Software Maintenance, ESOF 326, Spring 2020
Due Jan. 24, beginning of class

Use the creation script to create the database, use the population script to populate the tables with the following:

- 11 semesters - Fall 2016 to Spring 2020
- 3 programs – SE, CS, EE
- 3 subjects – CSCI, ESOF and M
  7 courses:
  - CSCI 135 Fundamentals of Computing I (associated with SE & CS)
  - CSCI 136 Fundamentals of Computing II (associated with SE & CS)
  - CSCI 246 Discrete Structures (associated with SE & CS)
  - CSCI 255 Embedded Systems (associated with SE, CS & EE)
  - ESOF 326 Software Maintenance (associated with SE & CS)
  - M 172 Calculus I (associated with SE, CS & EE)
  - ESOF 411 Verification and Validation (associated with SE)
- 5 intervals – and these are associated with the items above

Create views or stored procedures for the following. Throughout this assignment you do not need to be concerned about semester intervals.

1. List the subject, number and name of all courses in the system.

```
/**
 * all_courses view returns the subject, number and name of
 * all courses in the system.
 */
CREATE OR REPLACE VIEW all_courses AS
        SELECT subjects.text, courses.number, courses.name
        FROM COURSES LEFT JOIN subjects
                ON courses.subject_id = subjects.id;
```

Result:

text	number	name
CSCI	135	Fundamentals of Computing I
CSCI	136	Fundamentals of Computing II
CSCI	246	Discrete Structures
CSCI	255	Embedded Systems
CSCI	172	Calculus I
ESOF	326	Software Maintenance
ESOF	411	Verification and Validation

A view is like a table. To run it later can write:

```
SELECT *
FROM all_courses;
```

2. Given the id of a program, list the subject, number and name of courses associated with that program.

   This requires a stored procedure. Can't use CREATE OR REPLACE with stored procedures without using a "drop-in replacement" for MYSQL.

```
DELIMITER //

DROP PROCEDURE IF EXISTS list_courses_in_program;

/**
 * list_courses_in_program takes the id of a program
 * and returns the subject, number and name of those courses
 * that have been associated with that program.
 *    inputs: id of a program
 *    outputs: list of subject, number and name of all courses that
 *           have been associated with that program.
 */
CREATE PROCEDURE list_courses_in_program(
        IN inputted_program_id  INT
)
BEGIN
        SELECT subjects.text, courses.number, courses.name
        FROM (COURSES LEFT JOIN subjects
           ON courses.subject_id = subjects.id)
              JOIN program_courses
                 ON courses.id = program_courses.course_id
        WHERE program_courses.program_id = inputted_program_id;
END //

CALL list_courses_in_program(2);
```

3. Given the abbreviation for a program, list the subject, number and name of courses associated with that program. The datatype of programs.abbrev is CHAR(5).

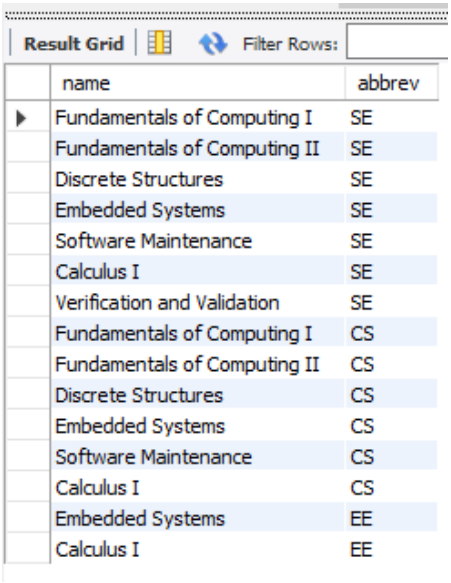
Need a stored procedure.

```
DELIMITER //

DROP PROCEDURE IF EXISTS list_courses_in_program;

/**
 * list_courses_in_program takes the abbreviation of a program
 * and returns the subject, number and name of those courses
 * that have been associated with that program.
 *    inputs: abbrev of a program
 *    outputs: list of subject, number and name of all courses that
 *           have been associated with that program.
 */
CREATE PROCEDURE list_courses_in_program(
        IN inputted_program_abbrev  CHAR(5)
)
BEGIN
        SELECT subjects.text, courses.number, courses.name
        FROM ((COURSES LEFT JOIN subjects
           ON courses.subject_id = subjects.id)
              JOIN program_courses
                 ON courses.id = program_courses.course_id)
                    JOIN programs
                       ON program_courses.program_id = programs.id
        WHERE programs.abbrev = inputted_program_abbrev;
END //

CALL list_courses_in_program('SE');          /* Watch the tick marks! */
```

4. For all course/program associations in the system, list the name of the course, along with the abbreviation of the program that the course is associated with.

```
/**
 * all_course_program_associations view returns the name of the course
 * and abbreviation of a program that the course is associated with.
 */
CREATE OR REPLACE VIEW all_course_program_associations AS
        SELECT courses.name, programs.abbrev
        FROM (program_courses JOIN courses
                ON program_courses.course_id = courses.id)
            JOIN programs
                ON program_courses.program_id = programs.id;

SELECT *
FROM all_course_program_associations;
```

Result:



Result Grid | Filter Rows:

name	abbrev
Fundamentals of Computing I	SE
Fundamentals of Computing II	SE
Discrete Structures	SE
Embedded Systems	SE
Software Maintenance	SE
Calculus I	SE
Verification and Validation	SE
Fundamentals of Computing I	CS
Fundamentals of Computing II	CS
Discrete Structures	CS
Embedded Systems	CS
Software Maintenance	CS
Calculus I	CS
Embedded Systems	EE
Calculus I	EE

5. Generate the next semester. Semester names are of the format Fall 2019, Spring 2020 and Summer 2020. The next semester to be generated would be Fall 2020, then Spring 2021, etc. The datatype of semesters.name is CHAR(11).

   This requires a stored procedure:

```
DELIMITER //

DROP PROCEDURE IF EXISTS generate_next_semester;

/**
 * generate_next_semester finds the most recent semester in
* the system, and generates the next semester, using the
* format and order indicated by the following: Fall 2019,
* Spring 2020, Summer 2020, Fall 2020, Spring 2021, etc.
 *    inputs: none
 *    outputs: id of most recent semester
 *     side effects: the next semester has been added to the
*              existing list of semesters.
 */
CREATE PROCEDURE generate_next_semester()
BEGIN
        DECLARE most_recent_semester CHAR(11);
        DECLARE new_year INT;

SELECT semesters.name
INTO most_recent_semester
FROM semesters
WHERE id =
        (SELECT MAX(id)
        FROM semesters);

IF SUBSTRING(most_recent_semester, 1, 6) = 'Spring' THEN
        INSERT INTO semesters (name)
        VALUES
        (CONCAT('Summer ', SUBSTRING(most_recent_semester, 8,4)));
ELSEIF SUBSTRING(most_recent_semester, 1, 6) = 'Summer' THEN
        INSERT INTO semesters (name)
        VALUES
        (CONCAT('Fall ', SUBSTRING(most_recent_semester, 8,4)));
ELSEIf SUBSTRING(most_recent_semester, 1, 4) = 'Fall' THEN
        SET new_year = 1 +
        CAST(SUBSTRING(most_recent_semester, 6,4) AS UNSIGNED);
        INSERT INTO semesters (name)
        VALUES
        (CONCAT('Spring ', CAST(new_year AS CHAR(4))));
END IF;
```

```
END //

CALL generate_next_semester();
```

Email your solutions, along with sample output, to me.