Q1. Given the schema below, answer the following questions.

Schema

Student(<u>sID</u>, surName, firstName, campus, email, cgpa)
Course(<u>dept, cNum</u>, name, breadth)
Offering(<u>oID</u>, dept, cNum, term, instructor)
Took(<u>sID, oID</u>, grade)

Offering[dept, cNum] ⊆ Course[dept, cNum]
Took[sID] ⊆ Student[sID]
Took[oID] ⊆ Offering[oID]

1. Answer each of the following questions with an arithmetic expression.
   Suppose a row occurs n times in table R and m times in table S.
   a. Using bag semantics, how many times will it occur in table R ∪ S?

      Soln:
      m+n
      Recall that bag semantics allows for duplicate entries/tuples.
      In bag semantics, {1, 1, 1, 3, 7, 7, 8} ∪ {1, 5, 7, 7, 8, 8} = {1, 1, 1, 1, 3, 5, 7, 7, 7, 7, 8, 8, 8}.
      Hence, if a row occurs n times in table R and m times in table S, when we take the union of R and S, we get all those rows.
      Hence, it will occur m+n times.

   b. Using bag semantics, how many times will it occur in table R ∩ S?

      Soln:
      min(m, n)
      In bag semantics, {1, 1, 1, 3, 7, 7, 8} ∩ {1, 5, 7, 7, 8, 8} = {1, 7, 7, 8}.
      This shows that for intersection, we always take the least amount of occurrences in either set.
      Hence, it will occur min(m, n) times.

   c. Using bag semantics, how many times will it occur in table R − S?

      Soln:
      max(m-n, 0)
      In bag semantics, {1, 1, 1, 3, 7, 7, 8} − {1, 5, 7, 7, 8, 8} = {1, 1, 3}.
      Hence, if the row exists in R, we have m-n occurrences.
      If the row does not exist in R, we have 0 occurrences.
      Hence, it will occur max(m-n, 0) times.

2. Use a set operation to find all terms when Jepson and Suzuki were both teaching.
   Include every occurrence of a term from the result of both operands.

   Soln:

```
Answer(term) :=
        (SELECT term FROM Offering WHERE instructor = "Jepson")
                INTERSECT ALL
        (SELECT term FROM Offering WHERE instructor = "Suzuki");
```

3. Find the sID of students who have earned a grade of 85 or more in some course, or who have passed a course taught by Atwood. Ensure that no sID occurs twice in the result.

   Soln:

```
(SELECT sID FROM Took WHERE grade >= 85)
        UNION
(SELECT sID FROM Took NATURAL JOIN Offering WHERE grade>=50
AND instructor = “Atwood”);
```

4. Find all terms when csc369 was not offered.

   Soln:

```
(SELECT term FROM Offering)
        EXCEPT
(SELECT term FROM Offering WHERE dept=“CSC” AND cNUM=369);
```

5. Make a table with two columns: oID and results. In the results column, report either “high” (if that offering had an average grade of 80 or higher), or “low” (if that offering had an average under 60). Offerings with an average in between will not be included.

   Soln:

```
(SELECT oID, “high” AS results FROM Took GROUP BY oID HAVING
avg(grade) >= 80)
        UNION
(SELECT oID, “low” AS results FROM Took GROUP BY oID HAVING
avg(grade) < 60);
```

Q2. Given the schema below, answer the following questions.

Schema

Student(<u>sID</u>, surName, firstName, campus, email, cgpa)
Course(<u>dept, cNum</u>, name, breadth)
Offering(<u>oID</u>, dept, cNum, term, instructor)
Took(<u>sID, oID</u>, grade)

Offering[dept, cNum] ⊆ Course[dept, cNum]
Took[sID] ⊆ Student[sID]
Took[oID] ⊆ Offering[oID]

1. Write a query to find the average grade, minimum grade, and maximum grade for each offering.

   Soln:

   Max grade:

```
SELECT max(grade) FROM Took GROUP BY oID;
```

   Min grade:

```
SELECT min(grade) FROM Took GROUP BY oID;
```

   Average grade:

```
SELECT avg(grade) FROM Took GROUP BY oID;
```

2. Which of these queries is legal?

```
SELECT surname, sid
FROM Student, Took
WHERE Student.sid = Took.sid
GROUP BY sid;
```

```
SELECT surname, Student.sid
FROM Student, Took
WHERE Student.sid = Took.sid
GROUP BY campus;
```

```
SELECT instructor, max(grade),
    count(Took.oid)
FROM Took, Offering
WHERE Took.oid = Offering.oid
GROUP BY instructor;
```

```
SELECT Course.dept, Course.cnum,
    count(oid), count(instructor)
FROM Course, Offering
WHERE Course.dept = Offering.dept and
    Course.cnum = Offering.cnum
GROUP BY Course.dept, Course.cnum
ORDER BY count(oid);
```

