Notifications

Pursue a verified certificate

- Earn a **verified certificate** of completion to showcase on your resumé
- Support our **mission** at edX

Upgrade for $50

SQL Social-Network Query Exercises

Bookmark this page

Hide Notes

Exercise due May 11, 2022 00:52 IST

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:

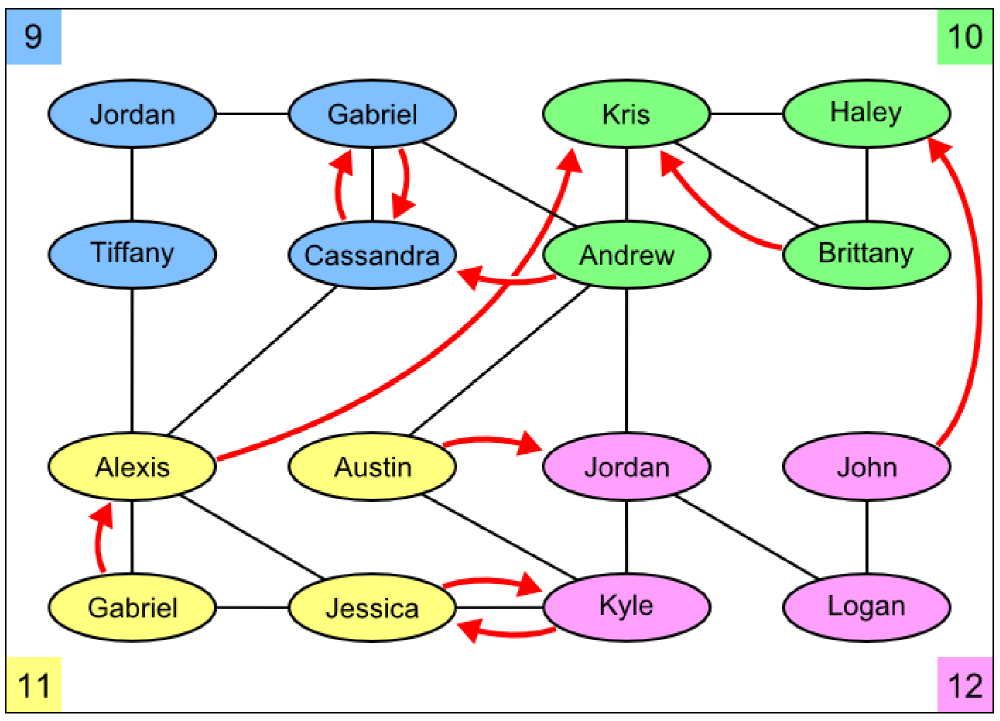
Highschooler (ID, name, grade)
English: There is a high school student with unique *ID* and a given *first name* in a certain *grade*.

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

Hide Notes

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 name from Highschooler where ID in
(SELECT ID2 from Friend as f JOIN Highschooler as h on
union
SELECT ID1 from Friend as f JOIN Highschooler as h on
```

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

Hide Notes

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 h.name, h.grade, h1.name, h1.grade as G from Li
```

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 h.name, h.grade, h1.name, h1.grade from Likes a
JOIN Highschooler as h on l1.ID1 = h.ID JOIN Highschoo
```

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 n, g FROM
(SELECT h.name as n, h.grade as g FROM Highschooler as
intersect
```

Hide Notes

```
SELECT h.name as n, h.grade as g FROM Highschooler as
```

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 h.name, h.grade, h1.name, h1.grade from Likes a
JOIN Highschooler as h on l1.ID1 = h.ID JOIN Highschoo
```

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

Submit

Correct

Q6

0/1 points (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 h.name, h.grade from Friend as f
JOIN Highschooler as h on f.ID1 = h.ID
JOIN Highschooler as h1 on f.ID2 = h1.ID
group by f.ID1 having count(distinct h1.grade) = 1 and
order by h.grade, h.name;
```

Hide Notes

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

Submit

Correct

Your Query Result:

Jordan	9
Brittany	10
Haley	10
Kris	10
Gabriel	11
John	12
Logan	12

Expected Query Result:

Jordan	9
Brittany	10
Haley	10
Kris	10
Gabriel	11
John	12
Logan	12

Q7

0/1 points (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 DISTINCT H1.name, H1.grade, H2.name, H2.grade,
FROM Highschooler H1, Highschooler H2, Highschooler H3
WHERE (H1.ID = L.ID1 AND H2.ID = L.ID2) AND H2.ID NOT
  SELECT ID2
  FROM Friend
  WHERE ID1 = H1.ID
) AND (H1.ID = F1.ID1 AND H3.ID = F1.ID2) AND (H2.ID =

```

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

Submit

Correct

Your Query Result:

Andrew	10	Cassandra	9	Gabriel	9
Austin	11	Jordan	12	Andrew	10
Austin	11	Jordan	12	Kyle	12

Hide Notes

Expected Query Result:

Andrew	10	Cassandra	9	Gabriel	9
Austin	11	Jordan	12	Andrew	10
Austin	11	Jordan	12	Kyle	12

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 c1-c2 from (SELECT count(name) as c1, count(dis
```

‹ Previous Next ›

Hide Notes

Connect

Blog
Contact Us
Help Center
Media Kit

GET IT ON Google play
Download on the App Store

Hide Notes