Instructions

Read these instructions all the way through before you start.

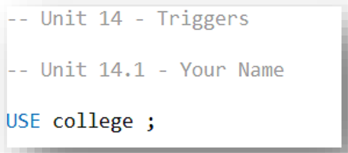
Download the answer document. Be sure to put your name at the top of the answer document where that's indicated. Answer the questions below in that document and when you are done, upload it into Blackboard.

Overview

In this unit you'll get a feel on how to create and use stored procedures and user functions.

Preparing your SQL

Make sure you include the comment and USE statement at the top of your SQL



```
-- Unit 14 - Triggers

-- Unit 14.1 - Your Name

USE college ;
```

and don't forget to beautify your SQL, by hand if needed. In your screen shot of the Result Grid you need to show only first 6-10 rows. If you're not sure about these, see Unit 02 instruction for details on how to do that.

Steps

Note: Since these steps will make changes in the College database, you'll want to run CollegeCreate.sql after each test to restore the database.

1. [5] Write an **insert** trigger than insures that the email of people added to the faculty table is in the standard form of FirstLast@college.edu, regardless of what value might be used in the insert statement. Create a stored procedure named **Faculty_Before_Insert**. Then execute an INSERT statement to put yourself into the faculty table. Use bad@bad.bad as your email. Be sure to show your INSERT statement in the answer document.

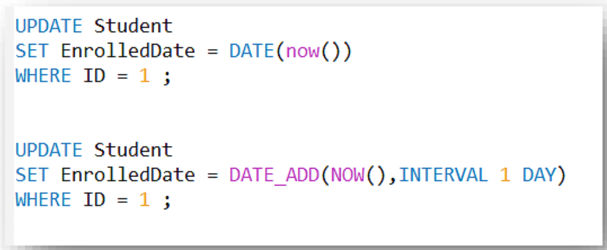
Then run this SELECT statement to show the newly added record.

```
SELECT
    LastName, FirstName, Email
FROM
    Faculty
```

```
WHERE
    ID = LAST_INSERT_ID();
```

2. [10] For this step, write a **before update** trigger that prevents an update to the student table if the EnrolledDate is in the future. Name the trigger **Student_Before_Update**. If the date is in the future, create a SQL signal and error with the message: “Enrolled date may not be in the future.”

After you’ve created the trigger, run this SQL



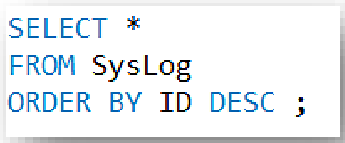
```
UPDATE Student
SET EnrolledDate = DATE(now())
WHERE ID = 1 ;


UPDATE Student
SET EnrolledDate = DATE_ADD(NOW(),INTERVAL 1 DAY)
WHERE ID = 1 ;
```

3. [5] Go back to your **Student_Before_Update** and add another rule of your choosing for it to enforce. Make sure your rule makes sense in the real world and that it produces a helpful error message. Write an update statement that causes your rule to display your error message.

4. [15] For this step, write an **after update** trigger that creates a record in a new table named SysLog whenever a record in the student table is changed. Name the trigger **Student_After_Update**. Use the **CreateSysLog.sql** file included with the assignment to create the new table.

In your trigger, create a log record with table of ‘Student’ and a message that looks something like this “Updated ID=X” where X is the ID of the student record that’s been updated.
After you’ve created the trigger, create and run update statements on several different student records, then run this SQL:



```
SELECT *
FROM SysLog
ORDER BY ID DESC ;
```

5. [15] Create a **before insert** trigger on the Registration table named **Registration_Before_Insert**. Have the trigger prevent a student from registering for a section that the student is already

registered for[1]. Write an INSERT statement that successfully creates a registration record then run the same INSERT statement a second time. Your trigger should reject the second INSERT.

[1] This could be accomplished by a unique index on the registration table, but what fun would that be?