Relational Algebra

Chapter 4, Part A

Why Is This Important?

- Once we have the data in a database, we want to access it.
- Relational algebra supports expressive queries by composing fairly simple operators.
- Only few operators needed
- We need to know the operators for the schema refinement discussion.

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 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: Lets 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

Example Instances

- "Sailors" and "Reserves" relations for our examples.
- We'll use positional or named field notation, assume 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 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 input relation.
- Projection operator has to eliminate duplicates. (Why?)
  - Note: real systems typically do not eliminate duplicates 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 the selection condition.
- No duplicates in result (Why?)
- Schema of result is identical to schema of input relation.
- Operator composition example.

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 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* (*C is the output*): $\rho(C(1\to sid1, 5\to sid2), S1\times R1)$

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 it 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.

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 = $\{\langle x \rangle \mid \forall \langle y \rangle \in B : \exists \langle x, y \rangle \in A\}$
    - 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 attributes
  - y is the list of fields in B, and x∪y is the list of fields of A.

Examples of Division A/B

A:

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

B1:

pno
p2

B2:

pno
p2
p4

B3:

pno
p1
p2
p4

A/B1:

sno
s1
s2
s3
s4

A/B2:

sno
s1
s4

A/B3:

sno
s1

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

A/B: $\pi_x(A)$ − all disqualified tuples

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

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:

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

- 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:

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

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

- 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 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):

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

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

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

Find the names of sailors who've reserved all boats

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

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

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

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

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

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.