CS252:HACD Fundamentals of Relational Databases
Notes for Section 6: Relational Algebra Part III and Other Operators

1. Cover slide

In Parts I and II we gave relational algebra counterparts for AND, OR, NOT and existential quantification. As those four operators are sufficient for expressing anything in predicate calculus, we don't really need any more. However, it is normal, once we have agreed on a set of primitive operators, to define some more in terms of those primitives, to provide convenient *shorthands*. We will look at the ones that were chosen for inclusion in **Tutorial D**.

That completes our collection of relational operators that, when invoked, yield relations. But there are other operators of interest in **Tutorial D** and we close this section by having a quick look at some of those.

2. The Running Example

No notes.

3. … and these

No Notes.

4. Some Useful Shorthands

The title really refers to the relational operators. The "other operators" are cannot be expressed in terms of existing operators, so must be regarded as primitive.

5. Semijoin

The strange name is inspired by the observation that the internal process to produce its result involves approximately half of what has to be done to perform a join. For each tuple in the first operand, look for matching tuples in the second operand. If any are found, then the first operand tuple belongs in the result; otherwise it doesn't. For a join, the system needs to find *all* the matching tuples and then join each one in turn to the first operand tuple.

6. Definition of MATCHING

Exercises: State the result of

1. (IS_CALLED MATCHING IS_ENROLLED_ON) MATCHING COURSE
2. IS_CALLED MATCHING (IS_ENROLLED_ON MATCHING COURSE)
3. (IS_CALLED MATCHING COURSE) MATCHING IS_ENROLLED_ON
4. IS_CALLED MATCHING (COURSE MATCHING IS_ENROLLED_ON)
5. (IS_CALLED MATCHING IS_ENROLLED_ON) MATCHING COURSE
6. IS_CALLED MATCHING (IS_ENROLLED_ON MATCHING COURSE)

Is semijoin commutative? Is it associative?

7. Composition

No notes.

8. Definition of COMPOSE

Exercises:

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

Is COMPOSE commutative? Is it associative?

9. Read-only Counterparts of Update Operators

Suppose, for example, that you wanted to see the effect of a certain invocation of UPDATE, without actually updating the database. Imagine that you want to make sure the folloiwng imperative will have the effect you hope it will have:

UPDATE IS_CALLED WHERE StudentId = 'S1' (Name := 'Ann') ;

This example was given in Lecture HACD.1, Introduction. It has the effect of replacing the single tuple with StudentId 'S1' by TUPLE { StudentId 'S1', Name 'Ann' } in the current value of the relvar IS_CALLED, retaining all the other tuples alone.

The example shown on the slide yields the relation consisting of the tuple(s) that would be affected if the above UPDATE *statement* were actually executed:

StudentId	Name
S1	Ann

The read-only version of UPDATE is also useful when you what to replace an existing attribute by the result of some calculation. For example,

UPDATE R (X := X + 1)

is shorthand for

(EXTEND R ADD (X + 1 AS new_X) { ALL BUT X }) RENAME (new_X AS X)

(assuming, of course, that R doesn't have an attribute named new_X).

We don't need read-only counterparts of INSERT and DELETE because the required effects can be obtained using operators we have already seen, UNION and restriction, as shown on the slide.

Exercises:

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

Note the addition of the semicolon in Exercise 2 here.

10. GROUP/UNGROUP

Relation A is a copy of IS_CALLED.

Notice how B can be interpreted to represent exactly the same information as A.

11. From A to B and Back Again

Exercises: State the result of

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

In each case assume that *r* does not have an attribute named G.

What exactly is *r* GROUP { *a* } AS G shorthand for, where *a* is a list of attribute names of *r*? Hint: look at Slide 12, **To obtain C_ER from COURSE and EXAM_MARK:,** in Relational Algebra Part II.

12. Other Operators

The aggregate operators we have already seen are the ones shown in Lecture HACD.5, Slides 123 and 14. Relation "selection" is the RELATION operator described in Lecture HACD.2, Slides 10-12 (and Slide 12 shows an example of tuple selection).

13. Relation Comparison Case Study

B and C can both be derived from our existing relvars as follows:

B = IS_CALLED GROUP { StudentId } AS N_StudentIds

C = IS_ENROLLED_ON GROUP { StudentId } AS C_StudentIds

14. Relation Comparison (1)

Important note concerning Rel: **Tutorial D**'s symbols ⊇ and ⊆ are not available on normal keyboards. For that reason, *Rel* uses the compound symbols >= for ⊇ and <= for ⊆.

"⊇" is pronounced "is a superset of". The other relational comparison operators are "⊆" (is a subset of) and "=" (is the same relation as).

Relation *r1* is a superset of *r2* if and only if every tuple of *r2* is a tuple of *r1*. Relation *r1* is a subset of *r2* if and only if *r2* is a superset of *r1*. If *r1*=*r2*, then each is a superset (and subset) of the other.

You can add "⊃" (is a proper superset of) and "⊂" (is a proper subset of) if you wish, but these would probably not be very useful in practice.

Note very carefully the second line of that predicate. Without it, the predicate is "Every student enrolled on CourseId is called Name." Consider course C4, on which nobody is enrolled. It is the case, then, that every student enrolled on C4 is called Anne, is called Boris, Cindy, Devinder, Eve, … and for that matter every name you can think of!

It gets worse. Everybody enrolled on the nonexistent course C5 is also called Anne, Boris, Cindy, Devinder, Eve and so on!

Logic plays a few funny tricks like that. In case it strikes you as strange that something can be true of everybody when it is also true of nobody, consider that there does not exist a student who is enrolled on C4 and not called Anne. There being no exception to the rule, "Everybody enrolled on C4 is called Anne", that rule must be considered to hold true.

