Triggers Social-Network Exercises

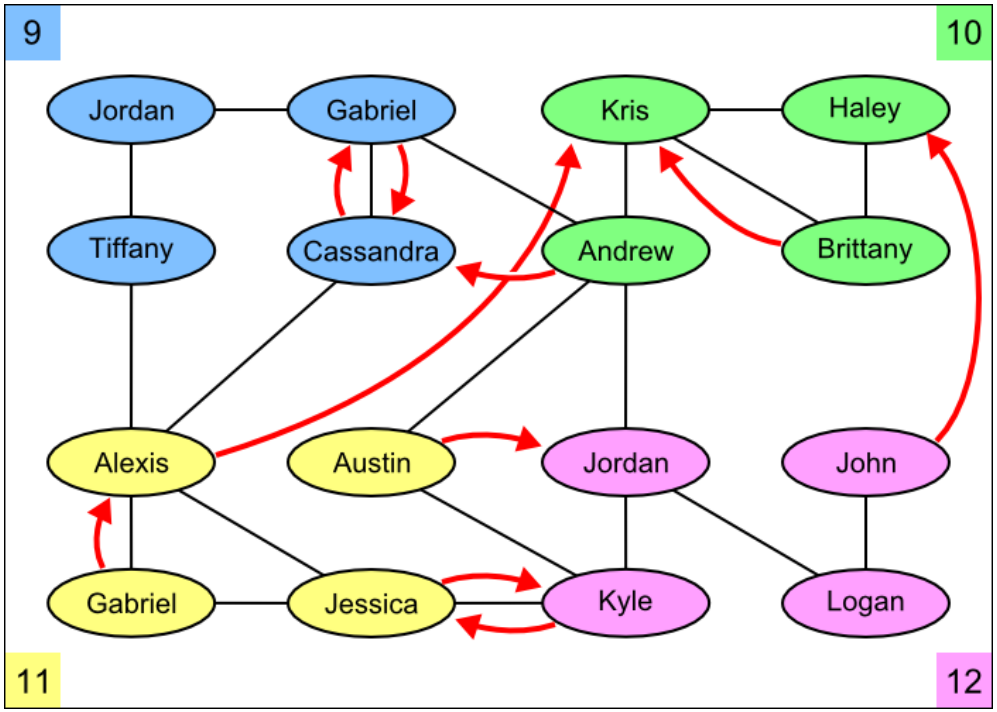
You will enhance the social-network database that was also used for the SQL Social-Network Query Exercises. In this set of exercises you will create triggers that add various behaviors to the data, and you will verify that the triggers are enforcing the desired behavior. You will implement similar triggers to those used in the SQL Social-Network Triggers Exercises, but these exercises explore in more depth the interaction among multiple triggers, and trigger behavior with and without recursive triggering enabled. A SQL file to set up the schema and data for the social-network database is downloadable here. The schema and data can be loaded as specified in the file into SQLite, MySQL, or PostgreSQL. However, currently only SQLite and PostgreSQL provide a rich enough trigger language for these exercises. Furthermore since the current mechanisms for creating triggers in PostgreSQL are quite cumbersome, we recommend SQLite. See our quick guide for installing and using all three systems.

Schema:
Highschooler (ID, name, grade)
English: There is a high school student with unique *ID* and a given *first name* in a certain *grade*.

Friend (ID1, ID2)
English: The student with *ID1* is friends with the student with *ID2*. Friendship is mutual, so if (123, 456) is in the *Friend* table, so is (456, 123).

Likes (ID1, ID2)
English: The student with *ID1* likes the student with *ID2*. Liking someone is not necessarily mutual, so if (123, 456) is in the *Likes* table, there is no guarantee that (456, 123) is also present.

For your convenience, here is a graph showing the various connections between the people in our database. 9th graders are blue, 10th graders are green, 11th graders are yellow, and 12th graders are purple. Undirected black edges indicate friendships, and directed red edges indicate that one person likes another person.



After each exercise to create a trigger or set of triggers, we include one or more data modification statements that should activate the triggers, followed by one or more queries to check that the final database state is correct. The query results over the correct final database for each exercise can be viewed by pressing the button at the bottom of the page.

Although the exercises are presented as a sequence, not all of them depend on each other. Specifically:
-- Exercises 1-2 can be worked together from the original database, independently of the other exercises. If performed correctly, these two exercises leave the database in its original state.
-- Exercise 3 can be worked from the original database, independently of the other exercises. If performed correctly, the command specified at the end of the exercise will return the database to its original state.
-- Exercises 4-6 can be worked together from the original database, independently of the other exercises. These three exercises do not leave the database in its original state.

