Week 7 Querying Databases: The Relational Way (2)

Assignment Solutions

1. Consider the relations **Students(id, gpa)** and **Enrolls(id,number,term)**. You are asked to find the difference between the average GPA of the students who take CS 411 and that of the students who take CS 511. Which of the following SQL queries will give you the correct answer (as a single-value relation)? Assume that there are a number of students who have taken CS 411 or CS 511 more than once.

   ○ SELECT S411.avg_gpa - S511.avg_gpa FROM (SELECT AVG(gpa) AS avg_gpa FROM Students, Enrolls WHERE Students.id = Enrolls.id AND number = "CS411") AS S411, (SELECT AVG(gpa) AS avg_gpa FROM Students, Enrolls WHERE Students.id = Enrolls.id AND number = "CS511") AS S511

   ○ SELECT S411.avg_gpa - S511.avg_gpa FROM (SELECT AVG(gpa) AS avg_gpa FROM Students WHERE id IN (SELECT id FROM Enrolls WHERE number = "CS411")) AS S411, (SELECT AVG(gpa) AS avg_gpa FROM Students WHERE id IN (SELECT id FROM Enrolls WHERE number = "CS511")) AS S511

   ○ SELECT (SELECT AVG(gpa) FROM Students WHERE id IN (SELECT id FROM Enrolls WHERE number = "CS411")) - (SELECT AVG(gpa) FROM Students WHERE id IN (SELECT id FROM Enrolls WHERE number = "CS511")) FROM Students

Answer: B

Explanation: Check each solution and use the process of elimination.

At first all solutions look like they would work, however upon closer inspection we can notice that two of the three options are clearly wrong. In the first option, since the students can take CS411 or CS511 more than once, their GPAs will be counted more than once. The last option returns the same value multiple times, which is not what the question wants (a single value relation).

This leaves us with the second option that returns one value and also makes sure to count the grade of each student once.

2. Consider the relations **Students(id, major)** and **Enrolls(id,number,term)**. You are asked to find, for every CS major, the total number of distinct courses taken; i.e., your goal is to produce a table **R(id, count)** that includes an entry for every student majoring in CS. Assume that there are brand-new CS majors who haven't registered for any course yet. Does the following SQL query meet the requirement?

   SELECT Students.id, COUNT(DISTINCT Enrolls.number) FROM Students, Enrolls WHERE Students.id = Enrolls.id AND Students.major = "CS" GROUP BY Students.id

   ○ Yes

   ○ No

Answer: B

Explanation: The answer is clearly option b, because those brand-new CS majors haven't registered yet, they're absent in Enrolls, therefore the join operation will eliminate such CS majors, resulting in an incomplete table.

There are a few ways to solve this query so that it fits with the question,

Possible solution 1, using subquery:

```
SELECT ID, (
SELECT COUNT(DISTINCT Enrolls.number)
FROM Enrolls
WHERE Enrolls.id = ID )
FROM Students
WHERE major = "CS"
```

Possible solution 2, using Union:

```
(
SELECT Students.id, COUNT(DISTINCT Enrolls.number)
FROM Students, Enrolls
WHERE Students.id = Enrolls.id AND Students.major = "CS" GROUP BY Students.id
)
UNION
(
SELECT ID, 0
FROM Students
WHERE ID NOT IN (SELECT ID FROM Enrolls) AND Major = "CS"
)
```

3. To find the "tougher than average" courses in **Enrolls(id, number, grade)**, we need to construct an SQL query resulting in a table **R(number, AVG(grade))** such that each course number in **R** has an average grade (over all the students having taken the course) that is lower than the mean of the average grades over all the courses (each course is equally weighted regardless of how many students have taken it). To get started, we have the following construct:

   SELECT number, AVG(grade) FROM Enrolls GROUP BY number HAVING AVG(grade) < (___)

   How would you complete the HAVING clause to get the right result?

   ○ SELECT AVG(grade) FROM Enrolls

   ○ SELECT AVG(grade) FROM Enrolls GROUP BY number

   ○ SELECT AVG(avg_grade) FROM (SELECT AVG(grade) AS avg_grade FROM Enrolls GROUP BY number)

Answer: C

Explanation: Here the question is asking us to find the mean of the average grades over all the courses. The hint in this question is that it is asking for two averages. Therefore, we are most likely looking for a subquery solution.

When we look at the first option, it only gives us the mean of the grades over all courses, but not the average grade of each course. Here this option weighs courses with more students higher than those with less.

