Database Management System

Relational Algebra and operations

Relational Algebra

- Basic operations:
 - $\frac{Selection}{Projection} \begin{pmatrix} \sigma \end{pmatrix}$ Selects a subset of rows from relation. Deletes unwanted columns from relation.

 - <u>Cross-product</u> (X) Allows us to combine two relations. <u>Set-difference</u> () Tuples in reln. 1, but not in reln. 2.

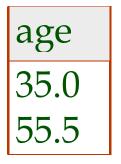
 - <u>Union</u> (1) Tuples in reln. 1 and in reln. 2.
- Additional operations:
 - Intersection, *join*, division, renaming: Not essential, but (very!) useful.
- Since each operation returns a relation, operations can be composed! (Algebra is "closed".)

Projection

- Deletes attributes that are not in *projection list*.
- Schema of result contains exactly the fields in the projection list, with the same names that they had in the (only) input relation.
- Projection operator has to eliminate *duplicates*! (Why??)
 - Note: real systems typically don't do duplicate elimination unless the user explicitly asks for it. (Why not?)

sname	rating
yuppy	9
lubber	8
guppy	5
rusty	10

 $\pi_{sname,rating}(S2)$



 $\pi_{age}(S2)$

Selection

- Selects rows that satisfy selection condition.
- No duplicates in result! (Why?)
- Schema of result identical to schema of (only) input relation.
- *Result* relation can be the *input* for another relational algebra operation! (*Operator composition.*)

sid	sname	rating	age
28	yuppy	9	35.0
58	rusty	10	35.0

 $\sigma_{rating > 8}^{(S2)}$

sname	rating
yuppy	9
rusty	10

 $\pi_{sname,rating}(\sigma_{rating>8}^{(S2)})$

Union, Intersection, Set-Difference

- All of these operations take two input relations, which must be <u>union-compatible</u>:
 - Same number of fields.
 - `Corresponding' fields have the same type.
- What is the *schema* of result?

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0
44	guppy	5	35.0
28	yuppy	9	35.0

$S1 \cup S2$

sid	sname	rating	age	sid	sname	rating	age
22	dustin	7	45.0	31	lubber	8	55.5
				58	rusty	10	35.0
	<i>S</i> 1–	-S2			S	$S1 \cap S2$	

Cross-Product

- Each row of S1 is paired with each row of R1.
- Result schema has one field per field of S1 and R1, with field names `inherited' if possible.
 - *Conflict*: Both S1 and R1 have a field called *sid*.

(sid)	sname	rating	age	(sid)	bid	day
22	dustin	7	45.0	22	101	10/10/96
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.5	22	101	10/10/96
31	lubber	8	55.5	58	103	11/12/96
58	rusty	10	35.0	22	101	10/10/96
58	rusty	10	35.0	58	103	11/12/96

• <u>Renaming operator</u>: $\rho(C(1 \rightarrow sid1, 5 \rightarrow sid2), S1 \times R1)$

Joins

$$R \bowtie_{c} S = \sigma_{c} (R \times S)$$

Condition Join:

(sid)	sname	rating	age	(sid)	bid	day
22	dustin		45.0	58	103	11/12/96
31	lubber	8	55.5	58	103	11/12/96
$S1 \bowtie_{S1.sid < R1.sid} R1$						

- *Result schema* same as that of cross-product.
- Fewer tuples than cross-product, might be able to compute more efficiently
- Sometimes called a *theta-join*.

Joins

<u>Equi-Join</u>: A special case of condition join where the condition c contains only equalities.

sid	sname	rating	age	bid	day
22	dustin	7	45.0	101	10/10/96
58	rusty	10	35.0	103	11/12/96

 $S1 \bowtie_{sid} R1$

- Result schema similar to cross-product, but only one copy of fields for which equality is specified.
- Natural Join: Equijoin on all common fields.

