Recall this schema, which we have used many times in class.

Question 1. [5 marks]

Part (a) [1 mark]

According to the schema, can a student take the same course more than once?

Solution: $\boxed{\text{Yes}}$ No.

Part (b) [2 marks]

Consider this constraint:

$Ploot(instructor1, instructor2, term) :=$

$$\Pi_{O1.instructor1, O2.instructor2, O1.term}\sigma_{\substack{O1.instructor<O2.instructor \\ \wedge \\ O1.dept=O2.dept=\text{`}CSC\text{'} \\ \wedge \\ O1.cNum=O2.cNum \\ \wedge \\ O1.term=O2.term}}[(\rho_{O1}\mathit{Offering}) \times (\rho_{O2}\mathit{Offering})]$$

$$\sigma_{\substack{P1.instructor1=P2.instructor1 \\ \wedge \\ P1.instructor2=P2.instructor2 \\ \wedge \\ P1.term \neq P2.term}}[(\rho_{P1}Ploot) \times (\rho_{P2}Ploot)] = \emptyset$$

Define an instance of Offering that violates the constraint.

Solution:

oID	dept	cNum	term	instructor
o1	CSC	343	termA	Craig
o2	CSC	343	termA	Horton
o3	CSC	108	termB	Craig
o4	CSC	108	termB	Horton

Part (c) [2 marks]

Write the following constraint using relational algebra: No CSC course may count towards the breadth requirement (that is, have breadth = True) unless it is a 100-level course.

Solution: $\sigma_{dept=\text{`}CSC\text{'} \wedge breadth \wedge cNum>199}Course = \emptyset$

Question 2. [8 MARKS]

Write a query in relational algebra to find the sID of every student who has taken each CSC course that has ever been offered.

Solution: this is one possible way to solve this
– This student has taken this CSC course.

$Taken(sID, cNum) := \Pi_(sID, cNum)\sigma_{dept='CSC'}[Offering \bowtie Took]$

– All CSC courses ever offered

$Offered(cNum) := \Pi_{cNum}\sigma_{dept=`CSC'}Offering$

– All students

$AllStudent(sID) = \Pi_{sID}Student$

– Combo of every student and every CS courses

$Checklist(sid, cNum) := AllStudent \times Offered$

– Students who didn't take them all

$MissedSome(sID) := \Pi_{sID}[Checklist - Taken]$

– Took all

$Solution(sID) := AllStudent - MissedSome$

Question 3. [5 MARKS]

Suppose we want to find the cNum of the first CSC course that was ever taught, that is, the one with the minimum value for term. If there was a tie, we want to report the cNums of all the tied courses.

The following query attempts to solve this. It is syntactically correct, but doesn't always produce the right answer.

Solution:

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

$NotFirst(cNum, term) := \Pi_{C2.cNum,C2.term}\sigma_{C1.term<C2.term}[(\rho_{C1}CSCterms) \times (\rho_{C2}CSCterms)]$

$Answer(cNum) := \Pi_{cNum}[CSCterms) - NotFirst]$

Part (a) [2 MARKS]

Define an instance of Offering with 4 rows on which the query gives the wrong answer. For simplicity, use integers to represent the terms.

Solution:

oID	dept	cNum	term	instructor
o1	CSC	343	1	Craig
o2	CSC	343	3	Horton
o3	CSC	209	4	Craig
o4	CSC	148	2	Horton

What relation *should be* produced in this case?

Solution:

cNum
343

What relation *is* produced in this case?

Solution:

cNum
(empty table)

Part (b) [3 MARKS]

On the query above, make the smallest change(s) that will correct it.

Solution: Add term to the attributes of NotFirst. It will need to be added to both the left-hand side and to the project statement. Then do the subtraction in the RHS of answer before the project.

Question 4. [7 MARKS]

Consider this new schema for a music industry database:

Relations

Musician(<u>mID</u>, surName, firstName, birthdate)

Album(<u>aID</u>, title, mID, year)

RecordCompany(<u>cID</u>, name, president)

Produced(<u>aID, cID</u>)

Integrity constraints

Album[mID] $\subseteq$ Musician[mID]

Produced[aID] $\subseteq$ Album[aID]

Produced[cID] $\subseteq$ RecordCompany[cID]

Part (a) [1 MARK]

Suppose relation *Album* has 1,000 tuples. How many tuples could *Produced* have? Circle all that apply:

Solution:

[✓ 0] [✓ 1] [✓ 821] [✓ 1,000] [✓ 2,500]

Part (b) [2 MARKS]

Which of the following constraints are enforced by the schema? Circle Yes or No for each.

Solution:

Every musician has at least one album.	Yes	[✓ No]
Every album has at least one record company that produced it.	Yes	[✓ No]
Every record company has produced at least one album.	Yes	[✓ No]
Every album has at most one record company that produced it.	Yes	[✓ No]
Every album has at most one musician.	[✓ Yes]	No

Part (c) [2 MARKS]

Suppose every album has one genre, and we add an attribute called genre to the Produced relation to keep track of it. Write a constraint in relational algebra to restrict the value of genre to either hip hop, pop, or country.

Solution:

$\sigma_{genre \neq 'hiphop' \wedge genre \neq 'pop' \wedge genre \neq 'country'} Produced = \emptyset$
OR Produced[genre] $\subseteq$ {'hip hop','pop','country'}

Part (d) [2 MARKS]

Why is it a bad idea to store this genre information in the Produced relation?

