

CS 386/586 Fall 2007 Exercise, 5 October 9, 2007 Suggested Answers

Work through the exercises

Undergraduate students, please turn in your paper (with papers from all other exercises) at the end of the term.

Teacher table:			
Number	Name	Office	E-mail
1	Lois	O12	lmd@whatever.edu
2	James	O14	jterwill@whatever.edu
3	Sally	O55	sally@whatever.edu

Course table:		
Number	Name	Description
386	Intro to Databases	Introduction to relational database management systems
161	Intro to Computer Science 1	Introduction to problem-solving, algorithm and program design

In the following instance of the Class-offering table, only two of the class offerings for the Fall 2007 Quarter are listed. We assume that CRN is the key for this table. We also assume that each class offering has only one teacher, just to keep things simple.

Class-offering table:			
CRN	Course	Section	Teacher
701	386	1	1
702	161	1	3

Student table:	
Id	Name
101	Joe
102	Mary
103	Bill

Scores table			
CRN	Id	Assign	Grade
701	101	1	95
701	102	1	90
701	101	2	100
701	102	2	95
701	101	3	95

Show the query answer for each of the following queries. (Note: the UNION, INTERSECT, and EXCEPT clauses requires union-compatible input tables. If the input tables are not union-

compatible, then the query is not well-formed and does not have a query answer. The same is true for the \cap , \cup , and $-$ operators in relational algebra.)

Show the query answer for each of the following queries.

Explain what the query computes in English.

Write an equivalent query in relational algebra.

1. (SELECT * FROM Teacher)
UNION
(SELECT * FROM Student);

This query has no answer because this query is not well-formed. Teacher and Student are not union-compatible; Teacher has four attributes and Student only has two attributes so we can tell immediately that these two tables are not union-compatible.

2. (SELECT Name FROM Teacher)
UNION
(SELECT Name FROM Student);

English description. This query produces a list of distinct names consisting of names of faculty or students.

This query takes the union of two intermediate tables, one consisting of only the name attribute from Teacher and the other consisting of only the name attribute of Student. Note that some DBMSs automatically eliminate duplicates from the query answer when computing a UNION.

The query answer is:

Name
Lois
James
Sally
Joe
Mary
Bill

The equivalent relational algebra query is:

$(\pi_{\text{Name}} \text{Teacher}) \cup (\pi_{\text{Name}} \text{Student})$

3. (SELECT Grade FROM Scores WHERE Id = 101)
INTERSECT
(SELECT Grade FROM Scores WHERE Id = 102);

English description: This query produces a list of grades that both student Id 101 and student Id 102 have earned.

This query is very similar to the preceding one. Here the intermediate tables are grades for student with id 101 and grades for student with id 102. We are then taking the intersection of those two intermediate tables. I am assuming that the DBMS will eliminate duplicates before processing the INTERSECT. (I am not using INTERSECT ALL) The query answer is:

Grade
95

and the equivalent relational algebra query is:

$(\pi_{\text{Grade}}(\sigma_{\text{Id}=101}\text{Scores})) \cap (\pi_{\text{Grade}}(\sigma_{\text{Id}=102}\text{Scores}))$

4. (SELECT Assign FROM Scores WHERE Id = 101)
EXCEPT
(SELECT Assign FROM Scores WHERE Id = 102);

English description: This query lists assignment numbers that were turned in by student 101 but not turned in by student 102.

This query is similar to the preceding one with the same intermediate tables except that we retain the Assign attribute rather than the Grade attribute. We take the set difference of those two intermediate tables to find assignments that student with Id 101 has turned in that student with Id 102 has not turned in. I am assuming that the DBMS will eliminate duplicates before processing the INTERSECT. The query answer is:

Assign
3

and the equivalent relational algebra query is:

$(\pi_{\text{Assign}}(\sigma_{\text{Id}=101}\text{Scores})) - (\pi_{\text{Assign}}(\sigma_{\text{Id}=102}\text{Scores}))$

For each of the following queries, show the query answer and explain what it computes in English. Use the data shown on the first page of the exercise, above.

