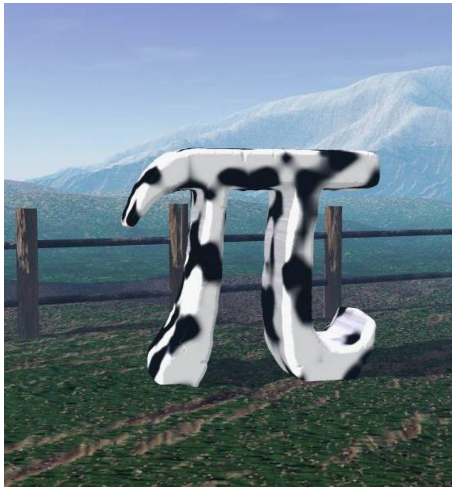
Relational Algebra

Chapter 4.1-4.2

Problem Set #1
was issued today.
It is due on 9/16.



Formal Query Languages

- What is the basis of Query Languages?
- Two formal Query Languages form the basis of "real" query languages (e.g. SQL):
  - *Relational Algebra*: Operational, it provides a ***recipe*** for evaluating the query. Useful for representing execution plans.
  - *Relational Calculus*: Lets users describe what they want, rather than how to compute it. (Non-operational, *declarative*.)

What is an "Algebra"

- Set of operands and operations that they are "closed" under all compositions
- Examples
  - Boolean algebra - operands are the logical values True and False, and operations include AND(), OR(), NOT(), etc.
  - Integer algebra - operands are the set of integers, operands include ADD(), SUB(), MUL(), NEG(), etc. many of which have special in-fix operator symbols (+,-,*,-)
- In our case *operands* are *relations*, what are the *operators*?

Example Instances

- "Sailors" and "Reserves" relations for our examples.
- We'll use "named field notation", which assumes that names of fields in query results are "inherited" from names of fields in query input relations.

R1

sid	bid	day
22	101	10/10/96
58	103	11/12/96

S1

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

S2

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

Relational Algebra

- Basic operations:
  - *Selection* (σ) Selects a subset of rows from relation.
  - *Projection* (π) Deletes unwanted columns from relation.
  - *Cross-product* ($\times$) Allows us to combine two relations.
  - *Set-difference* ($-$) Tuples in reln. 1, but not in reln. 2.
  - *Union* ($\cup$) 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?)

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

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

age
35.0
55.5

$$\pi_{age}(S2)$$

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.*)

<u>sid</u>	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	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 *union-compatible*:
  - 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
22	dustin	7	45.0

$S1 - S2$

sid	sname	rating	age
31	lubber	8	55.5
58	rusty	10	35.0

$S1 \cap S2$

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

- *Renaming operator*: $\rho(T(S1.sid \rightarrow sid1, R1.sid \rightarrow sid2), S1 \times R1)$

Joins

- Condition Join: $R \bowtie_{c} S = \sigma_{c}(R \times S)$

(sid)	sname	rating	age	(sid)	bid	day
22	dustin	7	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*.

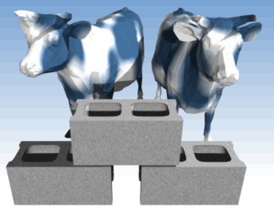
Joins

- *Equi-Join*: 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 (no labels on bowtie).



Division

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

  Find sailors who have reserved ***all*** *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*.**
  - 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 $x \cup y$ is the list of fields of *A*.

Examples of Division A/B

sno	pno
s1	p1
s1	p2
s1	p3
s1	p4
s2	p1
s2	p2
s3	p2
s4	p2
s4	p4

A

pno
p2

B1

pno
p2
p4

B2

pno
p1
p2
p4

B3

sno
s1
s2
s3
s4

A/B1

sno
s1
s4

A/B2

sno
s1

A/B3

Expressing A/B Using Basic Operators

- Division is not essential; it's 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: $\pi_x((\pi_x(A) \times B) - A)$

A/B: $\pi_x(A) - \pi_x((\pi_x(A) \times B) - A)$

Relational Algebra Examples

- Assume the following extended schema:
  - ***Sailors(sid: integer, sname: string, rating: integer, age: real)***
  - ***Reserves(sid: integer, bid: integer, day: date)***
  - ***Boat(bid: integer, bname: string, bcolor: string)***
- Objective: Write a relational algebra expression whose result instance satisfies the specified conditions
  - May not be unique
  - Some alternatives might be more efficient (in terms of time and/or space)

Example

Sailors:

sid	sname	rating	age
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35.0
64	Horatio	7	35.0
71	Zorba	10	16.0
74	Horatio	9	35.0
85	Art	3	25.5
95	Bob	3	63.5

Reservations:

sid	bid	day
22	101	10/10/98
22	102	10/10/98
22	103	10/8/98
22	104	10/7/98
31	102	11/10/98
31	103	11/6/98
31	104	11/12/98
64	101	9/5/98
64	102	9/8/98
74	103	9/8/98

Boats:

bid	bname	color
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

Names of sailors who've reserved boat #103

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

- Solution 2: $\rho\,(Temp1,\ \sigma_{bid=103} \text{Reserves})$

  $\rho\,(Temp2,\ Temp1 \bowtie Sailors)$

  $\pi_{sname}(Temp2)$

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

Names of sailors who've reserved a red boat

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

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

- A more efficient solution:

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

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

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:

$$\rho\ (Tempboats, (\sigma_{color='red' \vee color='green'}\ Boats))$$

$$\pi_{sname}(Tempboats \bowtie Reserves \bowtie Sailors)$$

- Can also define Tempboats using union! (How?)
- What happens if $\vee$ is replaced by $\wedge$ in this query?

Sailors who've reserved a red and 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):

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

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

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

Names of sailors who've reserved all boats

- Use division; schemas of the input relations to / must be carefully chosen:

$$\rho\ (Tempsids, (\pi_{sid,bid} Reserves) \ / \ (\pi_{bid} Boats))$$

$$\pi_{sname} (Tempsids \bowtie Sailors)$$

- To find sailors who've reserved all 'Interlake' boats:

$$\rho(iBoats, \sigma_{bname='Interlake'} Boats)$$

$$\rho(Tempsids, (\pi_{sid,bid} Reserves)/(\pi_{bid} iBoats))$$

$$\pi_{sname} (Tempsids \bowtie Sailors)$$

Summary

- Relational algebra is an operational specification for queries
- Each operation applies to relations and results in a new relation
- Equivalent queries can be achieved via many alternative relational algebra expressions
- Relational algebra provides a more than minimal set of operators to provide compact specifications