When we look at the second option, we see that it finds the average grades of each individual course but omits finding the mean of all courses.

The last option puts the other two options together and gives us the correct solution.

4. Using the relation **Enrolls(id, number, term)**, we want to find the IDs of the students who take CS 511 but not CS 411. Of course the easiest way to formulate the SQL query is to use the **EXCEPT** operator:

(SELECT id FROME Enrolls WHERE number = "CS511") EXCEPT (SELECT id FROM Enrolls WHERE number = "CS411")

Now say the database system you use was found to have a buggy implementation of EXCEPT and you need to find an alternative approach to get the answer correctly. Check all correct responses below.

- [ ] No alternative query can achieve the exact same result as using **EXCEPT**.
- [ ] With an additional relation **Students(id)**, we can get the same result efficiently without using **EXCEPT**:

  SELECT id FROM Students WHERE id IN (SELECT id FROM Enrolls WHERE number = "CS511") AND id NOT IN (SELECT id FROM Enrolls WHERE number = "CS411")
- [ ] By self-joining Enrolls we can get the right answer without using **EXCEPT**:

  SELECT DISTINCT id FROM Enrolls E1, Enrolls E2 WHERE E1.id = E2.id AND E1.number = "CS511" AND E2.number != "CS411"

Answer: B

Explanation: The last option returns the set of students who have taken CS 511, and another course that is not CS 411 (including CS 511 itself). It doesn't guarantee that the student does not take CS 411.

Whereas the second option uses the fact that A Difference B means the set of elements IN A AND NOT IN B, which is what the question is asking for. Using the additional Students(id) relation is advantageous because:

1) It is most likely much smaller than Enrolls, hence it's less work to scan through.

2) There is no need to delete duplicates, because id is the key.

Since the correct answer is the second option we do not need the first option.

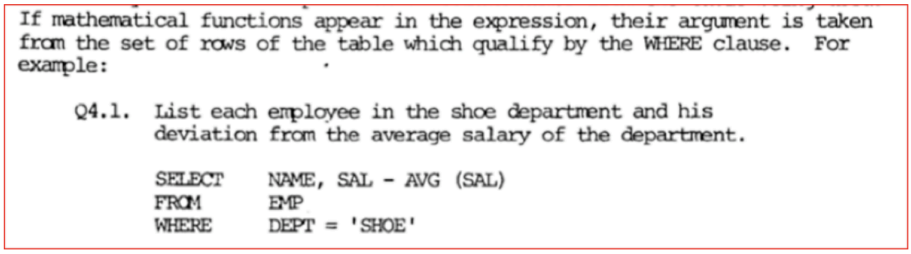
5. From the lectures, you have learned that in aggregate queries, the SELECT clause can only have group attributes, but not tuple attributes. Suppose the database implementation you use does not reject such illegal queries with errors. Is there any condition under which using a tuple attribute in the SELECT clause of an aggregate query does NOT result in unpredictable behavior in the query result?

- ○ No such condition exists, using such illegal queries always result in unpredictable behavior.
- ○ Yes, when the tuple attribute is functionally determined by the group attributes.
- ○ Yes, when the tuple attribute functionally determines the group attributes.

Answer: B

Explanation: From the previous week about *Designing Schemas* we learned that functional dependencies can determine a singular attribute based on another attribute. Therefore if we group by a tuple attribute that is functionally determined by group attributes then we will only get one value of the tuple attribute. This is predictable behavior. The last option, says that the FD follows the opposite direction; however, this does not guarantee that for every group there is only one value of the tuple attribute. Thus, the second option is correct.

6. **[Food for Thought]** The SQL query below is from the original SEQUEL paper. With what you have learned from the lectures, do you see anything wrong?



If mathematical functions appear in the expression, their argument is taken from the set of rows of the table which qualify by the WHERE clause. For example:

Q4.1. List each employee in the shoe department and his deviation from the average salary of the department.

```
SELECT    NAME, SAL - AVG (SAL)
FROM      EMP
WHERE     DEPT = 'SHOE'
```

- ○ It works fine, because AVG(SAL) produces a single scalar value.
- ○ Using NAME and SAL along with AVG(SAL) in SELECT is illegal.
- ○ Using DEPT in WHERE along with AVG(SAL) in SELECT is illegal.

Answer: B

Explanation: Since this is an aggregate query (using AVG), the SELECT clause can only use group attributes (only AVG(SAL)), not tuple attributes. To fix the query, we need to use a subquery to obtain AVG(SAL) as a single scalar value.