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:

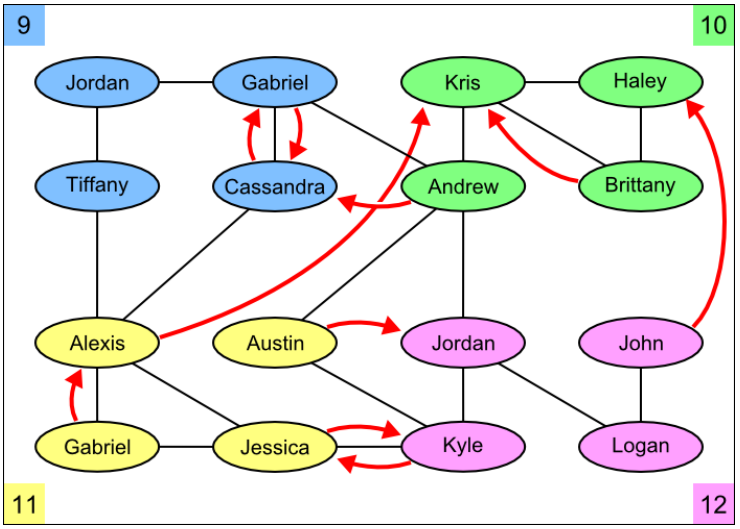
Highschooler (ID, name, grade)
English: There is a high school student with unique *ID* and a given *first name* in a certain *grade*.

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)

For every situation where student A likes student B, but student B likes a different student C, return the names and grades 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,h2.name,h2.grade,h3.name,h3.grade from likes l1,likes l2, highschooler h1,highschooler h2
2 where l1.ID2 = l2.ID1
3 and l1.ID1 = h1.ID and l2.ID2 = h3.ID
4 and l1.ID2 = h2.ID and l1.ID1 <> l2.ID2
```

Correct

Correct

Your Query Result:

Andrew	10	Cassandra	9	Gabriel	9
Gabriel	11	Alexis	11	Kris	10

Expected Query Result:

Andrew	10	Cassandra	9	Gabriel	9
Gabriel	11	Alexis	11	Kris	10

Submit **Reset**

Q2 (1/1 point)

Find those students for whom all of their friends are in different grades from themselves. Return the students' names and grades.

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 ID1 from friend
except
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
)
```

Correct

Correct

Your Query Result:

Austin	11

Expected Query Result:

Austin	11

Submit **Reset**

Q3 (1/1 point)

What is the average number of friends per student? (Your result should be just one number.)

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

```
select avg (NewTable.value) from (select count(ID2) as value from friend
group by ID1) as NewTable
```

Correct

Correct

Your Query Result:

2.5

Expected Query Result:

2.5

Submit **Reset**

Q4 (1/1 point)

Find the number of students who are either friends with Cassandra or are friends of friends of Cassandra. Do not count Cassandra, even though technically she is a friend of a friend.

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

```
select count(h2.ID)+count(distinct f.ID1) from friend f,highschooler h2
where f.ID2 = h2.ID and h2.name <> "Cassandra"
and f.ID1 IN
(
select h2.ID from friend f,highschooler h1,highschooler h2
where f.ID1 = h1.ID and h1.name = "Cassandra"
and f.ID2 = h2.ID
)
```

Correct

Correct

Your Query Result:

7

Expected Query Result:

7

Submit **Reset**

Q5 (1/1 point)

Find the name and grade of the student(s) with the greatest number of friends.

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

```
select name,grade from friend f join highschooler h
on f.ID1 = h.ID
group by ID1
having count(ID2) =
(
select max(MAXVL.countvalue)
from (select count(ID2) as countvalue, ID1 as ID
from friend group by ID1) as MAXVL
)
```

Correct

Correct

Your Query Result:

Alexis	11
Andrew	10

Expected Query Result:

Alexis	11
Andrew	10

Submit **Reset**

Previous Next