CS252.HACD — Solutions to Lecture Exercises (in Sections 4, 5, 6)

CS252.HACD.4: Relational Algebra, Principles and Part I

Slide 16: Degenerate Cases of JOIN

Consider *r1* JOIN *r2*. When *r1* and *r2* have disjoint headings (i.e., no attributes in common), then every tuple in *r1* matches every tuple in *r2* and we call the join a Cartesian product. Some authorities permit or require *r1* TIMES *r2* to be written in this special case, but that turns out to be not such a good idea. **Exercise: Why not? (Hint: look at the notes for Slide 12)**

> Consider relations R1 { A, B }, R2 { B, C }, and R3 { C, D }. We have seen that, thanks to the commutativity and associativity of JOIN, we can join these three together in any order. For example: (R1 JOIN R3) JOIN R2. But if we are required to use TIMES instead of JOIN for the join of R1 and R3, that particular expression is illegal.

CS252.HACD.5: Relational Algebra, Part II

Slide 9: Definition of Extension

Assume the existence of the following relvars:

CUST with attributes C# and DISCOUNT
ORDER with attributes O#, C#, and DATE
ORDER_ITEM with attributes O#, P#, and QTY
PRODUCT with attributes P# and UNIT_PRICE

The underlined attributes are those specified in a KEY declaration for each relvar. Thus, for example, there cannot be more than one order item for the same part in the same order.

The price of an order item can be calculated by the formula QTY*UNIT_PRICE*(1-(DISCOUNT/100)). Write down a relation expression to yield a relation with attributes O#, P#, and PRICE, giving the price of each order item.

```
WITH CUST JOIN ORDER JOIN ORDER_ITEM AS COI :
EXTEND COI ADD (QTY*UNIT_PRICE*(1-(DISCOUNT/100)) AS PRICE )
       { O#, P#, PRICE }
```

Slide 10: Two More Relvars

Write down a relational expression to give, for each pair of students sitting the same exam, the absolute value of the difference between their marks. Assume you can write ABS(x) to obtain the absolute value of *x*.

```
WITH EXAM_MARK RENAME ( StudentId AS S1, Mark as M1 ) AS EM1 ,
     EXAM_MARK RENAME ( StudentId AS S2, Mark as M2 ) AS EM2 ,
     EM1 JOIN EM2 AS EM1_2 ,
     EM1_2 WHERE S1 <> S2 AS Sat_same_exam ,
EXTEND Sat_same_exam ADD ( ABS ( M1 - M2 ) AS Diff )
{ S1, S2, Diff }
```

Slide 20: Definition of UNION

1. What is the result of *r* UNION *r*? **Answer:** *r*
2. Is UNION commutative? I.e., do *r1* UNION *r2* and *r2* UNION *r1* always denote the same relation?**Answer:** Yes, because OR is commutative.

3. Is UNION associative? I.e., do (*r1* UNION *r2*) UNION *r3* and *r1* UNION (*r2* UNION *r3*) always denote the same relation? **Answer:** Yes, because OR is associative.

Slide 23: Definition of NOT MATCHING

State the result of

1. *r* NOT MATCHING TABLE_DEE

   Answer: an empty relation with the same heading as *r*. Every tuple in *r* matches the single 0-tuple in TABLE_DEE.

2. *r* NOT MATCHING TABLE_DUM

   Answer: *r,* there being no tuples in TABLE_DUM for tuples in *r* to match.

3. *r* NOT MATCHING *r*

   Answer: an empty relation with the same heading as *r*. Every tuple in *r* matches itself.

4. (*r* NOT MATCHING *r*) NOT MATCHING *r*

   Answer: an empty relation with the same heading as *r,* because *r* NOT MATCHING *r* is empty.

5. *r* NOT MATCHING (*r* NOT MATCHING *r*)

   Answer: *r,* there being no tuples in *r* NOT MATCHING *r* for tuples in *r* to match.

Is NOT MATCHING associative? **Answer:** No, as shown by the answers to 4 and 5 above. Is it commutative? **Answer:** No, because the result is always a subset of the *first* operand.

Slide 24: MINUS

Define *r1* NOT MATCHING *r2* in terms of MINUS.

r1 NOT MATCHING *r2* ≡ *r1* MINUS ((*r1* JOIN *r2*) { *Hr1* }), where *Hr1* is a list consisting of the attribute names of *r1*. Alternatively *r1* MINUS (*r1* JOIN (*r2* { *C* })), where *C* is a list of the names of the attributes common to *r1* and *r2*.

