CS122A: Introduction to Data Management

Lecture #7
Relational Algebra I

Instructor: Chen Li

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:
  - *Relational Algebra*: More operational, very useful for representing execution plans.
  - *Relational Calculus*: Let's users describe what they want, rather than how to compute it. (Non-operational, *declarative*.)

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.
- Positional vs. named-field notation:
  - Positional notation easier for formal definitions, named-field notation more readable.
  - Both used in SQL (but try to avoid positional stuff!)

Example Instances

R1

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

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

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. (I.e., don't add expressive power, but…)
- Since each operation returns a relation, operations can be ***composed!*** (Algebra is "closed".)

Projection

- Removes 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.
- Relational projection operator has to eliminate *duplicates*! (Why??)
  - Note: real systems typically don't do duplicate elimination unless the user explicitly asks for it. (***Q:*** 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

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

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

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

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 are of 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

$$S1 \cap S2$$

Q: Any issues w/duplicates?

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

Result relation name

Attribute renaming list

Source expression (anything!)

- *Renaming operator*: $\rho\,(C(1 \rightarrow sid1, 5 \rightarrow sid2),\ S1 \times R1)$

Renaming

- *Conflict*: S1 and R1 both had *sid* fields, giving:

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

- Several renaming options available:

$$\rho\ (S1R1(1 \rightarrow sid1),\ S1 \times R1)$$

Positional renaming

$$\rho\ (TempS1(sid \rightarrow sid1),\ S1)$$

Name-based renaming

$$TempS1 \times R1$$

Generalized projection

(I like this notation best ☺)

$$(\pi_{sid \rightarrow sid1, sname, rating, age}(S1)) \times R1$$

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

$$S1 \bowtie_{sid<sid} R1$$

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

More 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*: An equijoin on *all* commonly named fields.

Division

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

  *Find sailors who have reserved **all** boats*.

- Let A have 2 fields, x and y, while B has one field y, so we have relations A(x,y) and B(y):
  - ***A/B* contains the *x* tuples (e.g., sailors) such that for *every y* tuple (e.g., 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 $x \cup y$ 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

sno
s1
s2
s3
s4

A/B1

pno
p2
p4

B2

sno
s1
s4

A/B2

pno
p1
p2
p4

B3

sno
s1

A/B3

Expressing A/B Using Basic Operators (Advanced Topic ☺)

- Division not an essential op; just a useful shorthand.

  (Also true of joins, but joins are so common and important that relational database 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 a *y* value from *B*, we obtain an *xy* tuple that does not appear in *A*.

Disqualified *x* values (*D*): $\pi_x((\pi_x(A) \times B) - A)$

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

Ex: *Wisconsin Sailing Club Database*

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	4	25.5
95	Bob	3	63.5

Reserves

sid	bid	date
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	9/8/93

Boats

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

Find names of sailors who've reserved boat #103

Sailors(sid, sname, rating, age) Reserves(sid, bid, day)
Boats(bid, bname, color)

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

Ex: *Wisconsin Sailing Club Database*

$$\sigma_{bid=103} \text{Reserves}$$

sid	bid	date
22	103	10/8/98
31	103	11/6/98
74	103	9/8/93

$$(\sigma_{bid=103} \text{Reserves}) \bowtie \textit{Sailors}$$

sid	bid	date	sname	rating	age
22	103	10/8/98	Dustin	7	45.0
31	103	11/6/98	Lubber	8	55.5
74	103	9/8/93	Horatio	9	35.0

$$\pi_{sname}((\sigma_{bid=103} \text{Reserves}) \bowtie \textit{Sailors})$$

sname
Dustin
Lubber
Horatio

Find names of sailors who've reserved a red boat

Sailors(sid, sname, rating, age)

Reserves(sid, bid, day)

Boats(bid, bname, color)

- Information about boat color only available in Boats; so need to do a join:

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

- A more "efficient" solution:

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

A query optimizer can find the latter given the first solution!

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

Sailors(sid, sname, rating, age)   Reserves(sid, bid, day)
Boats(bid, bname, color)

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

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

$$\pi_{sname}(Tempboats \bowtie \mathrm{Re}serves \bowtie Sailors)$$

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

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

Sailors(sid, sname, rating, age) Reserves(sid, bid, day)
Boats(bid, bname, color)

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

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

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

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

Find the names of sailors who've reserved all boats

Sailors(sid, sname, rating, age)   Reserves(sid, bid, day)
Boats(bid, bname, color)

- Uses division; schemas of the input relations feeding / operator must be *carefully chosen*:

$$\rho\ (Tempsids, (\pi_{sid,bid} \text{Re}serves) \,/\, (\pi_{bid} Boats))$$

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

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

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

Relational Algebra Summary

- The relational model has (several) rigorously defined query languages that are both simple and powerful in nature.
- Relational algebra is more operational; very useful as an internal representation for query evaluation plans.
- Several ways of expressing a given query; a query optimizer should choose the most efficient version. (Take CS122C? ☺)
- We'll add a few more operators later on…
- Next up for now: *Relational Calculus*