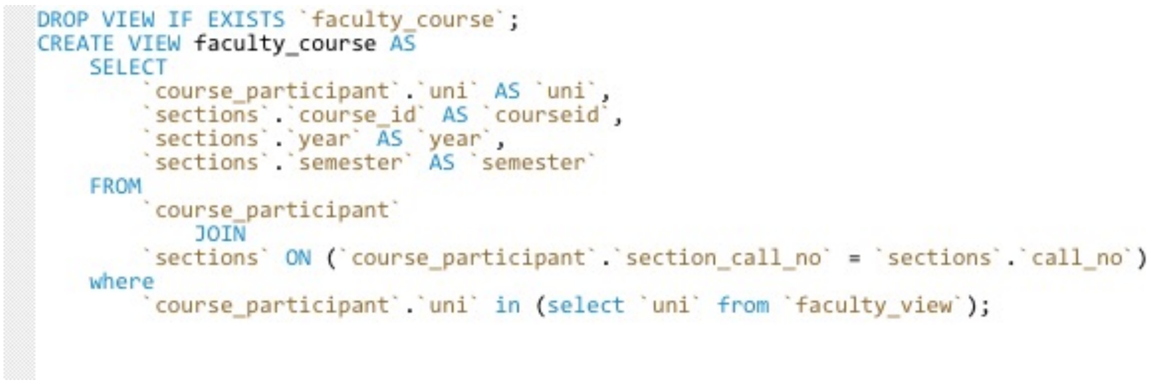
This is the function “convert_semester”. It takes a specific month and returns what semester that month corresponds to. This function is used for creating view for “completed_course”.



```sql
DROP VIEW IF EXISTS `completed_course`;
CREATE VIEW completed_course AS
    SELECT
        `course_participant`.`uni` AS `uni`,
        `sections`.`course_id` AS `courseid`,
        `sections`.`year` AS `year`,
        `sections`.`semester` AS `semester`
    FROM
        `course_participant`
            JOIN
        `sections` ON (`course_participant`.`section_call_no` = `sections`.`call_no`)
    where
        EXTRACT(YEAR FROM date(now())) > `sections`.`year`
        OR
        (EXTRACT(YEAR FROM date(now())) = `sections`.`year` AND
        convert_semester(EXTRACT(MONTH FROM date(now()))) > `sections`.`semester`);
```

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.



```
DROP VIEW IF EXISTS `faculty_course`;
CREATE VIEW faculty_course AS
    SELECT
        `course_participant`.`uni` AS `uni`,
        `sections`.`course_id` AS `courseid`,
        `sections`.`year` AS `year`,
        `sections`.`semester` AS `semester`
    FROM
        `course_participant`
            JOIN
        `sections` ON (`course_participant`.`section_call_no` = `sections`.`call_no`)
    where
        `course_participant`.`uni` in (select `uni` from `faculty_view`);
```

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.