Question 1. [11 MARKS]

Recall this schema, which we have used many times in class. Here we are adding one more relation called *Program*. It records the subject POSts that students are enrolled in. ("POSt" is short for "program of study", by the way.)

Relations

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

Course(dept, cNum, name, breadth)

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

Took(sID, oID, grade)

Program(sID, POSt)

Integrity constraints

Offering[dept, cNum] $\subseteq$ Course[dept, cNum]

Took[sID] $\subseteq$ Student[sID]

Took[oID] $\subseteq$ Offering[oID]

Program[sID] $\subseteq$ Student[sID]

Part (a) [7 MARKS]

Write a query to find the sID of every student who either (a) has a POSt that no student from the UTM campus has or (b) has at least two POSts. Use only the basic operators $\Pi, \sigma, \bowtie, \times, \cap, \cup, -, \rho$, and assignment.

Solution:

$$NoUTM(POSt) := \Pi_{POSt}Program - \Pi_{POSt}(Program \bowtie \sigma_{campus=\text{`}UTM\text{'}}Student)$$
$$HasRare(sID) := \Pi_{sID}(Program \bowtie NoUTM)$$
$$TwoPOSts(sID) := \Pi_{P1.sID}\sigma_{P1.sID=P2.sID \wedge P1.POSt \neq P2.POSt}(\rho_{P1}Program \times \rho_{P2}Program)$$
$$Answer(sID) := HasRare \cup TwoPOSts$$

Part (b) [4 MARKS]

Consider the following query:

$$One(dept, cNum, term) := \Pi_{O1.dept, O1.cNum, O1.term}\sigma_{O1.dept=O2.dept \wedge O1.cNum=O2.cNum \wedge O1.term<O2.term}(\rho_{O1} Offering \times \rho_{O2} Offering)$$

$$Two(dept, cNum, term) := (\Pi_{dept, cNum, term} Offering) - One$$

$$Answer(instructor) := \Pi_{instructor}(Two \bowtie Offering)$$

1. Below is an instance of the only relation that is relevant to this query, *Offering*. Add the fewest possible rows to *Offering* so that professors Able and Bland will not appear in the result of the query, but professors Cranky and Devlish will.

Offering:

oID	dept	cNum	term	instructor
O3	CSC	324	1	Bland
O6	CSC	324	4	Able
O1	CSC	443	2	Eager
O2	CSC	324	2	Bland
O7	CSC	148	3	Devlish
O9	CSC	148	2	Cranky

Solution:

- Professor Able is currently in the result, so we have to do something to remove her. We can do this by adding a later offering of CSC324. This doesn't change whethere or not anyone else is in the result.
- Professor Bland is not in the result, so we don't need to do anything with him.
- Professor Cranky is not in the result, so we need to have her teach the last offering of something, in order to get her in the result. We have to be careful how we do this, so we don't affect other instructors in the wrong way.
- Professor Devlish needs is already in the result, so there is no need to do anything with him.

In total, we only need to add 2 rows. Here is a complete solution:

Offering:

oID	dept	cNum	term	instructor
O3	CSC	324	1	Bland
O6	CSC	324	4	Able
O1	CSC	443	2	Eager
O2	CSC	324	2	Bland
O7	CSC	148	3	Devlish
O9	CSC	148	2	Cranky
O10	CSC	324	5	Eager
O11	CSC	148	3	Cranky

2. What does this query compute? Do not describe the steps it takes, only what is in the result, and make your answer general to any instance of the schema.

Solution:
All instructors who have taught a course in the last term in which it was offered.

Question 2. [6 marks]

Part (a) [2 marks]

In the previous question, we introduced a new relation called *Program* to record information about students' POSts. Does our schema enforce the following constraint:

Every student has at least one POSt.

Circle one answer. If the statement is enforced, say what part of the schema enforces it. If it is not enforced, write an integrity constraint that would enforce it, using the form $R = \emptyset$.

