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:

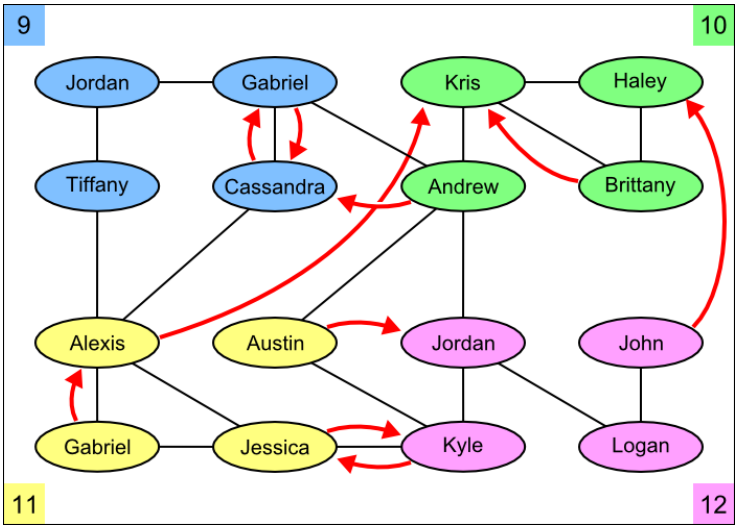
Highschooler (ID, name, grade)
English: There is a high school student with unique *ID* and a given *first name* in a certain *grade*.

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. When you click "Check Answer" our back-end runs your query against the sample database using SQLite. It displays the result and compares your answer against the correct one. When you're satisfied with your solution for a given problem, click the "Save Answers" button to save your progress. Click "Submit Answers" to submit the entire exercise set.

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)

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 distinct name from highschooler h join friend f
on h.ID = f.ID2
where ID IN
(Select ID2 from highschooler h join friend f
on h.ID = f.ID1
where h.name = 'Gabriel')
```

Correct

Correct

Your Query Result:

Alexis
Andrew
Cassandra
Jessica
Jordan

Expected Query Result:

Alexis
Andrew

Cassandra
Jessica
Jordan

Submit **Reset**

Q2 (1/1 point)

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.

```
1 select h2.name,h2.grade,h1.name,h1.grade from highschooler h1, highschooler h2, likes l
2 where h1.ID = l.ID2 and h2.ID = l.ID1
3 and (h2.grade - h1.grade) >= 2
```

Correct

Correct

Your Query Result:

John	12	Haley	10

Expected Query Result:

John	12	Haley	10

Submit **Reset**

Q3 (1/1 point)

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 h1.Name,h1.grade,h2.name,h2.grade from likes l1,likes l2, highschooler h1, highschooler h2
where h1.ID = l1.ID1 and h2.ID = l1.ID2
and l1.ID1 = l2.ID2 and l1.ID2 = l2.ID1
and h1.name < h2. name
```

Correct

Correct

Your Query Result:

Cassandra	9	Gabriel	9
Jessica	11	Kyle	12

Expected Query Result:

Cassandra	9	Gabriel	9
Jessica	11	Kyle	12

Submit **Reset**

Q4 (1/1 point)

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 h.name,h.grade from highschooler h
where h.ID NOT IN (select ID1 from likes)
and h.ID NOT IN (select ID2 from likes)
order by grade,name
```

Correct

Correct

Your Query Result:

Jordan	9
Tiffany	9
Logan	12

Expected Query Result:

Jordan	9
Tiffany	9
Logan	12

(Order matters)

Submit **Reset**

Q5 (1/1 point)

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.

```
1 select h1.Name,h1.grade,h2.Name,h2.grade from likes l, highschooler h1, highschooler h2
2 where l.ID2 NOT IN (select ID1 from likes)
3 and l.ID1 = h1.ID and l.ID2 = h2.ID
```

Correct

Correct

Your Query Result:

Alexis	11	Kris	10

Austin	11	Jordan	12
Brittany	10	Kris	10
John	12	Haley	10

Expected Query Result:

Alexis	11	Kris	10
Austin	11	Jordan	12
Brittany	10	Kris	10
John	12	Haley	10

Submit **Reset**

Q6 (1/1 point)

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 f.ID1 from friend f, highschooler h1, highschooler h2
where f.ID1 = h1.ID and f.ID2 = h2.ID
and h1.grade <> h2.grade)
order by grade,name
```

Correct

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

(Order matters)

Submit **Reset**

Q7 (1/1 point)

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.

```
1 select h1.name,h1.grade,h3.name,h3.grade,h2.name,h2.grade from likes l, friend f1, friend f2, highschooler h1, hi
2 where l.ID1 = h1.ID and l.ID2 = h3.ID
3 and h1.ID = f1.ID1 and h2.ID = f1.ID2
4 and h2.ID = f2.ID1 and h3.ID = f2.ID2
5 and h3.ID NOT IN (select ID2 from friend where ID1=h1.ID)
```

Correct

Correct

Your Query Result:

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

Expected Query Result:

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

Submit **Reset**

Q8 (1/1 point)

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.

```
1 select count(*)- count(distinct name) from highschooler
```

Correct

Correct

Your Query Result:

2

Expected Query Result:

2

Submit **Reset**

Q9 (1/1 point)

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 name,grade from highschooler
where ID IN
(select ID2 from likes
group by ID2
having count(ID1) > 1)
```

Correct

Correct

Your Query Result:

Cassandra	9
Kris	10

Expected Query Result:

Cassandra	9
Kris	10

Submit **Reset**

Previous Next