Division

Not supported as a primitive operator, but useful for expressing queries like:

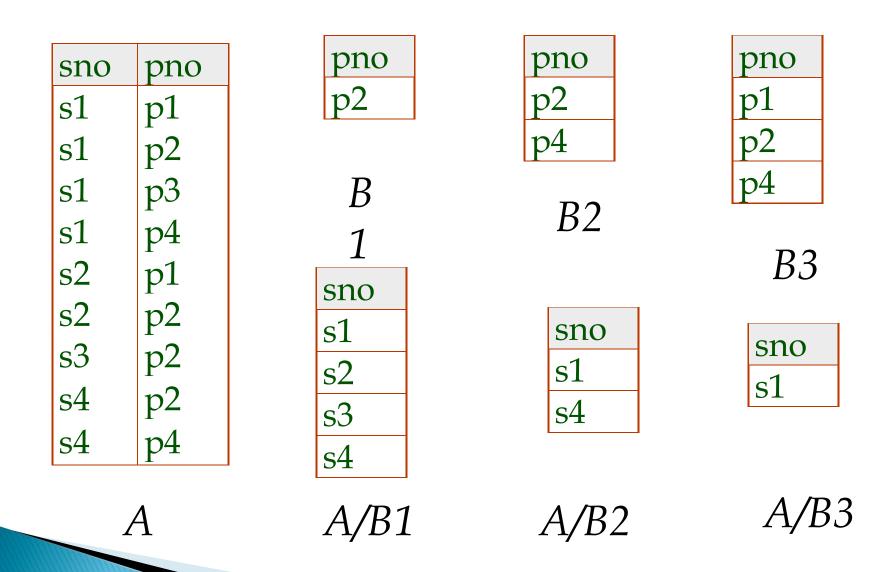
Find sailors who have reserved <u>all</u> boats.

Let *A* have 2 fields, *x* and *y*, *B* have only field *y*.

•
$$A/B = \{\langle x \rangle \mid \exists \langle x, y \rangle \in A \ \forall \langle y \rangle \in B\}$$

- i.e., *A/B* contains all *x* tuples (sailors) such that for *every y* tuple (boat) in *B*, there is an *xy* tuple in *A*.
- Or. If the set of y values (boats) associated with an x value (sailor) in A contains all y values in B, the x value is in A/B.
- In general, x and y can be any lists of fields; y is the list of fields in B, and x y is the list of fields of A.

Examples of Division A/B



Expressing A/B Using Basic Operators

- Division is not essential op; just a useful shorthand.
 - (Also true of joins, but joins are so common that systems implement joins specially.)
- Idea: For A/B, compute all x values that are not `disqualified' by some y value in B.
 - *x* value is *disqualified* if by attaching *y* value from *B*, we obtain an *xy* tuple that is not in *A*.

Disqualified *x* values:
$$\pi_{\chi}((\pi_{\chi}(A) \times B) - A)$$

A/B: $\pi_{\chi}(A)$ – all disqualified tuples

Find names of sailors who've reserved boat #103

▶ Solution 1: $\pi_{sname}((\sigma_{bid=103} \text{Reserves}) \bowtie \text{ Sailors})$

* Solution 2: ρ (*Templ*, $\sigma_{bid=103}$ Reserves)

 ρ (*Temp2*, *Temp1* \bowtie *Sailors*)

 π_{sname} (Temp2)

* Solution 3: $\pi_{sname}(\sigma_{bid=103}(\text{Reserves} \bowtie Sailors))$

Find names of sailors who've reserved a red boat

 Information about boat color only available in Boats; so need an extra join:

$$\pi_{sname}((\sigma_{color='red'}Boats) \bowtie \text{Reserves} \bowtie Sailors)$$

✤ A more efficient solution:

 $\pi_{sname}(\pi_{sid}((\pi_{bid}\sigma_{color='red'}Boats) \bowtie \operatorname{Res}) \bowtie \operatorname{Sailors})$

A query optimizer can find this, given the first solution!

Find sailors who've reserved a red or a green boat

Can identify all red or green boats, then find sailors who've reserved one of these boats:

 $\rho \ (Tempboats, (\sigma_{color} =' red' \lor color =' green', Boats))$ $\pi_{sname} (Tempboats \bowtie \text{Reserves} \bowtie Sailors)$

Can also define Tempboats using union! (How?)

* What happens if \vee is replaced by \wedge in this query?

Find sailors who've reserved a red <u>and</u> a green boat

Previous approach won't work! Must identify sailors who've reserved red boats, sailors who've reserved green boats, then find the intersection (note that *sid* is a key for Sailors):

$$\rho$$
 (Tempred, $\pi_{sid}((\sigma_{color='red'}Boats) \bowtie \text{Reserves}))$

 ρ (Tempgreen, $\pi_{sid}((\sigma_{color = green}, Boats)) \bowtie \text{Reserves}))$

 $\pi_{sname}((Tempred \cap Tempgreen) \bowtie Sailors)$