On Making Relational Division Comprehensible

Lester I. McCann

mccann@uwp.edu

Computer Science Department

University of Wisconsin — Parkside

Kenosha, WI

Frontiers in Education

November 7, 2003

Outline

- Background
- The Relational Division Operator
  - Purpose
  - Connection with Cartesian Product
  - An Example of Its Use
- Division in Relational Algebra
- Division in SQL
  - From Relational Algebra Expression
  - Using a Logical Tautology
  - Using Set Containment
  - Comparing Set Cardinalities
- Division Pitfalls
- Conclusion

Background

- Relational database management systems are based on Codd's relational data model
  - Rooted in set theory
- Codd's original data languages:
  - Relational Calculus (non–procedural)
    - Based on First–Order Predicate Calculus
  - Relational Algebra (procedural)
    - Five fundamental operators: σ, π, $\times$, $-$, $\cup$
    - Three additional operators: $\cap$, $\bowtie$, $\div$

Division

- Division is considered the most challenging of the eight operators
  - Defined using three operators (π, $-$, and $\times$) and six operations
  - Based on finding values that are not answers
  - Not easily expressed in SQL
  - A challenge to explain to students
- Often an afterthought in database texts

- But necessary to answer a specific type of query!

What Division Does

- Division identifies the attribute values from a relation that are found to be paired with all of the values from another relation.
- Viewed another way:
  - As multiplication is to division in arithmetic, Cartesian Product ($\times$) is to Division in relational algebra.

Cartesian Product and Division

- Consider the unary relations $\boxed{m}$ and $\boxed{n}$, and their Cartesian Product $\boxed{o}$:

$\boxed{m}$

C
4
8

$\boxed{n}$

D
3
1
7

$\boxed{o}$

C	D
4	3
4	1
4	7
8	3
8	1
8	7

Cartesian Product and Division

- Division is the opposite of Cartesian Product:

o

C	D
4	3
4	1
4	7
8	3
8	1
8	7

$o \div n =$ **m**

C
4
8

$o \div m =$ **n**

D
3
1
7

Cartesian Product and Division

- Division is the opposite of Cartesian Product:

$\boxed{o}$	C	D
	4	3
	4	1
	4	7
	8	3
	8	1
	8	7

$o \div n =$

$\boxed{m}$	C
	4
	8

$o \div m =$

$\boxed{n}$	D
	3
	1
	7

- That's easy! Who needs a formal definition? : -)

A More Practical Example

- Consider this subset of Date's famous Suppliers–Parts–Projects schema:

p

pno	pname	color	weight	city
P1	Nut	Red	12.0	London
...	...	...	...	...
P6	Cog	Red	19.0	London

spj

sno	pno	jno	qty
S1	P1	J1	200
...	...	...	...
S5	P6	J4	500

A More Practical Example (cont.)

Query: *Find the sno values of the suppliers that supply all parts of weight equal to 17.*

p	<u>pno</u>	pname	color	weight	city

spj	<u>sno</u>	<u>pno</u>	<u>jno</u>	qty

A More Practical Example (cont.)

Query: *Find the sno values of the suppliers that supply all parts of weight equal to 17.*

p	pno	pname	color	weight	city

spj	sno	pno	jno	qty

- Students can tell us that we need to create this schema:

α	sno	pno

β	pno

A More Practical Example (cont.)

- Constructing $\boxed{\alpha}$ and $\boxed{\beta}$ is straight–forward:

$$\alpha \leftarrow \pi_{sno,pno}(SPJ) \text{ and } \beta \leftarrow \pi_{pno}(\sigma_{weight=17}(P))$$

$\boxed{\alpha}$

sno	pno
S1	P1
S2	P3
S2	P5
S3	P3
S3	P4
S4	P6
S5	P1
S5	P2
S5	P3
S5	P4
S5	P5
S5	P6

$\boxed{\beta}$

pno
P2
P3

Division in Relational Algebra

Idea: Find the values that *do not* belong in the answer, and remove them from the list of possible answers.

- In our P–SPJ example, the list of possible answers is just the available *sno* values in α:

$\pi_{sno}(\alpha)$

sno
S1
S2
S3
S4
S5

Division in Relational Algebra (cont.)