CS252.HACD.6: Relational Algebra, Part III, and Other Operators

Slide 6: Definition of MATCHING

State the result of

1. (IS_CALLED MATCHING IS_ENROLLED_ON) MATCHING COURSE

StudentId	**Name**
S1	Anne
S2	Boris
S3	Cindy
S4	Devinder

   The IS_CALLED tuple for S5 disappears under the first invocation of NOT MATCHING, because S5 is not enrolled on any course. Now, because IS_CALLED and COURSE have no common attributes, every tuple in (IS_CALLED MATCHING IS_ENROLLED_ON) matches every tuple in COURSE. As there is at least one tuple in COURSE, the second invocation of NOT MATCHING returns its first operand.

2. IS_CALLED MATCHING (IS_ENROLLED_ON MATCHING COURSE)

The result is as in 1. The invocation (IS_ENROLLED_ON MATCHING COURSE) returns its first operand, because every enrolment is indeed for some existing course. The overall result therefore consists of each tuple of IS_CALLED that represents a student who is enrolled on some course.

3. (IS_CALLED MATCHING COURSE) MATCHING IS_ENROLLED_ON

   The result is as in 1. Because IS_CALLED and COURSE have no common attributes and COURSE is not empty, the first invocation of NOT MATCHING returns its first operand. The second then returns the relation consisting of those tuples representing students who are enrolled on some course, as before.

4. IS_CALLED MATCHING (COURSE MATCHING IS_ENROLLED_ON)

   The result is as in 1. The invocation (COURSE MATCHING IS_ENROLLED_ON) returns a relation consisting of each tuple of COURSE that represents a course on which somebody is enrolled. The overall result therefore consists of each tuple of IS_CALLED that represents a student who is enrolled on some course (that somebody is enrolled on!).

5. (IS_CALLED MATCHING IS_ENROLLED_ON) MATCHING COURSE

   The result is as in 1. The invocation (IS_CALLED MATCHING IS_ENROLLED_ON) returns a relation consisting of each tuple of IS_CALLED that represents a student who is enrolled on some course. As every such tuple matches some tuple in COURSE, that is also the overall result, as before.

6. IS_CALLED MATCHING (IS_ENROLLED_ON MATCHING COURSE)

   The result is as in 1. The invocation (IS_ENROLLED_ON MATCHING COURSE) returns its first operand, because every enrolment is indeed for some existing course. The overall result therefore consists of each tuple of IS_CALLED that matches some tuple of IS_ENROLLED_ON (on StudentId) and thus represents a student who is enrolled on some course, as before.

Is semijoin commutative? **Answer:** No—it returns a subset of its *first* operand. Is it associative? **Answer:** Yes. Every tuple in *r1* that matches (a tuple in *r2* that matches a tuple in *r3*) defines the same set of tuples as every (tuple in *r1* that matches a tuple in *r2*) that matches a tuple in *r3*.

Slide 8: Definition of COMPOSE

1. State the result of (IS_CALLED COMPOSE IS_ENROLLED_ON) COMPOSE COURSE

Name	**Title**
Anne	Database
Boris	Database
Devinder	Database
Anne	HCI
Cindy	Op Systems

The invocation (IS_CALLED_ON COMPOSE IS_ENROLLED_ON) results in

Name	CourseId
Anne	C1
Boris	C1
Devinder	C1
Anne	C2
Cindy	C3

and then the second invocation of COMPOSE joins that with COURSE and discards CourseId.

2. State the result of IS_CALLED COMPOSE (IS_ENROLLED_ON COMPOSE COURSE)

The result is as in 1.

The invocation (IS_ENROLLED_ON COMPOSE COURSE) yields

StudentId	Title
S1	Database
S2	Database
S4	Database
S1	HCI
S3	Op Systems

and then the second invocation of COMPOSE joins that with IS_CALLED and discards StudentId.

Is COMPOSE commutative? **Answer:** Yes, because JOIN is. Is it associative? **Answer:** No! It might appear to be from the above result, but consider the following simpler example:

RELATION {TUPLE { X 2} } COMPOSE
(RELATION { TUPLE{X 1} } COMPOSE RELATION { TUPLE { X 1}})

The second line yields TABLE_DEE, and when TABLE_DEE is composed with any relation *r* the result is *r* (here, RELATION {TUPLE { X 2} }). However, if we change the order like this:

(RELATION {TUPLE { X 2} } COMPOSE RELATION { TUPLE{X 1} })
COMPOSE RELATION { TUPLE { X 1}})

then the first line yields TABLE_DUM and TABLE_DUM composed with any relation *r* yields the empty relation of the same heading as *r*.

