

Relational Algebra

R & G, Chapter 4

By relieving the brain of all unnecessary work, a good notation sets it free to concentrate on more advanced problems, and, in effect, *increases the mental power of the race.*

-- Alfred North Whitehead (1861 - 1947)



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

➡ Understanding Algebra & Calculus is key to understanding SQL, query processing!

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 over any legal instance)
 - The **schema for the result** of a given query is also **fixed**. It is determined by the definitions of the query language constructs.
- **Positional vs. named-field notation:**
 - Positional notation easier for formal definitions, named-field notation more readable.
 - Both used in SQL
 - Though positional notation is not encouraged

Relational Algebra: 5 Basic Operations

- **Selection** (σ) Selects a subset of **rows** from relation (horizontal).
- **Projection** (π) Retains only wanted **columns** from relation (vertical).
- **Cross-product** (\bowtie) Allows us to combine two relations.
- **Set-difference** ($-$) Tuples in r1, but not in r2.
- **Union** (\cup) Tuples in r1 and/or in r2.

Since each operation returns a relation, **operations can be composed!** (Algebra is "closed".)

Example Instances

<i>R1</i>	sid	bid	day
	22	101	10/10/96
	58	103	11/12/96

<i>S1</i>	sid	sname	rating	age
	22	dustin	7	45.0
	31	lubber	8	55.5
	58	rusty	10	35.0

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

Boats

<i>S2</i>	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

- Examples: $\pi_{age}(S2)$; $\pi_{sname,rating}(S2)$
- Retains only attributes that are in the "projection list".
- Schema of result:
 - exactly the fields in the projection list, with the same names that they had in the input relation.
- Projection operator has to **eliminate duplicates** (How do they arise? Why remove them?)
 - Note: real systems typically don't do duplicate elimination unless the user explicitly asks for it. (Why not?)

Projection

sname	rating
yuppy	9
lubber	8
guppy	5
rusty	10

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

age
35.0
55.5

 $\pi_{age}(S2)$

Selection (σ)

- Selects rows that satisfy **selection condition**.
- Result is a relation.
- Schema of result is same as that of the input relation.
- Do we need to do duplicate elimination?

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

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

sname	rating
yuppy	9
rusty	10

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

Union and 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.
- For which, if any, is duplicate elimination required?

Union

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

 $S1$

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

 $S1 \cup S2$

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

 $S2$

Set Difference

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

 $S1$

sid	sname	rating	age
22	dustin	7	45.0

 $S1 \cap S2$

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

 $S2$

sid	sname	rating	age
28	yuppy	9	35.0
44	guppy	5	35.0

 $S2 - S1$



Cross-Product

- $S1 \times R1$: Each row of S1 paired with each row of R1.
- Q: How many rows in the result?
- **Result schema** has one field per field of S1 and R1, with field names 'inherited' if possible.
 - May have a naming conflict: Both S1 and R1 have a field with the same name.
 - In this case, can use the *renaming operator*:
 $\rho(C1 \leftarrow sid1, S1 \leftarrow sid2), S1 \times R1$



Cross Product Example

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

R1

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

S1

$R1 \times S1 =$

(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



Compound Operator: Intersection

- In addition to the 5 basic operators, there are several additional "Compound Operators"
 - These add no computational power to the language, but are useful shorthands.
 - Can be expressed solely with the basic ops.
- Intersection takes two input relations, which must be **union-compatible**.
- Q: How to express it using basic operators?
 $R \cap S = R \times (R \times S)$



Intersection

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

S1

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

$S1 \cap S2$

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

S2



Compound Operator: Join

- Joins are compound operators involving cross product, selection, and (sometimes) projection.
- Most common type of join is a "**natural join**" (often just called "join"). $R \bowtie S$ conceptually is:
 - Compute $R \times S$
 - Select rows where attributes that appear in both relations have equal values
 - Project all unique attributes and one copy of each of the common ones.
- Note: Usually done much more efficiently than this.
- Useful for putting "normalized" relations back together.



Natural Join Example

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

R1

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

S1

$R1 \bowtie S1 =$

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



Other Types of Joins

- **Condition Join (or "theta-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.
- May have fewer tuples than cross-product.
- **Equi-Join:** Special case: condition c contains only conjunction of equalities.



Compound Operator: Division

- Useful for expressing "for all" queries like:
*Find sids of sailors who have reserved **all** boats.*
- For A/B attributes of B are subset of attrs of A .
– May need to "project" to make this happen.
- E.g., let A have 2 fields, x and y ; B have only field y :

$$A/B = \{x \mid \forall y \exists B(x, y) \subseteq A\}$$

A/B contains all tuples (x) such that for every y tuple in B , there is an xy tuple in 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 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: $\sigma_x ((\sigma_x(A) \times B) \not\subseteq A)$

A/B : $\sigma_x(A) \setminus \text{Disqualified } x \text{ values}$



Examples

Reserves

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

Sailors

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

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

- **Solution 1:** $\sigma_{sname}(\sigma_{bid=103} \text{Reserves}) \bowtie \text{Sailors}$
- **Solution 2:** $\sigma_{sname}(\sigma_{bid=103} (\text{Reserves} \bowtie \text{Sailors}))$



Find names of sailors who've reserved a red boat

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

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

- ❖ A more efficient solution:

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

- ➔ 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:**

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

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


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):**

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

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

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


Find the names of sailors who've reserved all boats

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

$$\sigma_{(Tempoids, (\sigma_{sid, bid} Reserves) / (\sigma_{bid} Boats))}$$

$$\sigma_{sname}(Tempoids \bowtie Sailors)$$

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

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


Summary

- **Relational Algebra: a small set of operators mapping relations to relations**
 - Operational, in the sense that you specify the explicit order of operations
 - A *closed* set of operators! Can mix and match.
- **Basic ops include:** σ , π , \bowtie , \cup , \cap
- **Important compound ops:** σ , \bowtie , $/$