- All possible *sno–pno* pairings can be generated easily:

$\pi_{sno}(\alpha)$

sno
S1
S2
S3
S4
S5

$\times$

β

pno
P2
P3

$=$

γ

sno	pno
S1	P2
S1	P3
S2	P2
S2	P3
S3	P2
S3	P3
S4	P2
S4	P3
S5	P2
S5	P3

Division in Relational Algebra (cont.)

- If we remove from $\boxed{\gamma}$ all of the pairings also found in $\boxed{\alpha}$, the result will the values of *sno* that we **do not** want.
- See next slide!

Division in Relational Algebra (cont.)

$\boxed{\gamma}$

sno	pno
S1	P2
S1	P3
S2	P2
S2	P3
S3	P2
S3	P3
S4	P2
S4	P3
S5	P2
S5	P3

$-$

$\boxed{\alpha}$

sno	pno
S1	P1
S2	P3
S2	P5
S3	P3
S3	P4
S4	P6
S5	P1
S5	P2
S5	P3
S5	P4
S5	P5
S5	P6

$=$

$\boxed{\delta}$

sno	pno
S1	P2
S1	P3
S2	P2
S3	P2
S4	P2
S4	P3

Victim tuples are shown in magenta.

Note that S5 is not represented in $\boxed{\delta}$.

Division in Relational Algebra (cont.)

- All that remains is to remove the ‘non–answer’ *sno* values from the set of possible answers:

$\pi_{sno}(\alpha)$

sno
S1
S2
S3
S4
S5

$-$

$\pi_{sno}(\delta)$

sno
S1
S2
S3
S4

$=$

$\div$

sno
S5

Relational Algebra Summary

- The complete division expression:

$$\alpha \div \beta = \pi_{A-B}(\alpha) - \pi_{A-B}((\pi_{A-B}(\alpha) \times \beta) - \alpha)$$

3 1 2

- Ignoring the projections, there are just three steps:
  1. Compute all possible attribute pairings
  2. Remove the existing pairings
  3. Remove the non–answers from the possible answers
- This is well within the grasp of DB students!

Moving On to SQL

- Most DB texts cover division when they cover Relational Algebra
  - But they often ignore/hide it in their SQL coverage!
  - Leaves students believing division isn't important — not good!
- Why do they overlook division in SQL?
  - No built–in division operator
  - Standard SQL expressions of division are complex
- Division in SQL need not be confusing

Expressing Division in SQL

- I know of four ways to do division in SQL...

1. Direct conversion of the Relational Algebra expression
2. By applying a quantification tautology
3. By using set containment
4. By comparing set cardinalities

- ... but books frequently choose to use 2 — the hard one!

#1: From Relational Algebra

- Recall the Relational Algebra formulation:

$$\alpha \div \beta = \pi_{A-B}(\alpha) - \pi_{A-B}((\pi_{A-B}(\alpha) \times \beta) - \alpha)$$

- We need to know that in SQL ...
  - ... `EXCEPT` means difference (–)
  - ... a join without the `WHERE` clause produces a Cartesian Product
  - ... nested `SELECT`s sometimes need an alias

    ...`(SELECT ...)  as alias`...

#1: From Relational Algebra (cont.)

- The direct translation from Relational Algebra:

$$\alpha \div \beta = \pi_{A-B}(\alpha) - \pi_{A-B}((\pi_{A-B}(\alpha) \times \beta) - \alpha)$$

```
select distinct sno from spj
except
select sno
  from ( select sno, pno
           from (select sno from spj) as t1,
                (select pno from p where weight=17) as t2
         except
         select sno, pno from spj
       ) as t3;
```

where α would be `select sno, pno from spj`

and β is `select pno from p where weight=17`

#2: By Logical Tautology

- Consider our original English P–SPJ query:

  Find the sno values of the suppliers that supply all parts of weight equal to 17.

- Now consider this rewording that makes the quantifications more explicit:

  Find the sno values such that <u>for</u> <u>all</u> parts of weight 17 <u>there</u> <u>exist</u> suppliers that supply them all

- Problem: For this we need $\forall a(\exists b\, f(a,b))$, but SQL does not support universal quantification.

#2: By Logical Tautology (cont.)

- Solution: We can apply this tautology:

$$\forall a(\exists b\, f(a,b)) \leftrightarrow \overline{\exists} a(\overline{\exists} b\, f(a,b))$$

- Wording before conversion:

Find the sno values such that for all parts of weight 17 there exist suppliers that supply them all

- Wording after conversion:

Find sno values such that there do not exist any parts of weight 17 for which there do not exist any suppliers that supply them all

#2: By Logical Tautology (cont.)