Slide 9, Read-only Counterparts of Update Operators:

Using the value for relvar EXAM_MARK as shown in Slide 3:

1. State the result of
UPDATE EXAM_MARK WHERE CourseId = 'C1' (Mark := Mark + 1)

StudentId	CourseId	Mark
S1	C1	86
S2	C1	50
S4	C1	94

Only the "updated" tuples appear in the result.

2. Give the value of EXAM_MARK that results from
UPDATE EXAM_MARK WHERE CourseId = 'C1' (Mark := Mark + 1) ;

StudentId	CourseId	Mark
S1	C1	86
S1	C2	49
S2	C1	50
S3	C3	66
S4	C1	94

All the tuples appear in the result, not just the updated ones.

Slide 11: From A to B and Back Again

Assuming that relation *r* has no attribute named G, state the result of

1. *r* GROUP { } AS G

   r, extended with a single attribute named G, whose value is RELATION { } { } (i.e., TABLE_DEE) in each tuple.

2. *r* GROUP { ALL BUT } AS G

   A unary relation *s* whose sole attribute, G, is of the same type as *r*. If *r* is empty, then *s* is empty; otherwise *s* is equal to RELATION { TUPLE { G *r* } }.

What exactly is *r* GROUP { *a* } AS G shorthand for, where *a* is a list of attribute names of *r*?

Let *A1, ... , An* be the attribute names in *a*. Then *r* GROUP { *a* } AS G is equivalent to:

EXTEND *r* ADD (*r* JOIN RELATION { TUPLE { *A1 A1, ... , An An* } } AS G
{ ALL BUT *A1, ... , An* }

Slide 14

State the result of

1. `TABLE_DEE ⊇ TABLE_DEE` (remember to replace ⊇ by >= if you try these in Rel)

   TRUE (every set is a superset of itself)

2. `TABLE_DEE ⊇ TABLE_DUM`

   TRUE (every set is a superset of the empty set)

3. `TABLE_DUM ⊇ TABLE_DEE`

   FALSE (the empty set is a superset of no nonempty set)

4. `TABLE_DUM ⊇ TABLE_DUM`

   TRUE (every set is a superset of itself)

5.
```
(((EXAM_MARK{ALL BUT Mark} GROUP({CourseId} AS G1))
   RENAME (StudentId AS Sid1)
 JOIN
  (EXAM_MARK{ALL BUT Mark} GROUP({CourseId} AS G2))
   RENAME (StudentId AS Sid2))
WHERE G1 ⊇ G2 AND Sid1<>Sid2){ALL BUT G1, G2}
```

Sid1	Sid2
S1	S2
S1	S4
S2	S4
S4	S2

What does the query in Exercise 5 here really mean?

Find pairs of student ids *Sid1* and *Sid2* such that student *Sid2* sat every exam that student *Sid1* sat.

Reasoning:

```
(EXAM_MARK{ALL BUT Mark} GROUP({CourseId} AS G1))
   RENAME (StudentId AS Sid1)
```

gives pairs <*Sid1, G1*> such that *G1* is the set of courses for which student *Sid1* sat the exam. Similarly,

```
(EXAM_MARK{ALL BUT Mark} GROUP({CourseId} AS G2))
   RENAME (StudentId AS Sid2)
```

gives pairs <*Sid2, G2*> such that *G2* is the set of courses for which student *Sid2* sat the exam. There are no common attributes for the JOIN of these two, so the result is the Cartesian product, consisting of quadruples <*Sid1, G1, Sid2, G2*> such that *G1* is the set of courses for which student *Sid1* sat the exam AND *G2* is the set of courses for which student *Sid2* sat the exam. From that we take just those tuples where the set of *Sid1*'s exams sat (*G1*) is a superset of *Sid2*'s set of exams sat (*G2*), and *Sid1* and *Sid2* are not in fact the same student. The final projection merely discards the atributes that are no longer needed.

Slide 19, Adding 2 and 3:

Slide 19 shows the following expression for adding 2 and 3 "the relational way":

c FROM TUPLE FROM (PLUS COMPOSE RELATION { TUPLE { a 2, b 3 } })

Write a similar expression, using the relation PLUS again, to subtract 4 from 5 and give the answer as a simple number.

Answer:

a FROM TUPLE FROM (PLUS COMPOSE RELATION { TUPLE { c 5, b 4 } })

or:

b FROM TUPLE FROM (PLUS COMPOSE RELATION { TUPLE { a 4, c 5 } })

I wrote the TUPLE expression that way around in the second solution just to remind you that the order of attributes is insignificant.

End of solutions