Soln:
SELECT instructor, max(grade), count(Took.oid)
FROM Took, Offering
WHERE Took.oid = Offering.oid
GROUP BY instructor;
and
SELECT Course.dept, Course.cnum, count(oid), count(instructor)
FROM Course, Offering
WHERE Course.dept = Offering.dept and Course.cnum = Offering.cnum
GROUP BY Course.dept, Course.cnum
ORDER BY count(oid);
are legal.
Note: For the other 2 queries, you cannot get the column(s) that you are not grouping by or using an aggregate function.

For
SELECT surname, sid
FROM Student, Took
WHERE Student.sid = Took.sid
GROUP BY sid;
you cannot select surname unless you use an aggregate function with it or you also group by it.

For
SELECT surname, Student.sid
FROM Student, Took
WHERE Student.sid = Took.sid
GROUP BY campus;
you cannot get surname or Student.sid unless you use an aggregate function with it or you also group by it.

3. Find the sid and minimum grade of each student with an average over 80.

   Soln:
   SELECT sID, min(grade) FROM Took GROUP BY sID HAVING avg(grade) > 80;

4. Find the sid, surname, and average grade of each student, but keep the data only for those students who have taken at least 10 courses.

   Soln:
   SELECT sID, surName, avg(grade) FROM Took NATURAL JOIN Offering NATURAL JOIN Student GROUP BY sID, surName HAVING count(oID) >= 10;

5. For each student who has passed at least 10 courses, report their sid and average grade on the courses that they passed.

   Soln:
   // This first query creates a view that contains the sID of all students who have
   // passed at least 10 courses.
   CREATE VIEW students_who_passed_at_least_10_courses(sID) AS
   SELECT sID FROM Took WHERE grade >= 50 GROUP BY sID
   HAVING count(grade) >= 10;

   // This second query creates a view that contains the sID of all students who have
   // passed at least 10 courses as well as the oID of the courses they passed.
   CREATE VIEW courses_passed(sID, oID) AS
   SELECT sID, oID FROM Took NATURAL JOIN
   students_who_passed_at_least_10_courses;

   // This third query gets the sID of all students who have passed at least 10 courses as
   // well as the average grade of the courses they passed.
   SELECT sID, avg(grade) FROM Took NATURAL JOIN courses_passed
   GROUP BY sID;

6. For each student who has passed at least 10 courses, report their sid and average grade on all of their courses.

   Soln:
   // This first query creates a view of all the students who passed at least
   // 10 courses.
   CREATE VIEW students_who_passed_at_least_10_courses(sID) AS
   SELECT sID FROM Took WHERE grade >= 50 GROUP BY sID
   HAVING count(grade) >= 10;

   // This second query gets the sID and average grade of all students who passed
   // at least 10 courses.
   SELECT sID, avg(grade) FROM Took NATURAL JOIN
   students_who_passed_at_least_10_courses GROUP BY sID;

7. Which of these queries is legal?

```
SELECT dept
FROM Took, Offering
WHERE Took.oID = Offering.oID
GROUP BY dept
HAVING avg(grade) > 75;
```

```
SELECT Took.oID, avg(grade)
FROM Took, Offering
WHERE Took.oID = Offering.oID
GROUP BY Took.oID
HAVING avg(grade) > 75;
```

```
SELECT Took.oID, dept, cNum, avg(grade)
FROM Took, Offering
WHERE Took.oID = Offering.oID
GROUP BY Took.oID
HAVING avg(grade) > 75;
```

```
SELECT oID, avg(grade)
FROM Took
GROUP BY sID
HAVING avg(grade) > 75;
```

Soln:

```
SELECT dept
FROM Took, Offering
WHERE Took.oID = Offering.oID
GROUP BY dept
HAVING avg(grade) > 75;
```

```
SELECT Took.oID, avg(grade)
FROM Took, Offering
WHERE Took.oID = Offering.oID
GROUP BY Took.oID
HAVING avg(grade) > 75;
```

Q3. Given the schema below, answer the following questions.