5. `SELECT Id, Name, Grade`
`FROM Student FULL OUTER JOIN Scores USING (Id);`

Remember that the `OUTER JOIN` includes every row that appears in an `INNER` (or regular) `JOIN`. In addition, a `FULL OUTER JOIN` will include `Student` rows for which there is no matching row from the `Scores` table (with `<null>` in the attributes from the `Scores` table). It will also include any rows from the `Scores` table for which there is no matching rows from the `Student` table. (It turns out that there aren't any such rows in the `Scores` table, for the sample data given.) If there were any such rows in the `Scores` table, they would have `<null>` in the attributes from the `Student` table.

English description: This query lists all of the grades each student has earned showing the student id, the student name, and the grade; the list includes all of the students even if the student doesn't have any grades. And it includes all grades even if the grade is not associated with a student.

The query answer is:

Id	Name	Grade
101	Joe	95
101	Joe	100
101	Joe	95
102	Mary	90
102	Mary	95
103	Bill	<null>

6. `SELECT Id, Name, Grade`
`FROM Student RIGHT OUTER JOIN Scores USING (Id);`

A `RIGHT OUTER JOIN` includes all rows from an `INNER JOIN` plus any rows from the table on the right (the `Grade` table, in this example) that do not join with the left table (the `Student` table in this example). Since we don't have any rows from the `Grade` table that don't match rows from the `Student` table, the query answer is the same as an `INNER` (i.e., regular) join.

English description: This query lists all of the grades earned by a student showing the student id, student name, and the grade. The list includes all grades, even if the grade is not associated with a student.


```
9. SELECT    DISTINCT Student.Id, Name, S1.Assign, S1.Grade
FROM        Student, Scores S1
WHERE       Student.Id = S1.Id and
            (SELECT MAX(Grade) FROM Scores S2
             WHERE S1.Id = S2.Id ) = S1.Grade;
```

This query includes a **WHERE** clause in the subquery.

English description: For all students that have at least one grade recorded in the database, this query lists the student id and name with the assignment number and grade for the assignment where this student achieved his or her highest score.

Id	Name	Assign	Grade
101	Joe	2	100
102	Mary	2	95

For questions 7, 8, and 9, which of the subqueries, if any, are correlated?

The subqueries in question 7 and 8 are **NOT** correlated because they do **NOT** mention any attributes from the outer query. That is, the subqueries in questions 7 and 8 compute the same answer every time the **WHERE** clause for the outer query is evaluated. This means that a **NON**-correlated subquery only needs to be evaluated once. The subquery in question 9 **IS** correlated because the subquery mentions **S1.id** from the outer query. The subquery in question 9 must be reevaluated every time the **WHERE** clause from the outer query is evaluated. That is, the subquery must be reevaluated for each different student that is considered in the outer query.

Consider the following table. It is the same as the table shown above except that we have added one new row, listed as the last row shown here.

Scores table			
<u>CRN</u>	<u>Id</u>	Assign	Grade
701	101	1	95
701	102	1	90
701	101	2	100
701	102	2	95
701	101	3	95
701	102	3	<i>null</i>

For each of the following queries, indicate whether this new, 6th row in the Scores table would satisfy the WHERE clause.

10. SELECT *
FROM Scores
WHERE Grade > 90;

The 6th row will NOT satisfy this WHERE clause because when Grade is *null*, the > predicate evaluates to unknown (rather than true).

11. SELECT *
FROM Scores
WHERE Grade = 95 OR Grade < 95 OR Grade > 95;

The 6th row will NOT satisfy this WHERE clause because when Grade is *null*, the > predicate, the < predicate, and the = predicate all evaluate to unknown (rather than true). Note: at first glance, this WHERE clause looks like it would evaluate to true for all possible Scores. That turns out not to be the case when the Grade is *null*.

12. SELECT *
FROM Scores
WHERE Grade IS NULL;

The 6th row will satisfy this WHERE clause because a Grade of *null* evaluates to true for the "IS NULL" predicate.

13. SELECT *
FROM Scores
WHERE Grade IS NOT NULL;

The 6th row will NOT satisfy this WHERE clause because a Grade of *null* evaluates to false for the "IS NOT NULL" predicate.