Problem Set #2

CSCI 3287

Modify this cell and put your Name and email

Name: Nicolas Mavromatis

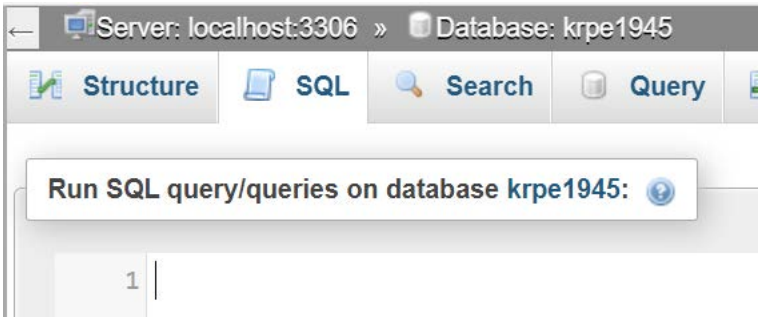
Working in collaboration with Steve Putt

Email: Nima6629@colorado.edu

Instructions / Notes:

Read these carefully

- Open your MySQL environment and go to your SQL tab.



- In this Word document you will enter your solution between the *### BEGIN SOLUTION* and *### END SOLUTION*

Submission Instructions:

- Save as pdf for submission.

Have fun!

Constraints and Triggers Review

Constraints and triggers are tools to impose restrictions on allowable data within a database, beyond the requirements imposed by table definition types.

Constraints, also known as *integrity constraints*, are used to constrain allowable database states. They prevent disallowed values from being entered into the database.

- non-null constraints
  - create Table MyTable(myValuedataType NOT NULL);
- key or uniqueness constraints
  - create Table MyTable(myId int PRIMARY KEY);
  - create Table MyTable(myValue1 dataType, myValue2 dataType, UNIQUE(myValue1,myValue2));
- attribute restrictions
  - create Table MyTable(myValuedataType check(myValue> 0))
- referential integrity (a.k.a. foreign keys)
  - create Table MyTable(otherId int, foreign key(otherId) references OtherTable(otherColumn))

Triggers are procedures that get run when specified events in a database view or table occur. They are useful for implementing monitoring logic at the database level.

- delete/update/insert
- before/after/instead of
- when(condition)
- row-level/statement level

Question 1 - Constraints [10 pts]

Write CREATE TABLE declarations with the necessary constraints for the following 4 tables and their specifications:

- Student(sID, name, parentEmail, gpa)
  - sID (should be unique)
  - name (should exist)
  - parentEmail(should exist)
  - gpa (real value between 0 and 4 inclusive)
- Class(cID, name, units)
  - cID (should be unique)
  - name (should exist)
  - units (must be between 1 and 5 inclusive)
- ClassGrade(sID, cID, grade)
  - sID (should reference a student)
  - cID (should reference a class)
  - grade (integer between 0 and 4 inclusive, for F,D,C,B,A)
  - student can only get 1 grade for each class
- ParentNotification(parentEmail, text)
  - parentEmail (should exist)
  - text (the message body, should exist)

Constraints, such as the value for grade, **must** use check to check that constraint.

Write your table definitions here. **Format your definitions so they are readable.**

BEGIN SOLUTION 1

```
CREATE TABLE Student(
   sID int PRIMARY KEY,
   name VARCHAR(50) NOT NULL,
   parentEmail VARCHAR(50) NOT NULL,
   gpa float(10,10) check (0<=gpa<=4)
);


CREATE TABLE Class(
   cID int  PRIMARY KEY,
   name VARCHAR(50) NOT NULL,
   units int check (1<=units<=5)
);
--NOTE: You could also do PRIMARY KEY(cID, sID, grade) if you wanted to make sure one grade
--per student per class exists, but either implementation ensures there is one student per class.
CREATE TABLE ClassGrade
(
   sID int,
   cID int,
   FOREIGN KEY(sID)  REFERENCES Student(sID),
   FOREIGN KEY(cID)  REFERENCES Class(cID),
   grade int check (0<=grade<=4),
   PRIMARY KEY(cID, sID)
);
CREATE TABLE ParentNotification
(
   ParentEmail varchar(50) NOT NULL,
   text varchar(100) NOT NULL
);
```

END SOLUTION

Question 2 - Triggers Introduction [20 pts in two parts of 10pts each]

Triggers are used to execute sql commands upon changes to the specified tables.
Documentation on Trigger support in SQLite can be found here.