Solution:

The same album can have multiple rows or zero rows in Produced. The genre information will be repeated in all of them or be missing. This is unnecessary duplication. The information should go in the Album relation.

Question 5. [8 MARKS]

For this question, you will write SQL queries using a simplified version of the Instagram schema from Assignment 1. (We removed unnecessary pieces and renamed table `User` to `Account` because `user` is a reserved word in SQL.)

Relations

Account(uID, name, website, phone)

Follows(follower, followed)

Post(pid, uid, location, caption)

Hashtag(pid, tag)

Integrity constraints

Follows[follower] $\subseteq$ Account[uID]

Follows[followed] $\subseteq$ Account[uID]

Post[uID] $\subseteq$ Account[uID]

Hashtag[pID] $\subseteq$ Post[pID]

Part (a) [3 MARKS]

Write a query in SQL to find users that have more than 100 posts where the location is Toronto. For each one, give their uID and the total number of their Toronto posts. Show the users in non-increasing order by their number of Toronto posts.

Solution:

```
SELECT uid, count(pid)
FROM Post
WHERE location = 'Toronto'
GROUP BY uid
HAVING count(pid) > 100
ORDER BY count(pid) DESC;
```

Part (b) [1 MARK]

Write an SQL query to find the pid of posts that have no hashtags.

Solution:

```
(SELECT pid FROM Post) EXCEPT (SELECT pid FROM Hashtag);
```

The following query is supposed to print the name, uID and number of followers for users who have fewer than three followers. It runs but does not always give the correct output.

```
SELECT name, uid, count(follower) AS num_followers

FROM Follows, Account

WHERE Follows.followed = Account.uid

GROUP BY Account.uid

HAVING count(follower) < 3;
```

Part (c) [2 marks]

Suppose that Follows and Account have these values. What will be the output of the query?

```
 follower | followed
----------+----------
        2 |        1
        3 |        2
        2 |        3
        4 |        2
        1 |        3
        4 |        3
```

```
 uid | name  | website
-----+-------+----------
   1 | user1 | website1
   2 | user2 | website2
   3 | user3 | website3
   4 | user4 | website4
```

Solution:

```
name   | uid | num_followers
-------+-----+---------------
 user2 |   2 |             2
 user1 |   1 |             1
```

Part (d) [1 mark]

Generalizing to any dataset, explain what is wrong with the output of this query.

Solution: It doesn't include users who have no followers at all.

Part (e) [1 mark]

Fix the query by making the smallest change that you can. Write your corrections directly on the query text above.

Solution: One option is:

```
SELECT name, uid, count(follower) AS num_followers
FROM Follows RIGHT JOIN Account
ON Follows.followed = Account.uid
GROUP BY Account.uid
HAVING count(follower) < 3;
```

Another is to do ... `FROM Account LEFT JOIN Follows ON ...`

Question 6. [8 marks]

Suppose we have these tables:

```
Follows:
     a     |       b
----------+----------------
 sina     | kanyewest
 sina     | RonConwayFacts
 diane    | LilaFontes
 diane    | swcarpentry
 diane    | mfeathers
 diane    | sina
 michelle | sina
 michelle | diane
 michelle | Jeff
(9 rows)
```

```
Profile:
    id    |    name    | location
----------+------------+----------
 alan     | catman     | Ottawa
 sina     | superman   |
 diane    | superwoman | Toronto
 michelle | rockstar   | Montreal
(4 rows)
```

```
Tweets:
id  | userid |    content
-----+--------+----------------
 123 | alan   | hellow twitter
 125 | alan   | bye twitter
 126 | alan   | hellow twitter
 128 | alan   | bye twitter
 476 | sina   | hellow twitter
 553 | diane  | hellow twitter
(6 rows)
```

Show the result of running each of the following queries. If a table is produced, include the column names. If the query generates an error, explain.

Solutions

```
SELECT count(*)
FROM Profile LEFT JOIN Follows
    ON a = id;


-- Output:

 count
-------
    10
(1 row)
```

```
SELECT P.id, count(T.content) AS number
FROM Profile P JOIN  Tweets t On
    T.userid = P.id  AND P.location='Toronto'
GROUP BY(p.id);\\
-- Output:

 id   | number
-------+--------
 diane |      1
(1 row)
```

Here are the tables again, for easy reference:

```
Follows:
    a     |       b
----------+----------------
 sina     | kanyewest
 sina     | RonConwayFacts
 diane    | LilaFontes
 diane    | swcarpentry
 diane    | mfeathers
 diane    | sina
 michelle | sina
 michelle | diane
 michelle | Jeff
(9 rows)
```

```
Profile:
   id     |    name    |
----------+------------+
 alan     | catman     |
 sina     | superman   |
 diane    | superwoman |
 michelle | rockstar   |
(4 rows)
```

```
Tweets:
id  | userid |    content
-----+--------+----------------
 123 | alan   | hellow twitter
 125 | alan   | bye twitter
 126 | alan   | hellow twitter
 128 | alan   | bye twitter
 476 | sina   | hellow twitter
 553 | diane  | hellow twitter
(6 rows)
```

```
SELECT id, count(b) AS followers
FROM Profile  JOIN Follows
    ON a = id ;




-- Output:

ERROR:  column "profile.id" must appear in the
GROUP BY clause or be used in an aggregate
function
LINE 1: SELECT id, count(b) AS followers
```

```
select location from Follows, Profile
 where id = a and b = 'sina';

-- Output:

 location
----------
 Toronto
 Montreal
(2 rows)
```