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 cNum of every CSC course that has been taught in the term whose value is 20129, and in every term after 20129. (You may use comparison operators such as $<$ to compare terms.) Write your query using only the basic operators $\Pi, \sigma, \bowtie, \times, \cap, \cup, -, \rho$, and assignment.

Solution:

$ShouldHaveOffered(cNum, term) := (\Pi_{cNum}\sigma_{dept='CSC'}Course) \times (\Pi_{term}\sigma_{term \geq 20129}Offering)$

$DidOffer(cNum, term) := \Pi_{cNum,term}\sigma_{dept='CSC'}Offering$

$Missed(cNum, term) := ShouldHaveOffered - DidOffer$

$Answer(cNum) := (\Pi_{cNum}Course) - (\Pi_{cNum}Missed)$

Part (b) [4 MARKS]

Consider the following query:

$Temp1(instructor, oID, term) := \Pi_{instructor,oID,term}\sigma_{dept='CSC'}\, Offering$

$Temp2(instructor) :=$
$\Pi_{O1.instructor}\sigma_{O1.instructor=O2.instructor \wedge O1.oID \neq O2.oID \wedge O1.term=O2.term}(\rho_{O1}\, Temp1 \times \rho_{O2}\, Temp1)$

$Answer(instructor) := (\Pi_{instructor}\, Temp1) - Temp2$

1. Below is an instance of the 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	Able
O6	CSC	324	4	Able
O1	CSC	443	5	Devlish
O7	CSC	148	2	Bland
O8	ECE	345	2	Bland
O9	ANT	101	6	Cranky

   Solution:

   - Professor Able is currently in the result, so we have to do something to remove her. We can do this by adding another CSC course in the same term. This doesn't change whether or not anyone else is in the result.
   - Same with Professor Bland.
   - Professor Cranky is not in the result, so we need to have her teach one offering of a CSC course, in order to get her in the result. Again, this won't affect other instructors.

- 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 3 rows. Here is a complete solution:

Offering:

oID	dept	cNum	term	instructor
O3	CSC	324	1	Able
O6	CSC	324	4	Able
O1	CSC	443	5	Devlish
O7	CSC	148	2	Bland
O8	ECE	345	2	Bland
O9	ANT	101	6	Cranky
O10	CSC	100	1	Able
O11	CSC	100	2	Blank
O12	CSC	100	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:
Find the instructors who have taught a CSC course but never two CSC offerings in the same term.

Question 2. [6 MARKS]

Part (a) [2 MARKS]

Does our schema enforce the following constraint:

Every course has been offered.

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:

$\sigma_{dept,cNum} Course - \sigma_{dept,cNum} Offering = \emptyset$

Part (b) [4 MARKS]

Consider this schema:

A(alpha, beta, gamma)
B(delta, epsilon)
C(iota, kappa)

B[delta] $\subseteq$ A[beta]
B[epsilon] $\subseteq$ C[iota]

Suppose relation B has 50 tuples. How many tuples could A have? Circle all answers that do not violate the schema.

0 1 23 **50** **128**

Suppose relation B has 50 tuples. How many tuples could C have? Circle all answers that do not violate the schema.

0 **1** **23** **50** **128**

Question 3. [5 MARKS]

The question refers to the schema from Question 1. Write a query in SQL to find the terms in which at least twenty-five grades greater than 90 were given. Note that one student can be given multiple grades greater than 90 in a single term. Report only the terms.

Solution:

```
SELECT term
FROM Took, Offering
WHERE grade > 90 AND Took.oID = Offering.oID
GROUP BY term
HAVING count(grade) >= 100
```

Question 4. [8 MARKS]

Part (a) [3 MARKS]

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

```
SELECT _________________________
FROM Offering, Course
WHERE Offering.dept = Course.dept AND Offering.cNum = Course.cNum
GROUP BY Offering.dept;
```

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

`count(term)` `Offering.cNum` `Offering.dept` `avg(cNum)`

`Course.dept` `count(Course.cNum)` `dept`

Solution: `count(term)`, `Offering.dept`, `count(Course.cNum)`

```
csc343h-dianeh=> select count(term)
csc343h-dianeh-> FROM Offering, Course
csc343h-dianeh-> WHERE Offering.dept = Course.dept AND Offering.cNum = Course.cNum
csc343h-dianeh-> GROUP BY Offering.dept;
 count
-------
     3
    18
     4
     6
     2
     3
(6 rows)

csc343h-dianeh=> select offering.cnum
csc343h-dianeh-> FROM Offering, Course
csc343h-dianeh-> WHERE Offering.dept = Course.dept AND Offering.cNum = Course.cNum
csc343h-dianeh-> GROUP BY Offering.dept;
ERROR:  column "offering.cnum" must appear in the GROUP BY clause or be used in an aggregate funct
LINE 1: select offering.cnum
                ^
csc343h-dianeh=>
csc343h-dianeh=> select offering.dept
csc343h-dianeh-> FROM Offering, Course
csc343h-dianeh-> WHERE Offering.dept = Course.dept AND Offering.cNum = Course.cNum
csc343h-dianeh-> GROUP BY Offering.dept;
 dept
------
 ANT
 CSC
```

```
 EEB
 ENG
 ENV
 HIS
(6 rows)

csc343h-dianeh=> select avg(cnum)
csc343h-dianeh-> FROM Offering, Course
csc343h-dianeh-> WHERE Offering.dept = Course.dept AND Offering.cNum = Course.cNum
csc343h-dianeh-> GROUP BY Offering.dept;
ERROR:  column reference "cnum" is ambiguous
LINE 1: select avg(cnum)
                   ^
csc343h-dianeh=> select course.dept
csc343h-dianeh-> FROM Offering, Course
csc343h-dianeh-> WHERE Offering.dept = Course.dept AND Offering.cNum = Course.cNum
csc343h-dianeh-> GROUP BY Offering.dept;
ERROR:  column "course.dept" must appear in the GROUP BY clause or be used in an aggregate functio
LINE 1: select course.dept
                ^
csc343h-dianeh=> select count(course.cnum)
csc343h-dianeh-> FROM Offering, Course
csc343h-dianeh-> WHERE Offering.dept = Course.dept AND Offering.cNum = Course.cNum
csc343h-dianeh-> GROUP BY Offering.dept;
 count
-------
     3
    18
     4
     6
     2
     3
(6 rows)

csc343h-dianeh=> select dept
csc343h-dianeh-> FROM Offering, Course
csc343h-dianeh-> WHERE Offering.dept = Course.dept AND Offering.cNum = Course.cNum
csc343h-dianeh-> GROUP BY Offering.dept;
ERROR:  column reference "dept" is ambiguous
LINE 1: select dept
```

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)`.

Consider the following two queries:

```
-- Query 1
```

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

-- Query 2
SELECT R.a
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**. Clearly label which result belongs to which query!

Solution: Any instance where a tuple `(a,b)` in R has the `b` value appear twice in `S`. The first query will produce just one occurrence of `a`, while the second produces two occurrences of `a`.

For example,

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

insert into S values
(10, 5),
(10, 10);
```

Part (c) [2 MARKS]

Fix Query 2 so that it produces the same results as Query 1 for any valid dataset. Your answer must still include a Cartesian product with `R`. You may not use the keywords `ANY, ALL, IN, EXISTS` in your solution. **Hint**: create a view.

Solution:

```
CREATE VIEW Bs AS
SELECT DISTINCT b
FROM S;

SELECT R.a
FROM R, Bs
WHERE R.b = S.b;
```

(This can also be done by making the `SELECT DISTINCT ...` a subquery of the `SELECT` inside the `FROM` clause.)