The following is an example of a trigger.

```
droptable if existsEmployee;
droptable if existsDepartment;
droptrigger if existsupdate_employee_count;
createtableEmployee(eIDint, nametext, dID int);
createtableDepartment(dIDint, nametext, employee_countint);

createtriggerupdate_employee_count
afterinsertonEmployee
for eachrow
begin
updateDepartmentsetemployee_count=employee_count+1where
dID=new.dID;
end;
```

Note that there is a difference between OLD values and NEW values in triggers that execute on statements that change values in a table. Both the WHEN clause and the trigger actions may access elements of the row being inserted, deleted or updated using references of the form "NEW.column-name" and "OLD.column-name", where column-name is the name of a column from the table that the trigger is associated with. Triggers on INSERT statements (like that above) can only access the NEW values (since OLD values don't exist!) and triggers on DELETE statements can only access OLD values.

Let's continue by adding data to the tables.

```
insertintoDepartmentvalues(1,'HR',0);
insertintoDepartmentvalues(2,'Engineering',0);
```

At this point, there are no empoloyees in the Employee table. As you can see below, each department has 0 employees.

```
selectname, employee_count
from department;
```

[8]:

name	employee_count
HR	0
Engineering	0

When we insert several employees into the Employee table, the trigger should fire and update values in the Department table.

```
insertintoEmployeevalues
(1,'Todd',1),(2,'Jimmy',1),(3,'Billy',2);
```

Now when we view the employee table, we see that the employee count has been updated by the trigger.

```
selectname, employee_count
from department;
```

name	employee_count
HR	2
Engineering	1

Now, it's your turn! Write a SQL trigger on the ClassGrade table you defined earlier. On each insertion into the ClassGrade table, the trigger should update the GPA of the corresponding student.

- gpa = sum(units*grade)/sum(units)

First, let's load data into the tables:

```
insertintoStudentvalues(1,'Timmy','timmysmom@gmail.com', 0.0);
insertintoStudentvalues(2,'Billy','billysmom@gmail.com',0.0);
insertintoClassvalues(1, 'CS3287',4);
insertintoClassvalues(2,'CS4122',3);

select*fromstudent;
```

sId	name	parentEmail	gpa
1	Timmy	timmysmom@gmail.com	0.0
2	Billy	billysmom@gmail.com	0.0

Now, write your trigger here [10pts]

BEGIN SOLUTION 2a

```
DELIMITER //
CREATE TRIGGER gpaTrig
AFTER INSERT ON ClassGrade
FOR EACH ROW
BEGIN
   UPDATE Student SET gpa =
      (
      SELECT (SUM(CAST(Class.units AS FLOAT) * ClassGrade.grade) / SUM(Class.units)) as g
      FROM Class, ClassGrade
      WHERE Class.cID = ClassGrade.cID
      AND ClassGrade.sID = NEW.sID GROUP BY ClassGrade.sID
      );
END;//
```

END SOLUTION

Now, write a second trigger here that inserts a row in ParentNotification with the parent's email and a message. The trigger should execute whenever a Student record is updated with a new GPA and that GPA is < 2.0.

A trigger like this can have a format similar to the following in SQL:

```
create trigger XYZ

   after update of myColumn on myTable

   for each row when (condition in myTable)

   begin

      insert/update/delete etc.

   end
```

You may want to look at the SQLite operators page to see how to do string concatenation.

Write your trigger here: [10pts]

BEGIN SOLUTION 2b

```
DELIMITER //
CREATE TRIGGER fail
AFTER UPDATE ON Student
FOR EACH ROW
BEGIN
	IF NEW.gpa<2.0 THEN
	INSERT INTO ParentNotification
		Values(Student.parentEmail, concat('Your son ', Student.name, ' is failing school.'));
	END IF;
END;//
```

END SOLUTION

We can now test the triggers.

```
insertintoClassGradevalues(1,1,2);
insertintoClassGradevalues(1,2,1);
insertintoClassGradevalues(2,1,1);
select*fromParentNotification;
```

parentEmail	message
timmysmom@gmail.com	Your son Timmy is failing school.
billysmom@gmail.com	Your son Billy is failing school.

