Ch 4. Relational Algebra and Calculus
- Ch 4.2 Relational Algebra -

Sang-Won Lee

http://icc.skku.ac.kr/~swlee

SKKU VLDB Lab.

(http://vldb.skku.ac.kr/)

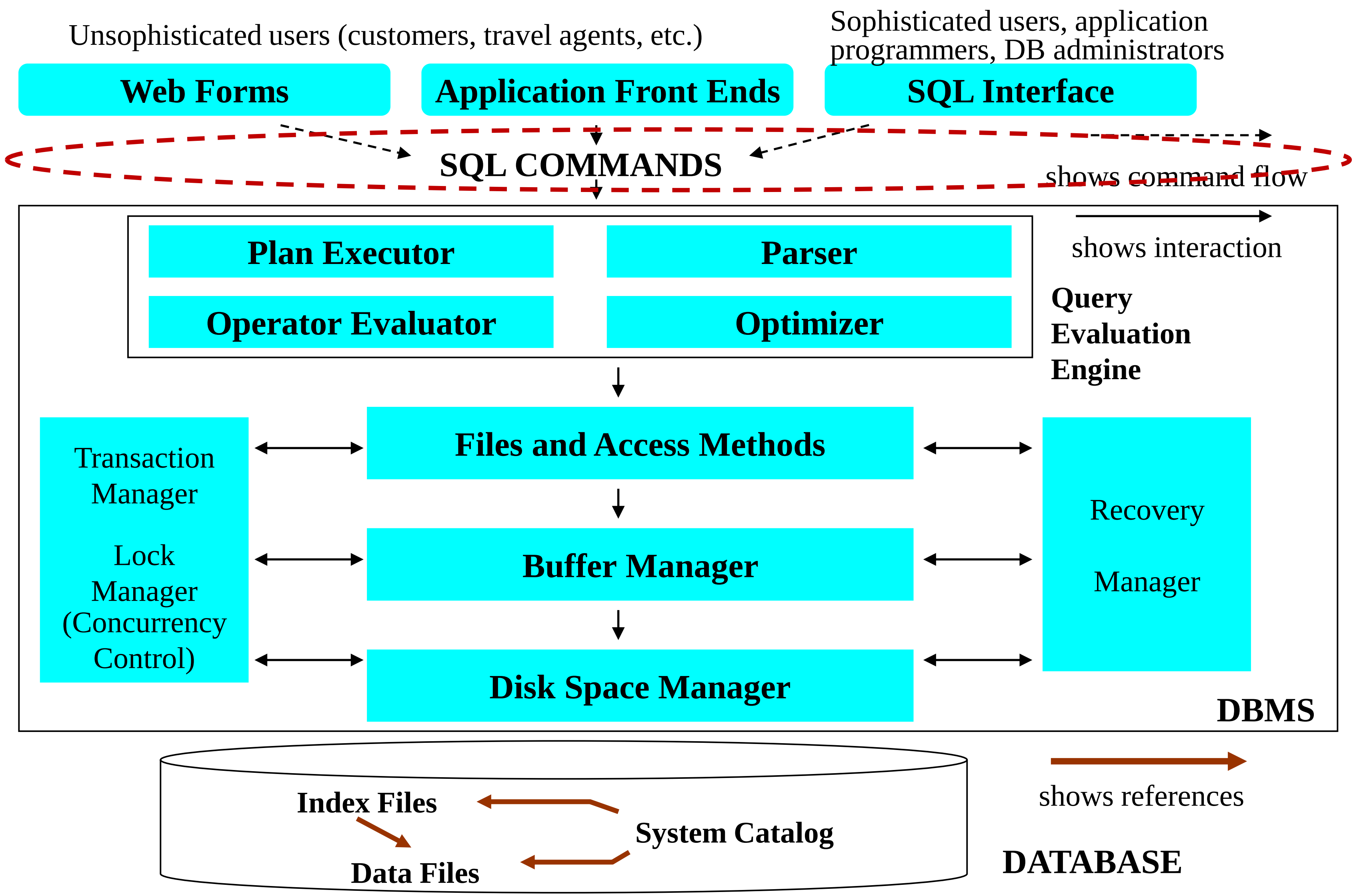
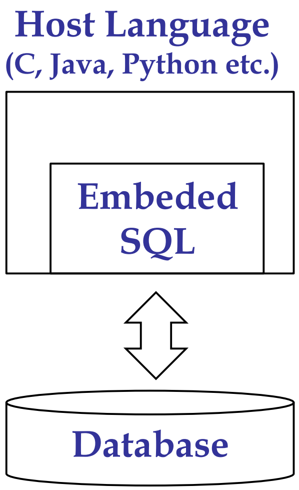