Exercises: State the result of

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

3. `TABLE_DUM ⊇ TABLE_DEE`

4. `TABLE_DUM ⊇ TABLE_DUM`

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}
```

What does the query in Exercise 5 here really mean?

15. Relation Comparison (2)

No notes.

16. Tuple extraction

TUPLE FROM is perhaps not obviously useful at first sight. It is most commonly used in conjunction with attribute value extraction …

17. Attribute Value Extraction

This is **Tutorial D**'s counterpart of SQL's so-called scalar subquery. Consider the folloiwng SQL expression:

```
SELECT CourseId
FROM IS_ENROLLED_ON
WHERE StudentId = ( SELECT StudentId FROM IS_CALLED WHERE Name = 'Anne' )
```

The expression on the right-hand side of = here is a scalar subquery, whose result is a value of type CHARACTER and thus comparable with the StudentId column of IS_ENROLLED_ON. Standing on its own, SELECT StudentId FROM IS_CALLED WHERE Name = 'Anne' is a query that results in a table, not a scalar value, but when such an expression appears, parenthesised, in such a place, it is *coerced* to being of the type of the single column in the SELECT clause. The subquery must result in a table with no more than one row, otherwise you get a run-time exception. (If the result is empty, you get NULL.)

Coercion, or implicit type conversion, is found in some languages but is deprecated by some authorities as unsound language design. Indeed, the coercion we see in SQL's scalar subqueries has led to certain difficulties in the more recent development of the language.

In **Tutorial D** we make the type conversion explicit. TUPLE FROM explicitly yields a single tuple (and gives a run-time exception if the operand has more than one tuple *or is empty*). And then Name FROM (in the example on the slide) explicitly denotes the value of the specified attribute (Name).

18. A Relational View of Arithmetic

Recall the relation PLUS, from Lecture HACD.2: Values, Types, Variables, Operators. It shows how a read-only operator can be represented as a relation, in this case mapping every pair of numbers <*a*,*b*> to a number, *c*, such that in each case we have an example (in fact, an instantiation) of *a*+*b*=*c*.

19. Adding 2 and 3

To obtain the result of 2+3, we must first substitute 2 for *a* and 3 for *b*. Here this is done by using the COMPOSE operator, on PLUS and the relation literal shown on the slide. The relation literal denotes a relation of cardinality one, whose single tuple provides those substitution values. Here is the result of PLUS COMPOSE RELATION { TUPLE { a 2, b 3 } }:

c
5

TUPLE FROM that yields the value TUPLE { c 5 }, and c FROM *that* yields the number 5.

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

20. Tuple Counterparts of Relational Operators

These operators perhaps need no further explanation, but note that in each case we can think of its relation counterpart as repeated application of the corresponding tuple operator on the tuples of the operand relation(s):

- Relation RENAME applies the same tuple RENAME to each tuple in the operand relation.
- Relation projection applies the same tuple projection to each tuple in the operand relation (but remember that each resulting tuple must appear just once in the resulting relation, so the internal process involves elimination of redundant duplicates).
- Relation extension applies the same tuple extension to each tuple in the operand relation.
- Relation UPDATE applies the same tuple UPDATE to each tuple in the operand relation.
- Relation JOIN applies the same tuple JOIN to each pair of tuples *t1* and *t2* such that *t1* is a tuple in the first operand and *t2* is a tuple in the second, and *t1* and *t2* can indeed be joined to yield a tuple. They can be joined if and only if their set-theory union is a tuple. For example:

  The union of { a 1, b 2 } and { b 2, c 3 } yields the set { a 1, b 2, c3 }, which is indeed a tuple.

  The union of { a 1, b 2 } and { b 3, c 4 } yields the set { a 1, b 2, b3, c 4 }, which is not a tuple because it has two distinct elements with attribute name b.

- Relation COMPOSE applies the same tuple COMPOSE to each pair of tuples *t1* and *t2* such that *t1* is a tuple in the first operand and *t2* is a tuple in the second, and *t1* and *t2* can be joined to yield a tuple. Tuple COMPOSE is tuple JOIN followed by tuple projection, just as relation COMPOSE is relation JOIN followed by relation projection.

EXERCISES

Here, again, are the exercises given in the numbered section of these Notes.

Slide 6, Definition of MATCHING:

State the result of

1. (IS_CALLED MATCHING IS_ENROLLED_ON) MATCHING COURSE
2. IS_CALLED MATCHING (IS_ENROLLED_ON MATCHING COURSE)
3. (IS_CALLED MATCHING COURSE) MATCHING IS_ENROLLED_ON
4. IS_CALLED MATCHING (COURSE MATCHING IS_ENROLLED_ON)
5. (IS_CALLED MATCHING IS_ENROLLED_ON) MATCHING COURSE
6. IS_CALLED MATCHING (IS_ENROLLED_ON MATCHING COURSE)

Is semijoin commutative? Is it associative?

Slide 8, Definition of COMPOSE:

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

Is COMPOSE commutative? Is it associative?

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

Note the addition of the semicolon in Exercise 2 here.

Slide 11: From A to B and Back Again

State the result of

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

In each case assume that *r* does not have an attribute named G.

What exactly is *r* GROUP { *a* } AS G shorthand for, where *a* is a list of attribute names of *r*? Hint: look at Slide 12, **To obtain C_ER from COURSE and EXAM_MARK:,** in Relational Algebra Part II.

Slide 14, Relation Comparison (1):

State the result of

1. `TABLE_DEE ⊇ TABLE_DEE`
2. `TABLE_DEE ⊇ TABLE_DUM`
3. `TABLE_DUM ⊇ TABLE_DEE`

4. `TABLE_DUM ⊇ TABLE_DUM`

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}
```

What does the query in Exercise 5 here really mean?

End of Notes