Bookmark this page

Students at your hometown high school have decided to organize their social network using databases. So far, they have collected information about sixteen students in four grades, 9-12. Here's the schema:

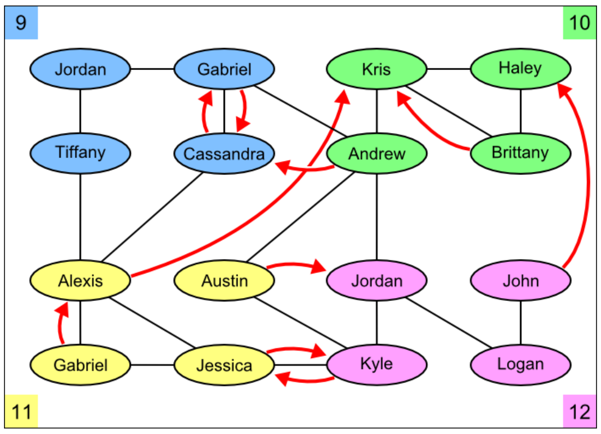
Highschooler (ID, name, grade)
English: There is a high school student with unique *ID* and a given *first name* in a certain *grade*.

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.

Your triggers will run over a small data set conforming to the schema. View the database. (You can also download the schema and data.)

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.



Instructions: You are to solve each of the following problems by writing one or more triggers. Our back-end creates triggers using SQLite on the original state of the sample database. It then performs a data modification statement that activate the trigger(s), runs a query to check that the final database state is correct, and restores the database to its original state. When you're satisfied with your solution for a given problem, click the "Submit" button to check your answer.

Important Notes:

- Our backend system is SQLite, so you must conform to the trigger constructs supported by SQLite. A guide to SQLite triggers is here, although you may find it easier to start from the triggers used in the video demonstrations.
- In the workbench and the grading program, triggers are executed with recursive triggering disabled ("recursive_triggers=off").
- *You are to translate the English into one or more triggers that perform the desired actions for all possible databases and modifications.* All we actually check is that the verification query gets the right answer on the small sample database. Thus, even if your solution is marked as correct, it is possible that your solution does not correctly reflect the problem at hand. Circumventing the system in this fashion will get you a high score on the exercises, but it won't help you learn about triggers. On the other hand, an incorrect attempt at a general solution is unlikely to behave correctly, so you shouldn't be led astray by our checking system.

You may perform these exercises as many times as you like, so we strongly encourage you to keep working with them until you complete the exercises with full credit.

Q1

1.0/1.0 point (graded)
Write a trigger that makes new students named 'Friendly' automatically like everyone else in their grade. That is, after the trigger runs, we should have ('Friendly', A) in the Likes table for every other Highschooler A in the same grade as 'Friendly'.

- Your triggers are created in SQLite, so you must conform to the trigger constructs supported by SQLite.

```
CREATE TRIGGER Friendlies
```

```
AFTER INSERT ON Highschooler
FOR EACH ROW
WHEN (NEW.name = "Friendly")
BEGIN
    INSERT INTO Likes
        SELECT NEW.ID, ID
        FROM Highschooler
        WHERE grade = NEW.grade AND ID <> NEW.ID;
END;
```

Press ESC then TAB or click outside of the code editor to exit

Correct

Correct

Trigger command(s) were executed.

To check your trigger(s), we first ran the following data modification statement(s): *insert into Highschooler values (1000, 'Friendly', 9);*
insert into Highschooler values (2000, 'Friendly', 11);
insert into Highschooler values (3000, 'Unfriendly', 10).

