

Subqueries: Solutions

Schema

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

Questions

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') Hofferings
WHERE Took.oid = Hofferings.oid;
```

Solution: It finds information about students who took an offering taught by Horton. On our dataset, this is the output:

sid	course	grade
99132	CSC343	79
98000	CSC343	82
98000	CSC263	78
99999	CSC343	89
157	CSC343	99

(5 rows)

2. What does this query do?

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

Solution: It finds information about students whose cgpa is higher than student 99999. On our dataset, this is the output:

sid	surname
99132	Marchmount

```
98000 | Fairgrieve
157 | Lakemeyer
(3 rows)
```

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');
```

Solution: It finds information about students got an 80 or higher in a course that some Lakemeyer took. They did not have to take the course together.

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);
```

Solution: It finds a values from R whose b occurs in S.

(b) Can we express this query without using subqueries?

Solution: You might think this query is equivalent:

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

(Or we could do a natural join.) But they are not the same in all cases. If a tuple from R connects successfully with more than one tuple from S, this new query will yield duplicates that the original did not.

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 );
```

Solution: It finds instructors who have exactly one offering. On our dataset, this is the output:

```
instructor
-----
```

Truta
Heap
Chechik
Davies
Johancsik
Reisman
Dow
Miller
Mendel
Richler
(10 rows)

6. What does this query do?

```
SELECT DISTINCT oid
FROM Took
WHERE EXISTS (
    SELECT *
    FROM Took t, Offering o
    WHERE
        t.oid = o.oid AND
        t.oid <> Took.oid AND
        o.dept = 'CSC' AND
        took.sid = t.sid )
ORDER BY oid;
```

Solution: It finds course offerings that include a student who has taken something else that is a CSC course. On our dataset, this is the output:

```
oid
-----
1
3
5
6
7
8
9
11
13
14
15
16
17
21
22
26
27
28
31
34
```

35
38
39
(23 rows)

7. Now let's write some queries! For each course, that is, each department and course number combination, find the instructor who has taught the most offerings of it. If there are ties, include them all. Report the course (eg "csc343"), instructor and the number of offerings of the course by that instructor.

- (a) First, create a view called Counts to hold, for each course, and each instructor who has taught it, their number of offerings.

Solution:

-- This intermediate result is helpful:

```
CREATE VIEW Counts as
SELECT dept || cnum as course, instructor, count(oid)
FROM Offering
GROUP BY cnum, dept, instructor;
```

-- Let's take a look at what this computes.

-- (Our dataset doesn't give this view a very good test.)
SELECT * from Counts;

course	instructor	count
CSC148	Miller	1
CSC148	Jepson	2
EEB263	Suzuki	1
CSC343	Mylopoulos	2
EEB216	Suzuki	1
ENG235	Richler	1
ENV200	Suzuki	1
EEB263	Johancsik	1
ENG235	Percy	1
HIS220	Dow	1
CSC343	Horton	1
CSC148	Chechik	1
EEB150	Mendel	1
CSC343	Truta	1
ENV320	Suzuki	1
ENG205	Reisman	1
HIS220	Young	1
ENG205	Atwood	1
CSC263	Horton	2
ENG110	Atwood	1
HIS296	Young	1
CSC207	Gries	2
ANT200	Zorich	1
ANT203	Davies	1
ENG110	Percy	1
ANT203	Zorich	1
CSC343	Heap	1
CSC320	Jepson	2

CSC207	Craig		2
CSC263	Craig		1

(30 rows)

- (b) Now solve the problem. Do not use any joins. (This will force you to use a subquery.)

Solution:

-- Now we can solve the problem using a subquery:

```
SELECT course, instructor, count
FROM Counts C1
WHERE count >= ALL (
    SELECT count
    FROM Counts C2
    WHERE C1.course = C2.course )
ORDER BY C1.course;
```

-- Here's another version:

```
SELECT course, instructor, count
FROM Counts C1
WHERE count = (
    SELECT max(count)
    FROM Counts C2
    WHERE C1.course = C2.course )
ORDER BY C1.course;
```

-- Here's what they both produce:

course	instructor	count
-----	+	-----
ANT200	Zorich	1
ANT203	Zorich	1
ANT203	Davies	1
CSC148	Jepson	2
CSC207	Craig	2
CSC207	Gries	2
CSC263	Horton	2
CSC320	Jepson	2
CSC343	Mylopoulos	2
EEB150	Mendel	1
EEB216	Suzuki	1
EEB263	Suzuki	1
EEB263	Johancsik	1
ENG110	Atwood	1
ENG110	Percy	1
ENG205	Atwood	1
ENG205	Reisman	1
ENG235	Richler	1
ENG235	Percy	1
ENV200	Suzuki	1
ENV320	Suzuki	1
HIS220	Dow	1
HIS220	Young	1
HIS296	Young	1

(24 rows)

8. Use EXISTS to find the surname and email address of students who have never taken a CSC course.

Solution:

```
SELECT surname, email
FROM Student
WHERE NOT EXISTS (
  SELECT *
  FROM Took JOIN Offering ON Took.oid = Offering.oid
  WHERE Took.sid = Student.sid AND Offering.dept = CSC);
```

```
  surname |   email
-----+-----
  Simpson | doh@gmail
(1 row)
```

9. Use EXISTS to find every instructor who has given a grade of 100.

Solution: This query is almost correct.

```
SELECT instructor
FROM Offering O1
WHERE EXISTS (
  SELECT *
  FROM Offering O2 JOIN Took ON O2.oid = Took.oid
  WHERE O1.instructor = O2.instructor AND grade = 100);
```

```
instructor
-----
Atwood
Atwood
(2 rows)
```

Can you see why an instructor is reported more than once? Could they be reported more than twice? How would you fix this?

10. Let's say that a course has level "junior" if its cNum is between 100 and 299 inclusive, and has level "senior" if its cNum is between 300 and 499 inclusive. Report the average grade, across all departments and course offerings, for all junior courses and for all senior courses. Report your answer in a table that looks like this:

```
  level | levelavg
-----|-----
  junior |
  senior |
```

Each average should be an average of the individual student grades, not an average of the course averages.

Solution:

```
CREATE VIEW Grades AS
SELECT cnum, dept, grade
FROM Offering natural join Took;
```

```
(SELECT 'junior' AS level, avg(grade) AS levelavg
FROM Grades
WHERE cnum >= 100 AND cnum <= 299)
union
(SELECT 'senior' AS level, avg(grade) AS levelavg
FROM Grades
WHERE cnum >= 300 AND cnum <= 499);
```

level	levelavg
junior	75.0952380952380952
senior	77.5000000000000000

(2 rows)