SQL Social-Network Query Exercises

Exercise due Aug 29, 2021 19:00 +08 Completed

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:

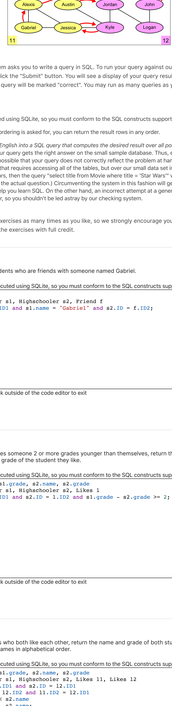
Highschooler (ID, name, grade)
English: There is a high school student with unique *ID* and a given *first name* in a certain *grade*.

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 queries 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 students 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 student likes another student.



Instructions: Each problem asks you to write a query in SQL. To run your query against our back-end sample database using SQLite, click the "Submit" button. You will see a display of your query result and the expected result. If the results match, your query will be marked "correct". You may run as many queries as you like for each question.

Important Notes:

- Your queries are executed using SQLite, so you must conform to the SQL constructs supported by SQLite.
- Unless a specific result ordering is asked for, you can return the result rows in any order.
- *You are to translate the English into a SQL query that computes the desired result over all possible databases.* All we actually check is that your query gets the right answer on the small sample database. Thus, even if *your solution is marked as correct, it is possible that your query does not correctly reflect the problem at hand. (For example, if we ask* for a complex condition that requires accessing all of the tables, but over our small data set in the end the condition is satisfied only by Star Wars, then the query "select title from Movie where title = 'Star Wars'" will be marked correct even though it doesn't reflect the actual question.) Circumventing the system in this fashion will get you a high score on the exercises, but it won't help you learn SQL. On the other hand, an incorrect attempt at a general solution is unlikely to produce the right answer, 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

1/1 point (graded)
Find the names of all students who are friends with someone named Gabriel.

Note: Your queries are executed using SQLite, so you must conform to the SQL constructs supported by SQLite.

```
select s2.name
from Highschooler s1, Highschooler s2, Friend f
where s1.ID = f.ID1 and s1.name = 'Gabriel' and s2.ID = f.ID2;
```

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

Submit

Correct

Q2

1/1 point (graded)
For every student who likes someone 2 or more grades younger than themselves, return that student's name and grade, and the name and grade of the student they like.

Note: Your queries are executed using SQLite, so you must conform to the SQL constructs supported by SQLite.

```
select s1.name, s1.grade, s2.name, s2.grade
from Highschooler s1, Highschooler s2, Likes l1, Likes l2
where s1.ID = l1.ID1 and s2.ID = l2.ID1
    and l1.ID1 = l2.ID2 and l1.ID2 = l2.ID1
    and s1.name < s2.name
order by s1.name, s2.name;
```

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

Submit

Correct

Q3

1/1 point (graded)
For every pair of students who both like each other, return the name and grade of both students. Include each pair only once, with the two names in alphabetical order.

Note: Your queries are executed using SQLite, so you must conform to the SQL constructs supported by SQLite.

```
select s1.name, s1.grade, s2.name, s2.grade
from Highschooler s1, Highschooler s2, Likes l1, Likes l2
where s1.ID = l1.ID1 and s2.ID = l2.ID1
    and l1.ID1 = l2.ID2 and l1.ID2 = l2.ID1
    and s1.name < s2.name
order by s1.name, s2.name;
```

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

Submit

Correct

Q4

1/1 point (graded)
Find all students who do not appear in the Likes table (as a student who likes or is liked) and return their names and grades. Sort by grade, then by name within each grade.

Note: Your queries are executed using SQLite, so you must conform to the SQL constructs supported by SQLite.

```
select s.name, s.grade
from Highschooler s
where s.ID not in (select ID1 from Likes)
    and s.ID not in (select ID2 from Likes)
order by s.grade, s.name;
```

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

Submit

Correct

Q5

1/1 point (graded)
For every situation where student A likes student B, but we have no information about whom B likes (that is, B does not appear as an ID1 in the Likes table), return A and B's names and grades.

Note: Your queries are executed using SQLite, so you must conform to the SQL constructs supported by SQLite.

```
select s1.name, s1.grade, s2.name, s2.grade
from Highschooler s1, Highschooler s2, Likes l1
where s1.ID = l1.ID1 and s2.ID = l1.ID2
    and l1.ID2 not in (select ID1 from Likes);
```

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

Submit

Correct

Q6

1/1 point (graded)
Find names and grades of students who only have friends in the same grade. Return the result sorted by grade, then by name within each grade.

Note: Your queries are executed using SQLite, so you must conform to the SQL constructs supported by SQLite.

```
select name, grade
from Highschooler
where ID not in
    (select A1.ID
    from Highschooler s1, Friend f1, Highschooler s2
    where s1.ID = f1.ID1 and s2.ID = f1.ID2 and s2.grade <> s1.grade)
order by grade, name;
```

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

Submit

Correct

Q7

1/1 point (graded)
For each student A who likes a student B where the two are not friends, find if they have a friend C in common (who can introduce them!). For all such trios, return the name and grade of A, B, and C.

Note: Your queries are executed using SQLite, so you must conform to the SQL constructs supported by SQLite.

```
select s1.name, s1.grade, s2.name, s2.grade, s3.name, s3.grade
from Highschooler s1, Highschooler s2, Highschooler s3, Likes l1, Friend f1, Friend f2
where s1.ID = l1.ID1 and s2.ID = l1.ID2 and s1.ID = f1.ID1 and s2.ID = f2.ID1
    and s2.ID not in (select ID2 from Friend where ID1 = s1.ID)
    and s1.ID not in (select ID2 from Friend where ID1 = s2.ID)
    and s3.ID = f1.ID2 and s3.ID = f2.ID2;
```

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

Submit

Correct

Q8

1/1 point (graded)
Find the difference between the number of students in the school and the number of different first names.

Note: Your queries are executed using SQLite, so you must conform to the SQL constructs supported by SQLite.

```
select count(*) - count(distinct name)
from Highschooler;
```

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

Submit

Correct

Q9

1/1 point (graded)
Find the name and grade of all students who are liked by more than one other student.

Note: Your queries are executed using SQLite, so you must conform to the SQL constructs supported by SQLite.

```
select s2.name, s2.grade
from Highschooler s1, Highschooler s2, Likes l1
where l1.ID1 = s1.ID and l1.ID2 = s2.ID
group by l1.ID2
having count(l1.ID1) > 1
```

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

Submit

Correct

Previous
Next Up: SQL Social-Network Query Exercises Extras
2 min + 1 activity