We then ran the following query: *select H1.name, H1.grade, H2.name, H2.grade from Likes L, Highschooler H1, Highschooler H2 where L.ID1 = H1.ID and L.ID2 = H2.ID order by H1.name, H1.grade, H2.name, H2.grade*

Your Query Result:

Alexis	11	Kris	10
Andrew	10	Cassandra	9
Austin	11	Jordan	12
Brittany	10	Kris	10
Cassandra	9	Gabriel	9
Friendly	9	Cassandra	9
Friendly	9	Gabriel	9
Friendly	9	Jordan	9
Friendly	9	Tiffany	9
Friendly	11	Alexis	11
Friendly	11	Austin	11
Friendly	11	Gabriel	11
Friendly	11	Jessica	11
Gabriel	9	Cassandra	9
Gabriel	11	Alexis	11
Jessica	11	Kyle	12
John	12	Haley	10
Kyle	12	Jessica	11

Expected Query Result:

Alexis	11	Kris	10
Andrew	10	Cassandra	9
Austin	11	Jordan	12
Brittany	10	Kris	10
Cassandra	9	Gabriel	9
Friendly	9	Cassandra	9
Friendly	9	Gabriel	9
Friendly	9	Jordan	9
Friendly	9	Tiffany	9
Friendly	11	Alexis	11
Friendly	11	Austin	11
Friendly	11	Gabriel	11
Friendly	11	Jessica	11

Gabriel	9	Cassandra	9
Gabriel	11	Alexis	11
Jessica	11	Kyle	12
John	12	Haley	10
Kyle	12	Jessica	11

Submit

Q2

1.0/1.0 point (graded)

Write one or more triggers to manage the grade attribute of new Highschoolers. If the inserted tuple has a value less than 9 or greater than 12, change the value to NULL. On the other hand, if the inserted tuple has a null value for grade, change it to 9.

- Your triggers are created in SQLite, so you must conform to the trigger constructs supported by SQLite.
- To create more than one trigger, separate the triggers with a vertical bar (|).

```
CREATE TRIGGER Grade
BEFORE INSERT ON Highschooler
FOR EACH ROW
WHEN (NEW.grade < 9 OR NEW.grade > 12 OR NEW.grade IS NULL)
BEGIN
    INSERT INTO Highschooler SELECT
        NEW.ID, NEW.name,
                    (SELECT NULL WHERE NEW.grade < 9 OR NEW.grade > 12
                    UNION
                    SELECT 9 WHERE NEW.grade IS NULL);
    SELECT RAISE(IGNORE);
END;
```

Press ESC then TAB or click outside of the code editor to exit

Correct

Correct

Trigger command(s) were executed.

To check your trigger(s), we first ran the following data modification statement(s): *insert into Highschooler values (2121, 'Caitlin', null);*
insert into Highschooler values (2122, 'Don', null);
insert into Highschooler values (2123, 'Elaine', 7);
insert into Highschooler values (2124, 'Frank', 20);
insert into Highschooler values (2125, 'Gale', 10)
.

We then ran the following query: *select * from Highschooler order by ID*

Your Query Result:

1025	John	12
1101	Haley	10
1247	Alexis	11
1304	Jordan	12
1316	Austin	11
1381	Tiffany	9
1468	Kris	10
1501	Jessica	11
1510	Jordan	9
1641	Brittany	10
1661	Logan	12
1689	Gabriel	9
1709	Cassandra	9
1782	Andrew	10
1911	Gabriel	11
1934	Kyle	12

2121	Caitlin	9
2122	Don	9
2123	Elaine	<NULL>
2124	Frank	<NULL>
2125	Gale	10

Expected Query Result:

1025	John	12
1101	Haley	10
1247	Alexis	11
1304	Jordan	12
1316	Austin	11
1381	Tiffany	9
1468	Kris	10
1501	Jessica	11
1510	Jordan	9
1641	Brittany	10
1661	Logan	12
1689	Gabriel	9
1709	Cassandra	9
1782	Andrew	10
1911	Gabriel	11
1934	Kyle	12
2121	Caitlin	9
2122	Don	9
2123	Elaine	<NULL>
2124	Frank	<NULL>
2125	Gale	10

Submit

Q3

1.0/1.0 point (graded)

Write one or more triggers to maintain symmetry in friend relationships. Specifically, if (A,B) is deleted from Friend, then (B,A) should be deleted too. If (A,B) is inserted into Friend then (B,A) should be inserted too. Don't worry about updates to the Friend table.