Figure 1.3 Anatomy of an RDBMS

4.0 Relational Query Languages

- Query languages*:* To manipulate and retrieve 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.
- Be very aware of the role of host and query language!!
  - e.g., Do joins using SQL; do not develop your own join logic using the host language .



Relational Database Language: Genealogy

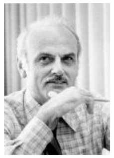
Relational Calculus

ALPHA
(by Codd, based on TRC)

QBE:Query By Example
(based on DRC)

QUEL (for Ingres,
based on TRC)

Relational Algebra(by E. F. Codd)



SEQUEL(by System R team)

SQL

ISO SQL Standards:

SQL86

SQL89

SQL2('92)

SQL3('1999)

SQL200n ….

SQL Analytics
(Window Functions – Ch 24.4)

Datalog
(Ch 24.1)

Codd's Theorem

- 3 Languages: (are equivalent in terms of expressive power)
  - Relational Algebra
  - Tuple Relational Calculus (safe expressions only)
  - Domain Relational Calculus (safe expressions only)

- Impact of Codd's theorem:
  - SQL itself is based on the relational calculus (TRC)
  - SQL implementation is based on relational algebra
  - Codd's theorem shows that SQL implementation is correct and complete.

(source: Tim Kraska's Lecture Note @ Fall 2018, MIT)

Formal Relational Query Languages

- Two mathematical query languages form the basis for "real" languages (e.g., SQL), and for implementation:
  - Relational Algebra (RA): More operational (or procedural); very useful for representing internal execution plans for SQLs
  - Relational Calculus (RC): Let users describe what they want, rather than how to compute it (Non-operational, *declarative*.)
    - ✓ Calculus: a system of computation; pebble (Latin);
- See this for comparing RA vs. RC. (https://techdifferences.com/difference-between-relational-algebra-and-relational-calculus.html)
- RA and RC are both relationally complete.
  - Relationally complete if a data language is at least as powerful as RA
  - Any query written in RC can be converted to RA and vice versa
    - ✓ V.s. general purpose host language: no loop, no conditional branch

What is an "Algebra"?

- Arabic *Al-jebr*: a resetting (of something broken) or a combination
- A mathematical system consisting of:
  - *Operands* --- variables or values from which new values can be constructed.
  - *Operators* --- symbols denoting procedures that construct new values from given values
  - *Axioms* – closure, commutative / associative / distributive laws, ..
- E.g. +, -, *, / on natural number; linear algebra
  - Input operands?
  - Operator's semantics?
  - Result type? Closed?

Linear Algebra matters!

- Data science, machine learning
- RDBMS and SQL do not support L.A. directly.
- Can we support L.A. on top of R.A.?
- Scalable Linear Algebra on a Relational Database System @ CACM 2020 August

Very Large Data Bases

What is Relational Algebra?

