Constraints and Triggers Activity

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(myValue dataType 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(myValue dataType 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

Activity 1 - Constraints

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)`

In [ ]:

```
%load_ext sql
%sql sqlite://
```

Write your table definitions here:

In [ ]:

```
%%sql
create table Student(sId int PRIMARY KEY, name text NOT NULL, parentEmail text NOT N
create table Class(cId int PRIMARY KEY, name text NOT NULL, units int check(units >
create table ClassGrade(sId int, cId int, grade int check(grade >= 0  and grade <= 4
create table ParentNotification(parentEmail text not null, message text not null);
```

Activity 2 - Triggers Introduction

Triggers are used to execute sql commands upon changes to the specified tables. Trigger support in SQLite can be found here (https://www.sqlite.org/lang_createtrigger.html).

The following is an example of a trigger in SQLite.

In [ ]:

```
%%sql
drop table if exists Employee;
drop table if exists Department;
drop trigger if exists update_employee_count;
create table Employee(eID int, name text, dID);
create table Department(dID int, name text, employee_count int);
```

In [ ]:

```
%%sql
create trigger update_employee_count
after insert on Employee
for each row
begin
  update Department set employee_count = employee_count + 1 where
  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.

In [ ]:

```
%%sql
insert into Department values(1,'HR',0);
insert into Department values(2,'Engineering',0);
```

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

In [ ]:

```
%%sql
select name, employee_count
from department;
```

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

In [ ]:

```
%%sql
insert into Employee values
(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.

In [ ]:

```
%%sql
select name, employee_count
from department;
```

Now, it's your turn! Write a SQLite 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:

In [ ]:

```
%%sql
insert into Student values(1,'Timmy','timmysmom@gmail.com', 0.0);
insert into Student values(2,'Billy','billysmom@gmail.com',0.0);
insert into Class values(1, 'CS145',4);
insert into Class values(2,'CS229',3);
```

In [ ]:

```
%%sql
select * from student;
```

Now, write your trigger here:

In [ ]:

```
%%sql
create trigger update_gpa
  after insert on ClassGrade
  for each row
  begin
    update Student set gpa = (select sum(grade * units)*1.0/sum(units) from ClassGra
                                          ClassGrade.cId = Class.cId and ClassGrade
    where sId = NEW.sId;
  end;
```

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 SQLite:

```
create trigger XYZ
    after update of myColumn on myTable
    for each row when (condition in myTable)
    begin
        insert/update/delete etc.
     end
```

Write your trigger here:

In [ ]:

```
%%sql
create trigger notify_parent
  after update of gpa on Student
  for each row when (NEW.gpa < 2.0)
  begin
    insert into ParentNotification values(NEW.parentEmail, 'Your son '||NEW.name||'
  end;
```

We can now test the triggers.

In [ ]:

```
%%sql
insert into ClassGrade values(1,1,2);
insert into ClassGrade values(1,2,1);
insert into ClassGrade values(2,1,1);
select * from ParentNotification;
```

Activity 3 - Advanced Triggers

Triggers can execute BEFORE, AFTER, or INSTEAD OF the sql statements that trigger them. SQLite notes (https://www.sqlite.org/lang_createtrigger.html) 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. 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 245 until they've gotten a B or better in CS 145.

In [ ]:

```
%%sql
drop trigger if exists enforce_cs245_prereqs;

insert into Class values (3,'CS245',3);
insert into Student values (3,'Johnny', 'johnnysmom@gmail.com', 0.0);
insert into ClassGrade values (3,1,4);

create trigger enforce_cs245_prereqs
before insert on ClassGrade
for each row
when exists (
        Select *
        from Class c1
        where c1.cID = new.cID
        and c1.name = 'CS245'
        and new.sID not in (
            Select cg.sID
            from class c2, ClassGrade cg
            where c2.cID = cg.cID
            and c2.name = 'CS145'
            and cg.grade > 2)
    )
begin
  select raise(rollback, 'A student must pass CS 145 before taking CS 245');
end;
```

With our trigger, student number 3, Johnny, should be able to take CS 245 since he got an A in CS 145.

In [ ]:

```
%%sql
insert into ClassGrade values (3,3,4.0);
```

In [ ]:

```
%sql select * from ClassGrade;
```

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.

In [ ]:

```
%%sql
insert into ClassGrade values (1,3,4.0);
```

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.

In [ ]:

```
%%sql
drop trigger if exists no_enrollment_with_parent_email;
create trigger no_enrollment_with_parent_email
before insert on ClassGrade
for each row
when exists (
        Select *
        from ParentNotification pe
        where new.sID in (
            select s.sID
            from Student s, ParentNotification pn
            where pn.parentEmail = s.parentEmail)
    )
begin
  select raise(rollback, 'A student cannot enroll in classes when there are outstan
end;
```

Assuming your trigger is correct, this statement should succeed.

In [ ]:

```
%%sql
insert into ClassGrade values (3,2,4);
```

And this one shoud fail.

In [ ]:

```
%%sql
insert into ClassGrade values(2,2,1);
```

Further Information

At this point, you should have all the knowledge you need to do Project Part 2. If you want to learn more about constraints and triggers, you can view the videos from Professor Widom below. Please note that this information is **NOT** necessary for any exams or homeworks that you will have in this class. You do not need it to do the constraints or triggers on the project.