- The resulting SQL version (with intentional misspellings of ‘local’ and ’global’):

```
select distinct sno from spj as globl
 where not exists
      ( select pno from p
          where weight = 17 and not exists
               ( select * from spj as locl
                   where locl.pno = p.pno
                     and locl.sno = globl.sno));
```

- Imagine presenting this to undergrads who have just a lecture or two of SQL under their belts.
- You **do** get the chance to talk about scoping of aliases...

#3: Set Containment

- Consider this: If a supplier supplies a superset of the parts of weight 17, the supplier supplies them all.
  - If only SQL had a superset (containment) operator...

#3: Set Containment

- Consider this: If a supplier supplies a superset of the parts of weight 17, the supplier supplies them all.
  - If only SQL had a superset (containment) operator...

- Logic to the rescue!

  If $A \supseteq B$, $B - A$ will be empty (or, $\overline{\exists}(B - A)$)

  where
  - A contains the parts of weight 17 that a supplier supplies
  - B contains all available parts of weight 17.

#3: Set Containment (cont.)

- The resulting SQL query scans through the sno values, computes A based on the current sno, and includes it in the quotient if the difference is empty:

```
select distinct sno from spj as globl
 where not exists (
    ( select pno from p where weight = 17 )
    except
    ( select p.pno
        from p, spj
       where p.pno = spj.pno
         and spj.sno = globl.sno )
 );
```

- The lack of a double negation makes this approach easier to understand.

#4: A Counting We Will Go

- The effect of the set containment approach is to indirectly count the members of each of the two sets, in hopes that the sums are equal.
- Thanks to SQL's `count()`, we can do the counting directly.
- The plan:
  - We find the suppliers that supply parts of weight 17 and how many of those parts each supplies.
  - A `having` clause compares each count to the total number of parts of weight 17.

#4: A Counting We Will Go (cont.)

- The resulting SQL query:

```
  select distinct sno
    from spj, p
   where spj.pno = p.pno and weight = 17
group by sno
  having count(distinct p.pno) =
           (select count (distinct pno)
            from p
            where weight = 17);
```

- No negations at all!
- Not surprisingly, students like it quite well.

Two Division Pitfalls

1. As “All” / “For All” queries need division, does that mean division $\equiv \forall$? No!
   - Consider this query:

     What are the names of the students taking all of the Computer Science seminar classes?

   - We need operand relations like these:

enroll	name	class

seminar	class

   - But . . . what if *seminar* is empty?

Two Division Pitfalls (cont.)

1. (cont.)
   - One can say that, if no seminar classes are offered, then all students are taking all seminars!
   - Of course, the real meaning of the query was:

     *What are the names of the students taking all of the Computer Science seminar classes, **assuming that at least one is being offered**?*

   - Students need to realize that the divisor . . .
     - . . . is usually the result of a subquery, and
     - . . . may well contain no tuples

Two Division Pitfalls (cont.)

2. Queries that give the same result as division are not replacements for division
   - Consider this variation of our 'all parts of weight 17' query:

     Find the sno values of the suppliers that supply all parts of weight equal to 19.
   - If students inspect Date's sample data, they learn the answer is suppliers S4 and S5 …
   - …which also is the result of this query:

     Find the sno values of the suppliers that supply parts of weight equal to 19.

Two Division Pitfalls (cont.)

2. (cont.)
   - That query can be answered with a simple join of the division operands:

     ```
     select distinct sno
     from (select sno, pno from spj) as one,
          (select pno from p where weight = 19) as two
     where one.pno = two.pno;
     ```

   - To help students avoid temptation, select a divisor relation that contains more than one tuple.
     - Only one part has weight 19, but two parts have weight 17.
     - Attempting the join on the ‘weight 17’ query would produce S2, S3, and S5 — all three supply at least one of the parts of weight 17.

Conclusion

- Division is as important in SQL as it is in Relational Algebra
- Students can understand division in both languages if we give them a chance
- A variety of possible implementations of division are possible in SQL
- Looking for shortcuts to division doesn't work

Any Questions?

This full–screen PDF presentation was created in LaTeX using the `prosper` presentation class.