- Operands: relations or <u>variables</u> that represent relations.
- Operators are designed to do the most common things that we need to do with relations in a database.
  - Unary or binary operators
- **Closeness**: the result of any relational operator is another relation; thus the result can be used as an *input operand relation* for another relational operator.
- *Three Axioms*
  - *Commutativeness: R X S = S x R*
  - *Associativeness: (R x S) x T = R x (S x T)*
  - *Distributivenss: R x (S U T) = (R x S) U (R x T)*

4.1 Preliminaries

- A query is applied to relation instance(s), 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

S1 (Sailor)

sid	*sname*	*rating*	*age*
22	Dustin	7	45.0
31	Lubber	8	55.5
58	Rusty	10	35.0

S2 (Sailor)

sid	*sname*	*rating*	*age*
28	yuppy	9	35.0
31	Lubber	8	55.5
44	guppy	5	35.0
58	Rusty	10	35.0

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

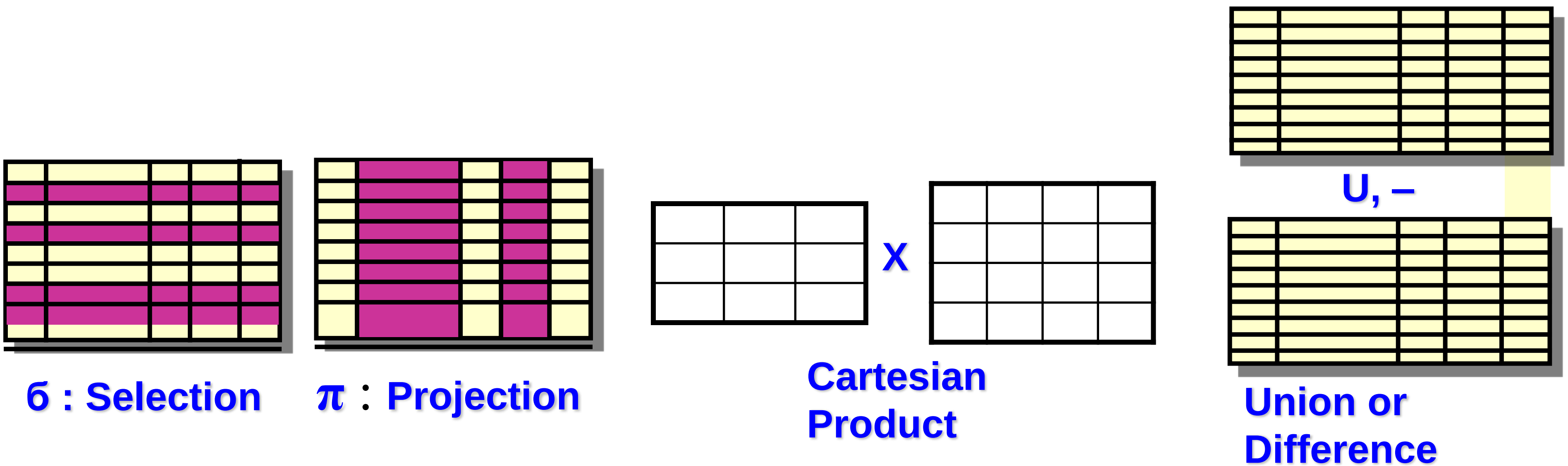
R1 (Reserve)

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

Boats

4.2 Relational Algebra

- Basic operations:
  - *Selection* (σ) : Selects a subset of rows from relation. (lowercase Greek letter **sigma**)
  - *Projection* (π) : Deletes unwanted columns from relation. (uppercase Greek letter **pi**)
  - *Cross-product* ($\times$) : Allows us to combine two relations. (Cartesian prod.)
  - *Set-difference* (—) : R1 – R2 - tuples in R1, but not in R2.
  - *Union* ($\cup$) : Tuples in either R1 and R2.



Relational Algebra

- Additional operations:
  - Intersection, join, division, renaming, aggregation:
  - Not essential, but (very) useful.
    - ✓ e.g. $R \cap S = R - (R - S)$

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

Very Large Data Bases

Projection

- Delete 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.

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

sname	rating
yuppy	9
lubber	8
guppy	5
rusty	10

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

Very Large Data Bases

Projection(2)

- Projection operator has to eliminate duplicates! (why?)
  - Note: real systems typically don't do duplicate elimination unless the user explicitly asks for it. (Why do they choose this semantic?)

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

→

age
35.0
55.5

$\pi_{age}(S2)$

Very Large Data Bases

Selection

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

- Selects rows that satisfy *selection condition*.
  - Table = a set of propositions(or tuples)

- No duplicates in result! (Why?)