- Your triggers are created in SQLite, so you must conform to the trigger constructs supported by SQLite.
- To create more than one trigger, separate the triggers with a vertical bar (|).

```
CREATE TRIGGER FriendDel
AFTER DELETE ON Friend
FOR EACH ROW
BEGIN
    DELETE FROM Friend WHERE ID1 = OLD.ID2 AND ID2 = OLD.ID1;
END;
|
CREATE TRIGGER FriendAdd
AFTER INSERT ON Friend
FOR EACH ROW
BEGIN
    INSERT INTO Friend VALUES (NEW.ID2, NEW.ID1);
END;
```

Press ESC then TAB or click outside of the code editor to exit

Correct

Correct

Trigger command(s) were executed.

To check your trigger(s), we first ran the following data modification statement(s): *delete from Friend where ID1 = 1641 and ID2 = 1468;*

delete from Friend where ID1 = 1247 and ID2 = 1911;
insert into Friend values (1510, 1934);
insert into Friend values (1101, 1709).
English description of modifications: Deleted friendship (Brittany, 10, Kris, 10);
Deleted friendship (Alexis, 11, Gabriel, 11);
Inserted friendship (Jordan, 9, Kyle, 12);
Inserted friendship (Haley, 10, Cassandra, 9)

We then ran the following query: *select H1.name, H1.grade, H2.name, H2.grade from Friend F, Highschooler H1, Highschooler H2 where F.ID1 = H1.ID and F.ID2 = H2.ID order by H1.name, H1.grade, H2.name, H2.grade*

Your Query Result:

Alexis	11	Cassandra	9
Alexis	11	Jessica	11
Alexis	11	Tiffany	9
Andrew	10	Austin	11
Andrew	10	Gabriel	9
Andrew	10	Jordan	12
Andrew	10	Kris	10
Austin	11	Andrew	10
Austin	11	Kyle	12
Brittany	10	Haley	10
Cassandra	9	Alexis	11
Cassandra	9	Gabriel	9
Cassandra	9	Haley	10
Gabriel	9	Andrew	10
Gabriel	9	Cassandra	9
Gabriel	9	Jordan	9
Gabriel	11	Jessica	11
Haley	10	Brittany	10
Haley	10	Cassandra	9
Haley	10	Kris	10
Jessica	11	Alexis	11
Jessica	11	Gabriel	11
Jessica	11	Kyle	12
John	12	Logan	12
Jordan	9	Gabriel	9
Jordan	9	Kyle	12
Jordan	9	Tiffany	9
Jordan	12	Andrew	10
Jordan	12	Kyle	12
Jordan	12	Logan	12
Kris	10	Andrew	10
Kris	10	Haley	10
Kyle	12	Austin	11
Kyle	12	Jessica	11
Kyle	12	Jordan	9
Kyle	12	Jordan	12
Logan	12	John	12
Logan	12	Jordan	12
Tiffany	9	Alexis	11
Tiffany	9	Jordan	9

Expected Query Result:

Alexis	11	Cassandra	9
Alexis	11	Jessica	11
Alexis	11	Tiffany	9
Andrew	10	Austin	11

Andrew	10	Gabriel	9
Andrew	10	Jordan	12
Andrew	10	Kris	10
Austin	11	Andrew	10
Austin	11	Kyle	12
Brittany	10	Haley	10
Cassandra	9	Alexis	11
Cassandra	9	Gabriel	9
Cassandra	9	Haley	10
Gabriel	9	Andrew	10
Gabriel	9	Cassandra	9
Gabriel	9	Jordan	9
Gabriel	11	Jessica	11
Haley	10	Brittany	10
Haley	10	Cassandra	9
Haley	10	Kris	10
Jessica	11	Alexis	11
Jessica	11	Gabriel	11
Jessica	11	Kyle	12
John	12	Logan	12
Jordan	9	Gabriel	9
Jordan	9	Kyle	12
Jordan	9	Tiffany	9
Jordan	12	Andrew	10
Jordan	12	Kyle	12
Jordan	12	Logan	12
Kris	10	Andrew	10
Kris	10	Haley	10
Kyle	12	Austin	11
Kyle	12	Jessica	11
Kyle	12	Jordan	9
Kyle	12	Jordan	12
Logan	12	John	12
Logan	12	Jordan	12
Tiffany	9	Alexis	11
Tiffany	9	Jordan	9

Submit

Q4

1.0/1.0 point (graded)

Write a trigger that automatically deletes students when they graduate, i.e., when their grade is updated to exceed 12.