Question 3 - Advanced Triggers [20 pts in one part]

Triggers can execute BEFORE, AFTER, or INSTEAD OF the sql statements that trigger them. SQLite notes that programmers should be very wary when executing BEFORE or INSTEAD OF triggers.

If a BEFORE UPDATE or BEFORE DELETE trigger modifies or deletes a row that was to have been updated or deleted, then the result of the subsequent update or delete operation is undefined. Furthermore, if a BEFORE trigger modifies or deletes a row, then it is undefined whether or not AFTER triggers that would have otherwise run on those rows will in fact run.

The value of NEW.rowid is undefined in a BEFORE INSERT trigger in which the rowid is not explicitly set to an integer.

Because of the behaviors described above, programmers are encouraged to prefer AFTER triggers over BEFORE triggers.

Triggers are one of the unfortunate areas where SQL implementations differ greatly. The correct semantics for a row-level "after" trigger, according to the SQL standard, is to activate the trigger after the entire triggering data modification statement completes, executing the trigger once for each modified row. PostgreSQL implements these semantics as does MySQL. SQLite instead implements semantics where the trigger is activated immediately after each row-level change, interleaving trigger execution with execution of the modification statement.

Finally, SQLite supports the RAISE() function. The function can be used to halt the execution of a trigger and the statement that caused it. Here's an example that would prevent students from getting a grade in CS 5817 until they've gotten a B or better in CS 3287.

```
droptrigger if existsenforce_cs5817_prereqs;

insertintoClassvalues (3,'CS5817',3);
insertintoStudentvalues (3,'Johnny', 'johnnysmom@gmail.com', 0.0);
insertintoClassGradevalues (3,1,4);

createtriggerenforce_cs5817_prereqs
beforeinsertonClassGrade
foreachrow
whenexists (
```

```
Select*
fromClassc1
wherec1.cID=new.cID
andc1.name='CS5817'
andnew.sIDnotin (
Selectcg.sID
fromclassc2, ClassGradecg
wherec2.cID=cg.cID
andc2.name='CS3287'
andcg.grade>2)
   )
begin
selectraise(rollback, 'A student must pass CS 3287 before taking CS 5817');
end;
```

With our trigger, student number 3, Johnny, should be able to take CS 5817 since he got an A in CS 3287.

```
   insertintoClassGradevalues (3,3,4.0);

selectStudent.name, Student.sID, Class.name, Class.cID, ClassGrade.grade
fromClass, ClassGrade,Student
WHEREClass.cID=ClassGrade.cID
ANDStudent.sID=ClassGrade.sID
```

name	sId	name_1	cId	grade
Timmy	1	CS3287	1	2
Timmy	1	CS4122	2	1
Billy	2	CS3287	1	1
Johnny	3	CS3287	1	4
Johnny	3	CS5817	3	4
Johnny	3	CS4122	2	4

As you can see, Johnny had no trouble getting a grade in the class. Now, if we try to enter a grade for Student 1, it should fail due to our trigger. It will present a rollback message if the trigger executes.

```
insertintoClassGradevalues (1,3,4.0);
```

You should have received an error.

Now, it's your turn! Write a trigger that prevents a student from getting a grade in any class when there are pending emails in the ParentNotification table for that student's parent.

Write your solution here [20 pts]

BEGIN SOLUTION 3

```
DELIMITER //
CREATE TRIGGER pending
BEFORE INSERT ON ClassGrade
FOR EACH ROW
WHERE ClassGrade.sID IN
            (SELECT DISTINCT sID
            FROM Students, ParentNotification
            WHERE ParentNotification.parentEmail = Students.parentEmail
            AND ParentNotification.ParentEmail IS NOT NULL  )
BEGIN
   RAISE(ROLLBACK, 'You need to fix your pending grade warnings before your grade can be recorded. ');
END//;
```

END SOLUTION

Assuming your trigger is correct, this statement should succeed (note that it can only be executed once)

```
insertintoClassGradevalues (3,2,4);

select*fromClassGrade
```

sId	cId	grade
1	1	2
1	2	1
2	1	1
3	1	4
3	3	4
3	2	4

And this one should fail.

```
insertintoClassGradevalues(2,2,1);
```