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

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

- Schema of result identical to schema of (only) input relation.

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

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

sname	rating
yuppy	9
rusty	10

Union, Intersection, Set-Difference

- All of these operations take two input relations (thus, binary operator), 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 (or Cartesian Product)

- R1 X S1: each row of S1 is paired with each row of R1.
  - How many rows in result?
- *Result schema* has one field per field of S1 and R1
  - Each field name from each table is `inherited' (if possible).
  - ***Naming conflict***: Both S1 and R1 have a field called *sid*.
    - ✓ Two (sid)s in the result of S1 X R1 are unnamed.

S1

sid	*sname*	*rating*	*age*
22	Dustin	7	45.0
31	Lubber	8	55.5
58	Rusty	10	35.0

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

R1

(*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

- Thus, **renaming** operator: e.g. $\rho\,(C(1 \rightarrow sid1, 5 \rightarrow sid2),\ S1 \times R1)$
  - ✓ lowercase Greek letter **rho**

SQL vs. Relation Algebra

```
SELECT   A1, A2, …, An     < - -  Projection (3)
FROM     T1, T2, … Tm      < - -  Cartesian product (1)
WHERE    P                 < - -  Selection (2)
```

- The above SFW (<u>S</u>elect-<u>F</u>rom-<u>W</u>here) syntax covers <u>selection</u>, <u>projection</u> and <u>Cartesian product</u>

$$\pi_{A1,A2,\ldots,An}(\sigma_P(T1 \times T2 \times \ldots \times Tm))$$

  – What if any Ti is empty?

Joins

1. 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.
  - A θ B, where θ is =, <, etc.; hence the name "theta-join."

Joins(2)

2. 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.

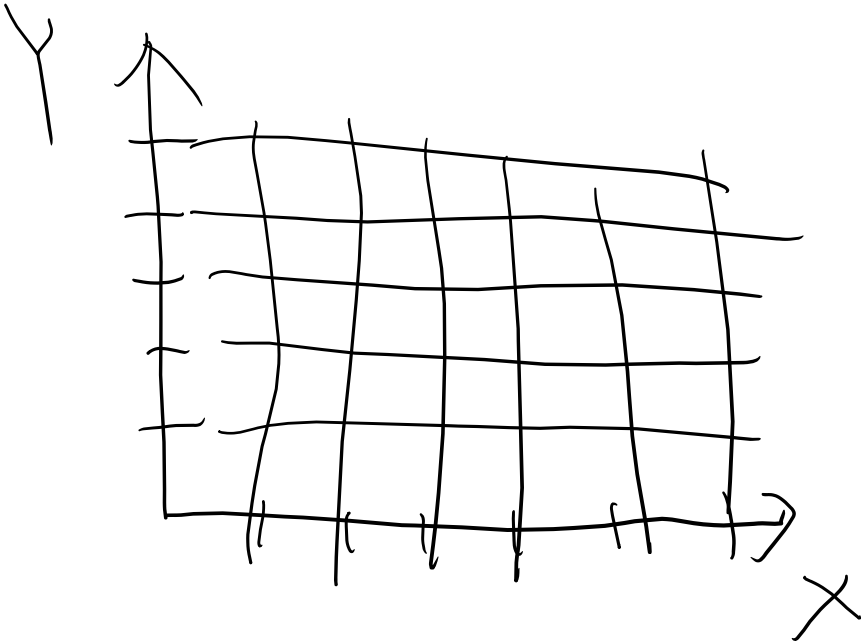
Joins(3)

3. Natural Join: Equi-join on *all* common fields.

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

$$S1 \bowtie R1$$

Cross-Product vs. Join



$y = x^2$

$y = x + 1$

$\sigma_c \quad (R \times S)$

Set vs. Bag Semantics

- Relational Algebra vs. SQL
  - "Set" vs. "multi-set"(or bag) semantics
  - By default, SQL takes bag semantics
- A *bag* is like a set, but an element may appear more than once
  - e.g. {1,2,1,3} is a bag. {1,2,3} is also a bag that happens to be a set.
  - Bags also resemble lists, but order in a bag is unimportant.
    - ✓ e.g. {1,2,1} = {1,1,2} as bags, but [1,2,1] != [1,1,2] as lists.
- Why SQL has taken the bag semantics?
  - Some operations, like projection, are more efficient on bags than sets
  - Duplicates will be eliminated if explicitly asked via "distinct" keyword

