Subqueries: Solutions

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]

Questions

1. What does this query do? (The || operator concatenates two strings.)

   SELECT sid, dept || cnum as course, grade
   FROM Took,
   (SELECT *
    FROM Offering
    WHERE instructor = 'Horton') Hofferin
   WHERE Took.oid = Hofferin.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
3. What does this query do?

```sql
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?

```sql
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:

```sql
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?

```sql
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
-----------
```
6. What does this query do?

```sql
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:

<table>
<thead>
<tr>
<th>oid</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>11</td>
</tr>
<tr>
<td>13</td>
</tr>
<tr>
<td>14</td>
</tr>
<tr>
<td>15</td>
</tr>
<tr>
<td>16</td>
</tr>
<tr>
<td>17</td>
</tr>
<tr>
<td>21</td>
</tr>
<tr>
<td>22</td>
</tr>
<tr>
<td>26</td>
</tr>
<tr>
<td>27</td>
</tr>
<tr>
<td>28</td>
</tr>
<tr>
<td>31</td>
</tr>
<tr>
<td>34</td>
</tr>
</tbody>
</table>
7. Now let’s write some queries! For each course 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. Suggestion: Use one or more views to hold intermediate step(s).

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      
CSC207    | Craig      | 2      
CSC263    | Craig      | 1      

-- 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 | Johansik   | 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. 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:

<table>
<thead>
<tr>
<th>level</th>
<th>levelavg</th>
</tr>
</thead>
<tbody>
<tr>
<td>junior</td>
<td></td>
</tr>
<tr>
<td>senior</td>
<td></td>
</tr>
</tbody>
</table>

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);
```

<table>
<thead>
<tr>
<th>level</th>
<th>levelavg</th>
</tr>
</thead>
<tbody>
<tr>
<td>junior</td>
<td>75.0952380952380952</td>
</tr>
<tr>
<td>senior</td>
<td>77.5000000000000000</td>
</tr>
</tbody>
</table>

(2 rows)