Schema

Student(sID, surName, firstName, campus, email, cgpa)

Course(dept, cNum, name, breadth)

Offering(oID, dept, cNum, term, instructor)

Took(sID, oID, grade)

Offering[dept, cNum] ⊆ Course[dept, cNum]

Took[sID] ⊆ Student[sID]

Took[oID] ⊆ Offering[oID]

1. Which of these queries is legal?

(a)
```
SELECT count(distinct dept), count(distinct instructor)
FROM Offering
WHERE term >= 20089;
```

(b)
```
SELECT distinct dept, distinct instructor
FROM Offering
WHERE term >= 20089;
```

(c)
```
SELECT distinct dept, instructor
FROM Offering
WHERE term >= 20089;
```

Soln:

a and c are both legal.
b is not legal because the keyword “distinct” is repeated.
Distinct cannot be used twice.

2. Under what conditions could these two queries give different results? If that is not possible, explain why.

```
SELECT surName, campus                    SELECT distinct surName, campus
FROM Student;                             FROM Student;
```

Soln:
If there are multiple rows with the same surName and campus, then the first query will list all of them while the second query only lists one of them.

3. For each student who has taken a course, report their sid and the number of different departments they have taken a course in.

Soln:
SELECT sID, count(DISTINCT dept) FROM Course NATURAL JOIN Took GROUP BY sID;

4. Suppose we have two tables with content as follows:

```
SELECT *                                  SELECT *
FROM One;                                 FROM Two;

 a  |  b                                     b  |  c
----+-----                                -----+-----
  1 |   2                                   2 |   3
  6 |  12                                 100 | 101
    | 100                                  20 |  21
 20 |                                       2 |   4
(4 rows)                                    2 |   5
                                          (5 rows)
```

a. What query could produce this result?

```
 a |   b  |   c
---+------+-----
 1 |    2 |   3
 1 |    2 |   4
 1 |    2 |   5
   |   20 |  21
   |  100 | 101
(5 rows)
```

Soln:
SELECT * FROM One RIGHT JOIN Two;

b. What query could produce this result?

```
 a  |  b  |  c
----+-----+-----
  1 |   2 |   3
  1 |   2 |   4
  1 |   2 |   5
  6 |  12 |
    | 100 | 101
 20 |     |
(6 rows)
```

Soln:
SELECT * FROM One LEFT JOIN Two;

Q4. Given the schema below, answer the following questions.

Schema

Student(sID, surName, firstName, campus, email, cgpa)
Course(dept, cNum, name, breadth)
Offering(oID, dept, cNum, term, instructor)
Took(sID, oID, grade)

Offering[dept, cNum] ⊆ Course[dept, cNum]
Took[sID] ⊆ Student[sID]
Took[oID] ⊆ Offering[oID]

1. What does this query do? (Recall that the || operator concatenates two strings.)

```
SELECT sid, dept || cnum as course, grade
FROM Took,
        (SELECT *
         FROM Offering
         WHERE instructor = 'Horton') Hoffering
WHERE Took.oid = Hoffering.oid;
```

Soln:
The query gets all the students who have taken some courses with instructor Horton and all the courses taught by instructor Horton.

2. What does this query do?

```
SELECT sid, surname
FROM Student WHERE cgpa >
        (SELECT cgpa
        FROM Student
        WHERE sid = 99999);
```

Soln:
This gets the sid and surname of all students who have a cgpa greater than the cgpa of the student with the sid 99999.

3. What does this query do?

```
SELECT sid, dept || cnum AS course, grade
FROM Took JOIN Offering ON Took.oid = Offering.oid
WHERE
        grade >= 80 AND
        (cnum, dept) IN (
                SELECT cnum, dept
                FROM Took JOIN Offering ON Took.oid = Offering.oid
                                JOIN Student ON Took.sid = Student.sid
                WHERE surname = 'Lakemeyer');
```

Soln:
The inner query gets all the courses taken by students with the surname Lakemeyer. The outer query gets the sids of all students who have gotten a mark of 80 or higher in a course taken by students with the surname Lakemeyer as well as the courses.

4.

a. Suppose we have these relations: R(a, b) and S(b, c). What does this query do?

```
SELECT a
FROM R
WHERE b in (SELECT b FROM S);
```

Soln:
This query gets you all the a's from relation R whose corresponding b value is also in S.

b. Can we express this query without using subqueries?

Soln:
Yes. The query is given below.

```
SELECT a from R NATURAL JOIN S;
```

5. What does this query do?

```
SELECT instructor
FROM Offering Off1
WHERE NOT EXISTS (
        SELECT *
        FROM Offering
        WHERE
                oid <> Off1.oid AND
                instructor = Off1.instructor);
```

Soln:

The inner query gets all offerings that are not Off1.oid but both have the same instructor.
I.e. The inner query is essentially that an instructor has taught multiple courses.
The outer query gets all instructors who did not teach multiple courses.
I.e. The outer query gets all instructors who only taught one course.