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:

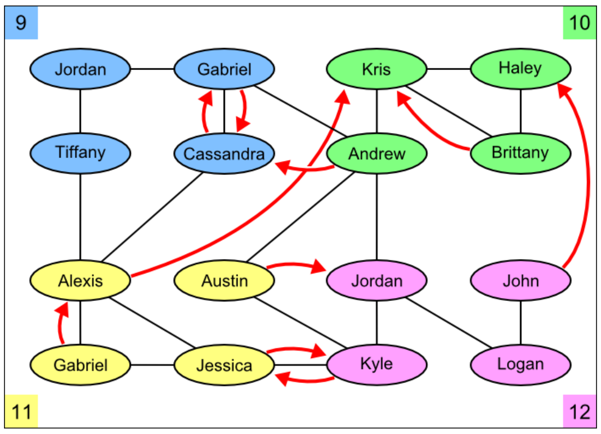
Highschooler (ID, name, grade)
English: There is a high school student with unique *ID* and a given *first name* in a certain *grade*.

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

1.0/1.0 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 DISTINCT ID2 FROM Friend
             WHERE ID1 IN (SELECT ID FROM Highschooler WHERE name = "Gabriel"))
```

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

Correct

Correct

Your Query Result:

Alexis
Andrew
Cassandra
Jessica
Jordan

Expected Query Result:

Alexis
Andrew
Cassandra
Jessica
Jordan

Submit

Q2

1.0/1.0 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 H1.name, H1.grade, H2.name, H2.grade
FROM Highschooler AS H1, Highschooler AS H2, Likes
WHERE H1.ID = ID1 AND H2.ID = ID2 AND (H1.grade - H2.grade) > 1
```

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

Correct

Correct

Your Query Result:

John	12	Haley	10

Expected Query Result:

John	12	Haley	10

Submit

Q3

1.0/1.0 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 H1.name, H1.grade, H2.name, H2.grade
FROM Likes AS L1, Likes AS L2, Highschooler AS H1, Highschooler AS H2
WHERE L1.ID1 = L2.ID2 AND L1.ID2 = L2.ID1 AND H1.name < H2.name AND H1.ID = L1.ID1 AND H2.ID = L1.ID2
```

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

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

Q4

1.0/1.0 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 name, grade
FROM Highschooler
WHERE ID NOT IN (SELECT ID1 FROM Likes) AND ID NOT IN (SELECT ID2 FROM Likes)
ORDER BY grade, name
```

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

Correct

Correct

Your Query Result:

Jordan	9
Tiffany	9
Logan	12

Expected Query Result:

Jordan	9
Tiffany	9
Logan	12

(Order matters)

Submit

Q5

1.0/1.0 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 H1.name, H1.grade, H2.name, H2.grade
FROM Likes, Highschooler AS H1, Highschooler AS H2
WHERE H1.ID = ID1 AND H2.ID = ID2 AND ID2 NOT IN (SELECT ID1 FROM Likes)
```

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

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

Q6

1.0/1.0 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 H1N, H1G
FROM
    (SELECT ID1, H1.name AS H1N, H1.grade AS H1G, ID2, H2.grade AS H2G
    FROM Friend, Highschooler AS H1, Highschooler AS H2
    WHERE H1.ID = Friend.ID1 AND H2.ID = Friend.ID2)
GROUP BY ID1
HAVING COUNT(DISTINCT H2G) = 1
ORDER BY H1G, H1N
```

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

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

Q7

1.0/1.0 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 FrA.name, FrA.grade, FrB.name, FrB.grade, FrC.name, FrC.grade
FROM (SELECT ID1 AS A, ID2 AS B, ID1 || "-" || ID2 AS likeRel FROM Likes
      WHERE Likes.ID1 IN (SELECT ID1 FROM Friend WHERE Likes.ID1 = Friend.ID1)
      AND Likes.ID2 NOT IN (SELECT ID2 FROM Friend WHERE Likes.ID1 = Friend.ID1)),

    (SELECT F1.ID1 AS C, F1.ID2, F2.ID2, (F1.ID2 || "-" || F2.ID2) AS frRel1, (F2.ID2 || "-" || F1.ID2) AS frRel2
    FROM Friend AS F1, Friend AS F2
    WHERE F1.ID1 = F2.ID1 AND F1.ID2 < F2.ID2),

Highschooler AS FrA, Highschooler AS FrB, Highschooler AS FrC

WHERE likeRel = frRel1 OR likeRel = frRel2 AND FrA.ID = A AND FrB.ID = B AND FrC.ID = C
```

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

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

Q8

1.0/1.0 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

Correct

Correct

Your Query Result:

2

Expected Query Result:

2

Submit

Q9

1.0/1.0 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 name, grade
FROM Highschooler, (SELECT ID2 FROM Likes GROUP BY ID2 HAVING COUNT(*) > 1)
WHERE ID = ID2
```

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

Correct

Correct

Your Query Result:

Cassandra	9
Kris	10

Expected Query Result:

Cassandra	9
Kris	10

Submit