

CSCC43 UTSC

Tutorial Week 6 – Joins and Subqueries

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] \subseteq Course [dept, cNum]

Took [sID] \subseteq Student [sID]

Took [oID] \subseteq Offering [oID]

Part 1: Joins

Questions

1. Which of these queries is legal?

- (a)

```
SELECT count(distinct dept), count(distinct instructor)
FROM Offering
WHERE term >= 20089;
```
- (b)

```
SELECT distinct dept, distinct instructor
FROM Offering
WHERE term >= 20089;
```
- (c)

```
SELECT distinct dept, instructor
FROM Offering
WHERE term >= 20089;
```

Solution:

a) Legal, and here is the result:

```
count | count
-----+-----
      6 |      16
(1 row)
```

b) ERROR: syntax error at or near "distinct"

LINE 1: SELECT distinct dept, distinct instructor

c)

```
dept | instructor
-----+-----
EEB  | Johancsik
ENG  | Percy
CSC  | Truta
HIS  | Young
CSC  | Horton
ENG  | Reisman
CSC  | Chechik
CSC  | Gries
ENG  | Atwood
ENV  | Suzuki
ANT  | Zorich
HIS  | Dow
CSC  | Craig
CSC  | Jepson
CSC  | Heap
ANT  | Davies
(16 rows)
```

2. Under what conditions could these two queries give different results? If that is not possible, explain why.

```
SELECT surName, campus
FROM Student;
```

```
SELECT distinct surName, campus
FROM Student;
```

Solution:

If there were two students on the same campus with the same surname, their surname and campus would be repeated in the result of the first query, but not in the result of the second.

3. For each student who has taken a course, report their sid and the number of different departments they have taken a course in.

Solution:

```
SELECT sid, count(distinct dept)
FROM Took JOIN Offering ON Took.oid = Offering.oid
GROUP BY sid;
```

sid	count
157	5
11111	3
98000	6
99132	4
99999	5

(5 rows)

The 'distinct' is necessary, otherwise every course the student has taken (unless it had a 'NULL' value for 'dept') would count, even if they were all in the same department!

```
SELECT sid, count(dept)
FROM Took JOIN Offering ON Took.oid = Offering.oid
GROUP BY sid;
```

sid	count
98000	15
99132	7
11111	5
99999	12
157	15

(5 rows)

4. Suppose we have two tables with content as follows:

```
SELECT *
FROM One;
```

a	b
1	2
6	12
	100
20	

(4 rows)

```
SELECT *
FROM Two;
```

b	c
2	3
100	101
20	21
2	4
2	5

(5 rows)

(a) What query could produce this result?

a	b	c
1	2	3
1	2	4
1	2	5
	20	21
	100	101

(5 rows)

```
SELECT * FROM One NATURAL RIGHT JOIN Two;
```

But note that PostgreSQL changes the column order on this query, actually producing:

b	a	c
2	1	3
2	1	4
2	1	5
20		21
100		101

(5 rows)

This would also provide the same rows, although in different column order:

```
SELECT * FROM Two NATURAL LEFT JOIN One;
```

(b) What query could produce this result?

a	b	c
1	2	3
1	2	4
1	2	5
6	12	
	100	101
20		

(6 rows)

```
SELECT * FROM One NATURAL LEFT JOIN Two;
```

But note that postgresSQL changes the column order on this query, actually producing:

b	a	c
2	1	3
2	1	4
2	1	5
12	6	
100		101
	20	

(6 rows)

This would also provide the same rows, although in different column order:

```
SELECT * FROM Two NATURAL RIGHT JOIN One;
```

Part 1: Subqueries

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
Arv
Miller
Mendel
Richler
(11 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
(23ows)
```

7. Now let us 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.

```
CREATE VIEW Counts AS
SELECT (dept || cNum) AS course, instructor, count(*) AS count
FROM Offering
GROUP BY (dept || cNum), instructor;
```

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

```
SELECT course, instructor
FROM Counts
WHERE (course, count)
IN (
    SELECT course, max(count)
    FROM Counts
    GROUP BY course
);
```

```
DROP VIEW Counts;
```

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

```
SELECT sid
FROM Student s
WHERE NOT EXISTS (
    SELECT *
    FROM Took t, Offering o
    WHERE t.oid = o.oid
    AND o.dept = 'CSC'
```

```

        AND s.sid = t.sid
    );

```

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

```

SELECT DISTINCT o.instructor
FROM Offering o
WHERE EXISTS (
    SELECT *
    FROM Took t
    WHERE t.oid = o.oid
    AND grade = 100
);

```

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.

```

CREATE VIEW LevelAverages AS
(
    SELECT 'junior' AS level, AVG(t.grade) AS grade
    FROM Took t, Offering o
    WHERE t.oid = o.oid
    AND o.cNum >= 100
    AND o.cNum < 300
    GROUP BY t.oid
)
UNION ALL
(
    SELECT 'senior' AS level, AVG(t.grade) AS grade
    FROM Took t, Offering o
    WHERE t.oid = o.oid
    AND o.cNum >= 300
    AND o.cNum < 500
    GROUP BY t.oid
);

SELECT level, AVG(grade)

```

```
FROM LevelAverages  
GROUP BY level;  
  
DROP VIEW LevelAverages;
```