Enforced This part of the schema enforces it:

[✓ Not enforced] This new integrity constraint would enforce it:

$(\Pi_{sID}Student - \Pi_{sID}Program) = \emptyset$

Part (b) [4 marks]

Consider this schema:

R(<u>one</u>, two, three)	R[three] $\subseteq$ T[seven]
S(<u>four</u>, five, six)	S[four] $\subseteq$ T[seven]
T(<u>seven</u>, eight)	

Suppose relation R has 100 tuples. How many tuples could T have? Circle all answers that do not violate the schema.

0 $\boxed{1}$ $\boxed{82}$ $\boxed{100}$ $\boxed{101}$

Suppose relation S has 100 tuples. How many tuples could T have? Circle all answers that do not violate the schema.

0 1 82 $\boxed{100}$ $\boxed{101}$

Question 3. [5 MARKS]

The question refers to the schema from Question 1. Write a query in SQL to find the maximum grade given in every course, across all offerings. For each, report the name of the department, the course number, and the maximum grade given in that course in any offering of that course

Solution:

```
SELECT dept, cNum, max(grade)
FROM Offering, Took
WHERE Offering.oID = Took.oID
GROUP BY dept, cNum
```

Question 4. [8 MARKS]

Part (a) [3 MARKS]

Consider the same schema from the Question 1. Suppose we wrote the query

```
SELECT _______________________
FROM Student, Took
WHERE Student.sID = Took.sID
GROUP BY Took.sID;
```

Which of the following could go in the `SELECT` clause? Circle all that apply.

Solution:

`sID` `oID` [✓ `count(oID)-2`] [✓ `Took.sID`] [✓ `avg(grade)`] [✓ `max(cgpa)`] `cgpa`

Here are the error messages for the problematic ones:

```
csc343h-dianeh=> select sid
csc343h-dianeh-> FROM Student, Took
csc343h-dianeh-> WHERE Student.sID = Took.sID
csc343h-dianeh-> GROUP BY Took.sID;
ERROR:  column reference "sid" is ambiguous
LINE 1: select sid
               ^
csc343h-dianeh=> select oid
csc343h-dianeh-> FROM Student, Took
csc343h-dianeh-> WHERE Student.sID = Took.sID
csc343h-dianeh-> GROUP BY Took.sID;
ERROR:  column "took.oid" must appear in the GROUP BY clause or be used in an aggregate function
LINE 1: select oid
               ^
csc343h-dianeh=> select cgpa
csc343h-dianeh-> FROM Student, Took
csc343h-dianeh-> WHERE Student.sID = Took.sID
csc343h-dianeh-> GROUP BY Took.sID;
ERROR:  column "student.cgpa" must appear in the GROUP BY clause or be used in an aggregate functi
LINE 1: select cgpa
               ^
```

Part (b) [3 MARKS]

We discussed in lecture how the SQL subquery operators could possibly be implemented using other SQL operations. Suppose we have two tables `R(a,b)` and `S(b,c)`. Note that their keys are underlined.

Consider the following two queries:

```
-- Query 1
SELECT a AS answer
FROM R
WHERE b > SOME (SELECT b FROM S);

-- Query 2
SELECT R.a AS answer
FROM R, S
WHERE R.b > S.b;
```

On the next page, give a **database instance** where these two queries produce *different* results, and the **results of the two queries**.

Solution:

```
insert into R values
(1, 10),
(2, 20),
(3, 30);

insert into S values
(25, 5),
(26, 10);

-- Query 1 gives:
 answer
--------
      3
(1 row)


-- Query 2 gives:
 answer
--------
      3
      3
(2 rows)
```

Part (c) [2 marks]

Without changing the FROM clause, fix Query 2 so that it produces the same results as Query 1 for any valid dataset. Do not use subqueries or views.

Solution:

```
SELECT DISTINCT R.a AS answer
FROM R, S
WHERE R.b > S.b;
```

```
-- Query 3 gives:
 answer
--------
      3
(1 row)
```