1. Write triggers to maintain symmetry in friend relationships: If (A,B) is in *Friend* then (B,A) should be too. The initial database obeys this constraint; your triggers should monitor insertions, deletions, and updates and perform corresponding modifications to maintain symmetry. *Make sure not to create duplicate tuples in Friend.*

Begin with recursive triggering disabled ("pragma recursive_triggers = off;" in SQLite). Run the following statements, which remove the friendship between Brittany and Kris, add a friendship between Brittany and Andrew, and change Jessica to be friends with Austin and Andrew instead of Alexis and Kyle.

```
pragma recursive_triggers=off;
delete from Friend where ID1 = 1641 and ID2 = 1468;
insert into Friend values (1641,1782);
update Friend set ID2 = 1316 where ID1 = 1501 and ID2 = 1247;
update Friend set ID2 = 1782 where ID1 = 1501 and ID2 = 1934;
```

Check the resulting database by computing the number of tuples in the *Friend* table. Compare your answer against ours.

2. Continuing with the previous problem, now make sure your solution also works with recursive triggering enabled ("pragma recursive_triggers = on;"). Run the following statements, which undo the previous changes -- remove the friendship between Brittany and Andrew, add a friendship between Brittany and Kris, and change Jessica to be friends with Alexis and Kyle instead of Austin and Andrew. *Hint:* If you get the error "too many levels of recursion," your triggers are probably inserting duplicate tuples.

```
pragma recursive_triggers = on;
delete from Friend where ID1 = 1641 and ID2 = 1782;
insert into Friend values (1641,1468);
update Friend set ID2 = 1247 where ID1 = 1501 and ID2 = 1316;
update Friend set ID2 = 1934 where ID1 = 1501 and ID2 = 1782;
```

Check the resulting database by writing a SQL query to compute the number of tuples in the *Friend* table. Compare your answer against ours.

3. Write triggers to manage the grade attribute of new highschoolers. If the inserted tuple has a non-null value for grade, don't permit the insert unless the grade is between 9 and 12. If the inserted tuple has a null value for grade, change it to 9.

Run the following statements to insert new highschoolers. To be on the safe side, disable recursive triggering ("pragma recursive_triggers = off;").

```
pragma recursive_triggers=off;
insert into HighSchooler values (2121, 'Caitlin', 7);
insert into HighSchooler values (2121, 'Caitlin', 20);
insert into HighSchooler values (2121, 'Caitlin', null);
insert into Highschooler select ID+1000, name, grade+1 from Highschooler;
```

Check the resulting database by writing one or more SQL queries to compute:

(a) The number of tuples in the *Highschooler* table

(b) The average grade level in the *Highschooler* table

Compare your answers against ours.

Before proceeding to Exercise 4, delete the new *Highschooler* tuples to bring the database back to its original state:
delete from Highschooler where ID > 2000;

4. Write a trigger that automatically deletes students when they graduate, i.e., when their grade is updated to exceed 12. Additionally, write a trigger or triggers that remove all friendships and likes relationships of deleted students.

Run the following statement to move Austin, Kyle, and Logan up a grade. To be on the safe side, disable recursive triggering ("pragma recursive_triggers = off;").

```
update Highschooler set grade = grade + 1
    where name = 'Austin' or name = 'Kyle' or name = 'Logan';
```

Check the resulting database by writing SQL queries to compute:

(a) The number of highschoolers remaining
(b) The number of *Friend* relationships (including symmetric ones)
(c) The number of *Likes* relationships

Compare your answers against ours.

5. Write a trigger so when a student is moved ahead one grade, then so are all of his or her friends. Your trigger from problem 4 should delete those students who "graduate" as a result.

Make sure recursive triggering is disabled ("pragma recursive_triggers = off;"), then run the following statement to move Andrew to 11th grade.

```
pragma recursive_triggers=off;
update Highschooler set grade = 11 where name = 'Andrew';
```

Check the resulting database by writing one or more SQL queries to compute:

(a) The number of tuples in the *Highschooler* table
(b) The average grade level in the *Highschooler* table

Compare your answers against ours.

Now run the following statements to move Tiffany to 11th grade and Jessica to 9th grade.

```
pragma recursive_triggers=off;
update Highschooler set grade = 11 where name = 'Tiffany';
update Highschooler set grade = 9 where name = 'Jessica';
```

Check the resulting database by writing one or more SQL queries to compute:

(c) The number of tuples in the *Highschooler* table
(d) The average grade level in the *Highschooler* table

Compare your answers against ours.

6. Continuing with the previous problem, now explore what happens when recursive triggering is enabled ("pragma recursive_triggers = on;"). Run the following statement to move Cassandra to 10th grade. *Hint:* Did you get the error "too many levels of recursion"? If so, you may need to add a condition to ensure updates don't continue indefinitely, preventing the graduation-delete rule from being activated.

```
pragma recursive_triggers = on;
update Highschooler set grade = 10 where name = 'Cassandra';
```

Check the resulting database by writing one or more SQL queries to compute:

(a) The names of all remaining highschoolers
(b) The number of *Friend* relationships (including symmetric ones)
(c) The number of *Likes* relationships

Compare your answers against ours.

Hide Query Results

1. 40

2. 40

3. (a) 30 (b) 10.6333

4. (a) 14 (b) 30 (c) 8

5. (a) 12 (b) 10.333 (c) 12 (d) 10.333

6. (a) John (b) 0 (c) 0