Division

- Not primitive operator, but useful for expressing queries like *"Find sailors who have reserved **all** boats"*
  - Inverse of Cartesian Product: (R X S) / S = R
  - Algebraic counterpart to the universal quantifier in relational calculus
- Let *A* have 2 fields, *x* and *y*; *B* have only field *y*:
  - *A/B* = $\left\{ \langle x \rangle \mid \exists \langle x, y \rangle \in A \ \forall \langle y \rangle \in B \right\}$
  - 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*".
  - *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

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 operator; 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: $\underline{\pi_x((\underline{\pi_x(A) \times B}) - A)}$

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

Very Large Data Bases

Relational Algebra vs. Calculus: A/B Division Example

- Relational algebra: procedural, operational

Disqualified x values: $\underline{\pi_X((\underline{\pi_X(A) \times B}) - A)}$

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

- Relational calculus (p.121 in textbook) : non-procedural, declarative

(Q9) *Find the names of sailors who have reserved all boats.*

$$\{P \mid \exists S \in Sailors\ \ \forall B \in Boats$$
$$(\exists R \in Reserves(S.sid = R.sid \wedge R.bid = B.bid \wedge P.sname = S.sname))\}$$

This query was expressed using the division operator in relational algebra. Note how easily it is expressed in the calculus. The calculus query directly reflects how we might express the query in English: "Find sailors S such that for all boats B there is a Reserves tuple showing that sailor S has reserved boat B."

“Relational Thinking” DOES matter!

4.2.6 More Examples of Algebra Queries

- Example schema

Boats

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

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

Reserves

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

M:N relationship
between Sailors and Boats

Q1: Find names of sailors who've reserved boat #103

- Solution 1:

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

- Solution 2:

$$\rho\ (Temp1,\ \sigma_{bid=103} Reserves)$$

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

$$\pi_{sname}(Temp2)$$

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

- Solution 3:
  - *You can use assignment operator, instead of renaming.*

$$\rho(Temp1, \sigma_{bid=103} Reserves) \Rightarrow Temp1 \leftarrow \sigma_{bid=103} Reserves$$

Q2: 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!

- Q2 vs. Q2': ….. reserved one and only one red boat
  - Q2 returns sailors who reserved at least one red boat

Q5: 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:

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

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

Boats

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

- Can also define Tempboats using union! (How?)

- What happens if $\vee$ is replaced by $\wedge$ in this query?

Very Large Data Bases

Q6: 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):

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

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

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

Q9: 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:

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

Very Large Data Bases

Building Complex Expressions in R.A.

- An algebra system allows to express sequences of operations in a natural way. (procedural vs. declarative)
  - Example: in arithmetic --- $(x + 4)*(y - 3)$.

- Relational algebra also does.

- Three notations, just as in arithmetic:
  1. Expressions with several operators.
  2. Sequences of assignment statements.
  3. Expression trees.

E.g. R x S x T

1. (R x S) x T

2. Temp1 = R x S
   Temp2 = Temp1 x T
   Result = Temp 2

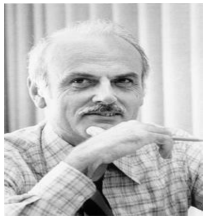
3.
```
        x
       / \
      x   T
     / \
    R   S
```

Summary

- The relational model provides simple but powerful query language based on formal mathematics
- Relational algebra is more operational; useful as internal representation for query evaluation plans.
- Several ways of expressing a given query; a query optimizer try to choose the most efficient version.
- **Great debate @ 1975**

Network Camp:
Programmer as Navigator

Vs.



Relational Camp:
Application Development Productivity
Data Independence
High Level Query Language