Relational Algebra

Module 4

Professor Alex Brodsky

Database Systems

Relational Query Languages

- *Query languages:* Allow manipulation and retrieval of data from a database.
- Relational model supports simple, powerful QLs:
  - Strong formal foundation based on logic.
  - Allows for much optimization.
- Query Languages **!=** programming languages!
  - QLs not expected to be "Turing complete".
  - QLs not intended to be used for complex calculations.
  - QLs support easy, efficient access to large data sets.

Formal Relational Query Languages

Two mathematical Query Languages form the basis for "real" languages (e.g. SQL), and for implementation:

❶ *Relational Algebra*: More operational, very useful for representing execution plans.

❷ *Relational Calculus*: Lets users describe what they want, rather than how to compute it. (Non-operational, *declarative*.)

☛ *Understanding Algebra is key to understanding SQL,*
☛ *and query processing!*

Algebra Preliminaries

- A query is applied to *relation instances*, and the result of a query is also a relation instance.
  - *Schemas* of input relations for a query are fixed (but query will run regardless of instance!)
  - The schema for the *result* of a given query is also fixed! Determined by definition of query language constructs.

Example Instances

- "Sailors" and "Reserves" relations for our examples.

S1

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

R1

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

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

Algebra Operations

- Look what we want to get from the following table:

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

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

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

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.

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

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

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

Composition of Operations

- *Result* relation can be the *input* for another relational algebra operation! (*Operator composition*.)

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

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

sname	rating
yuppy	9
rusty	10

What do we want to get from two 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

What about: Who reserved boat 101?

Or: Find the name of the sailor who reserved boat 101.

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.

sid1	sname	rating	age	sid2	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 (because naming conflict)*:

$$\rho(sid \rightarrow sid1,\ S1) \times \rho(sid \rightarrow sid2,\ R1)$$

Why does this cross product help

Query: Find the name of the sailor who reserved boat 101.

$$CP = \rho(sid \rightarrow sid1,\ S1) \times \rho(sid \rightarrow sid2,\ R1)$$

$$\text{Result} = \pi_{Sname}(\sigma_{sid1 = sid2 \text{ and } bid = 101}(CP))$$

* Note my use of "temporary" relation CP.

Another example

- Find the name of the sailor having the highest rating.

$$\text{AllR} = \pi_{ratingA}(rating \rightarrow ratingA,\ S2)$$

$$\text{Result?} = \pi_{Sname}(\sigma_{rating < ratingA}(S2 \times \text{AllR}))$$

What's in "Result?" ?

Does it answer our query?

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

$S1 \cap S2$

Back to our query

- Find the name of the sailor having the highest rating.

$$\text{AllR} = \pi_{ratingA}(rating -> ratingA,\ S2)$$

$$\text{Tmp} = \pi_{Sid,Sname}(\sigma_{rating<ratingA}(S2 \times \text{AllR}))$$

$$\text{Result} = \pi_{Sname}(\pi_{Sid,Sname}(S2) - \text{Tmp})$$

* Why not project on Sid **only** for Tmp?

Relational Algebra (Summary)

- Basic operations:
  - *Selection* (σ) Selects a subset of rows from relation.
  - *Projection* (π) Deletes unwanted columns from relation.
  - *Cross-product* (×) Allows us to combine two relations.
  - *Set-difference* (-) Tuples in reln. 1, but not in reln. 2.
  - *Union* (∪) Tuples in reln. 1 and in reln. 2
  - *Rename* (ρ) Changes names of the attributes
- Since each operation returns a relation, operations can be *composed*! (Algebra is "closed".)
- Use of temporary relations recommended.

Natural Join

- *Natural-Join*:

$S1 \bowtie R1$ =

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

- *Result schema* similar to cross-product, but only one copy of each field which appears in both relations.
- *Natural Join* = Cross Product + Selection on common fields + drop duplicate fields.

Query revisited using natural join

Query: Find the name of the sailor who reserved boat 101.

$$\text{Result} = \pi_{Sname}(\sigma_{bid=101}(S1 \bowtie R1))$$

Or

$$\text{Result} = \pi_{Sname}(S1 \bowtie \sigma_{bid=101}(R1))$$

Consider yet another query

- Find the sailor(s) who reserved all the red boats.

R1

sid	bid	day
22	101	10/10/96
22	103	10/11/96
58	102	11/12/96

B

bid	color
101	Red
102	Green
103	Red

Start an attempt

- who reserved what boat:

$$S1 = \pi_{sid,bid}(R1) =$$

sid	bid
22	101
22	103
58	102

- All the red boats:

$$S2 = \pi_{bid}(\sigma_{color=red}(B)) =$$

bid
101
103

Division

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

  Find sailors who have reserved all red boats.

- Let *S1* have 2 fields, *x* and *y*; *S2* have only field *y*:
  - *S1/S2* = $\{\langle x \rangle \mid \textit{for all } \langle y \rangle \textit{ in } R2 \text{ (there exists } \langle x,y \rangle \textit{ in } R1)\}$
  - i.e., ***S1/S2* contains all *x* tuples (sailors) such that for *every* *y* tuple (redboat) in *S2*, there is an *xy* tuple in *S1* (i.e, x reserved y).**

- In general, *x* and *y* can be any lists of fields; *y* is the list of fields in *S2*, and $x \cup y$ is the list of fields of *S1*.

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

Find names of sailors who've reserved boat #103

- Solution 1:

$$\pi_{sname}((\sigma_{bid=103}(\text{Re}serves)) \bowtie Sailors)$$

- Solution 2:

$$Temp1 = \sigma_{bid=103}(\text{Re}serves)$$

$$Temp2 = Temp1 \bowtie Sailors$$

$$\text{Re}sult = \pi_{sname}(Temp2)$$

- Solution 3:

$$\pi_{sname}(\sigma_{bid=103}(\text{Re}serves \rtimes Sailors))$$

Find names of sailors who've reserved a red boat

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

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

- 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!*

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:

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

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

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

Find 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):

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

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

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

Find the names of sailors who've reserved all boats

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

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

$$\text{Result} = \pi_{sname}(Tempsids \bowtie Sailors)$$

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

..... $/\pi_{bid}(\sigma_{bname='Interlake'}(Boats))$

Summary

- The relational model has rigorously defined query languages that are simple and powerful.
- Relational algebra is more operational; useful as internal representation for query evaluation plans.
- Several ways of expressing a given query; a query optimizer should choose the most efficient version.