- Your triggers are created in SQLite, so you must conform to the trigger constructs supported by SQLite.

```
CREATE TRIGGER Graduation
AFTER UPDATE OF grade ON Highschooler
FOR EACH ROW
WHEN (NEW.grade > 12)
BEGIN
    DELETE FROM Highschooler WHERE ID = NEW.ID;
END;
```

Press ESC then TAB or click outside of the code editor to exit

Correct

Correct

Trigger command(s) were executed.

To check your trigger(s), we first ran the following data modification statement(s): *update Highschooler set grade = grade + 1 where name = 'Austin' or name = 'Kyle' or name = 'Logan'*.

We then ran the following query: *select * from Highschooler order by name, grade*

Your Query Result:

1247	Alexis	11
1782	Andrew	10
1316	Austin	12
1641	Brittany	10
1709	Cassandra	9
1689	Gabriel	9
1911	Gabriel	11
1101	Haley	10
1501	Jessica	11
1025	John	12
1510	Jordan	9
1304	Jordan	12
1468	Kris	10
1381	Tiffany	9

Expected Query Result:

1247	Alexis	11
1782	Andrew	10
1316	Austin	12
1641	Brittany	10
1709	Cassandra	9
1689	Gabriel	9
1911	Gabriel	11
1101	Haley	10
1501	Jessica	11
1025	John	12
1510	Jordan	9
1304	Jordan	12
1468	Kris	10
1381	Tiffany	9

Submit

Q5

1.0/1.0 point (graded)

Write a trigger that automatically deletes students when they graduate, i.e., when their grade is updated to exceed 12 (same as Question 4). In addition, write a trigger so when a student is moved ahead one grade, then so are all of his or her friends.

- Your triggers are created in SQLite, so you must conform to the trigger constructs supported by SQLite.
- To create more than one trigger, separate the triggers with a vertical bar (|).

```
CREATE TRIGGER Graduation
AFTER UPDATE OF grade ON Highschooler
FOR EACH ROW
WHEN (NEW.grade > 12)
BEGIN
    DELETE FROM Highschooler WHERE ID = NEW.ID;
END;
|
CREATE TRIGGER MoveYear
BEFORE UPDATE OF grade ON Highschooler
```

```
FOR EACH ROW
BEGIN
    UPDATE Highschooler SET grade = grade + 1
        WHERE ID IN (SELECT ID2 FROM Friend WHERE ID1 = NEW.ID);
END;
```

Press ESC then TAB or click outside of the code editor to exit

Correct

Correct

Trigger command(s) were executed.

To check your trigger(s), we first ran the following data modification statement(s): *update Highschooler set grade = grade + 1 where name = 'Austin' or name = 'Kyle' or name = 'Logan'*.

We then ran the following query: *select * from Highschooler order by name, grade*

Your Query Result:

1247	Alexis	11
1782	Andrew	11
1316	Austin	12
1641	Brittany	10
1709	Cassandra	9
1689	Gabriel	9
1911	Gabriel	11
1101	Haley	10
1501	Jessica	11
1510	Jordan	9
1468	Kris	10
1381	Tiffany	9

Expected Query Result:

1247	Alexis	11
1782	Andrew	11
1316	Austin	12
1641	Brittany	10
1709	Cassandra	9
1689	Gabriel	9
1911	Gabriel	11
1101	Haley	10
1501	Jessica	11
1510	Jordan	9
1468	Kris	10
1381	Tiffany	9

Submit

Q6

1.0/1.0 point (graded)

Write a trigger to enforce the following behavior: If A liked B but is updated to A liking C instead, and B and C were friends, make B and C no longer friends. Don't forget to delete the friendship in both directions, and make sure the trigger only runs when the "liked" (ID2) person is changed but the "liking" (ID1) person is not changed.

- Your triggers are created in SQLite, so you must conform to the trigger constructs supported by SQLite.

```
CREATE TRIGGER FriendDel
AFTER DELETE ON Friend
FOR EACH ROW
BEGIN
    DELETE FROM Friend WHERE ID1 = OLD.ID2 AND ID2 = OLD.ID1;
END;
|
CREATE TRIGGER EndFriendAfterLike
```

```
AFTER UPDATE ON Likes
FOR EACH ROW
WHEN (OLD.ID1 = NEW.ID1)
BEGIN
    DELETE FROM Friend WHERE ID1 = OLD.ID2 AND ID2 = NEW.ID2;
END;
```
Press ESC then TAB or click outside of the code editor to exit

Correct

Correct

Trigger command(s) were executed.

To check your trigger(s), we first ran the following data modification statement(s): *update Likes set ID2 = 1501 where ID1 = 1911; update Likes set ID2 = 1316 where ID1 = 1501; update Likes set ID2 = 1304 where ID1 = 1934; update Likes set ID1 = 1661, ID2 = 1641 where ID1 = 1025; update Likes set ID2 = 1468 where ID1 = 1247.*
English description of modifications: Changed Gabriel-11 to like Jessica-11 instead of Alexis-11;
Changed Jessica-11 to like Austin-11 instead of Kyle-12;
Changed Kyle-12 to like Jordan-12 instead of Jessica-11;
Changed 'John-12 liking Haley-10' to 'Logan-12 liking Brittany-10';
Changed Alexis-11 to like Kris-10 instead of Kris-10 (so no actual change)

We then ran the following query: *select H1.name, H1.grade, H2.name, H2.grade from Friend F, Highschooler H1, Highschooler H2 where F.ID1 = H1.ID and F.ID2 = H2.ID order by H1.name, H1.grade, H2.name, H2.grade*

Your Query Result:

Alexis	11	Cassandra	9
Alexis	11	Gabriel	11
Alexis	11	Tiffany	9
Andrew	10	Austin	11
Andrew	10	Gabriel	9
Andrew	10	Jordan	12
Andrew	10	Kris	10
Austin	11	Andrew	10
Brittany	10	Haley	10
Brittany	10	Kris	10
Cassandra	9	Alexis	11
Cassandra	9	Gabriel	9
Gabriel	9	Andrew	10
Gabriel	9	Cassandra	9
Gabriel	9	Jordan	9
Gabriel	11	Alexis	11
Gabriel	11	Jessica	11
Haley	10	Brittany	10
Haley	10	Kris	10
Jessica	11	Gabriel	11
Jessica	11	Kyle	12
John	12	Logan	12
Jordan	9	Gabriel	9
Jordan	9	Tiffany	9
Jordan	12	Andrew	10
Jordan	12	Kyle	12
Jordan	12	Logan	12
Kris	10	Andrew	10
Kris	10	Brittany	10
Kris	10	Haley	10
Kyle	12	Jessica	11
Kyle	12	Jordan	12
Logan	12	John	12
Logan	12	Jordan	12
Tiffany	9	Alexis	11
Tiffany	9	Jordan	9

Expected Query Result:

Alexis	11	Cassandra	9
Alexis	11	Gabriel	11
Alexis	11	Tiffany	9
Andrew	10	Austin	11
Andrew	10	Gabriel	9
Andrew	10	Jordan	12
Andrew	10	Kris	10
Austin	11	Andrew	10
Brittany	10	Haley	10
Brittany	10	Kris	10
Cassandra	9	Alexis	11
Cassandra	9	Gabriel	9
Gabriel	9	Andrew	10
Gabriel	9	Cassandra	9
Gabriel	9	Jordan	9
Gabriel	11	Alexis	11
Gabriel	11	Jessica	11
Haley	10	Brittany	10
Haley	10	Kris	10
Jessica	11	Gabriel	11
Jessica	11	Kyle	12
John	12	Logan	12
Jordan	9	Gabriel	9
Jordan	9	Tiffany	9
Jordan	12	Andrew	10
Jordan	12	Kyle	12
Jordan	12	Logan	12
Kris	10	Andrew	10
Kris	10	Brittany	10
Kris	10	Haley	10
Kyle	12	Jessica	11
Kyle	12	Jordan	12
Logan	12	John	12
Logan	12	Jordan	12
Tiffany	9	Alexis	11
Tiffany	9	Jordan	9

Submit