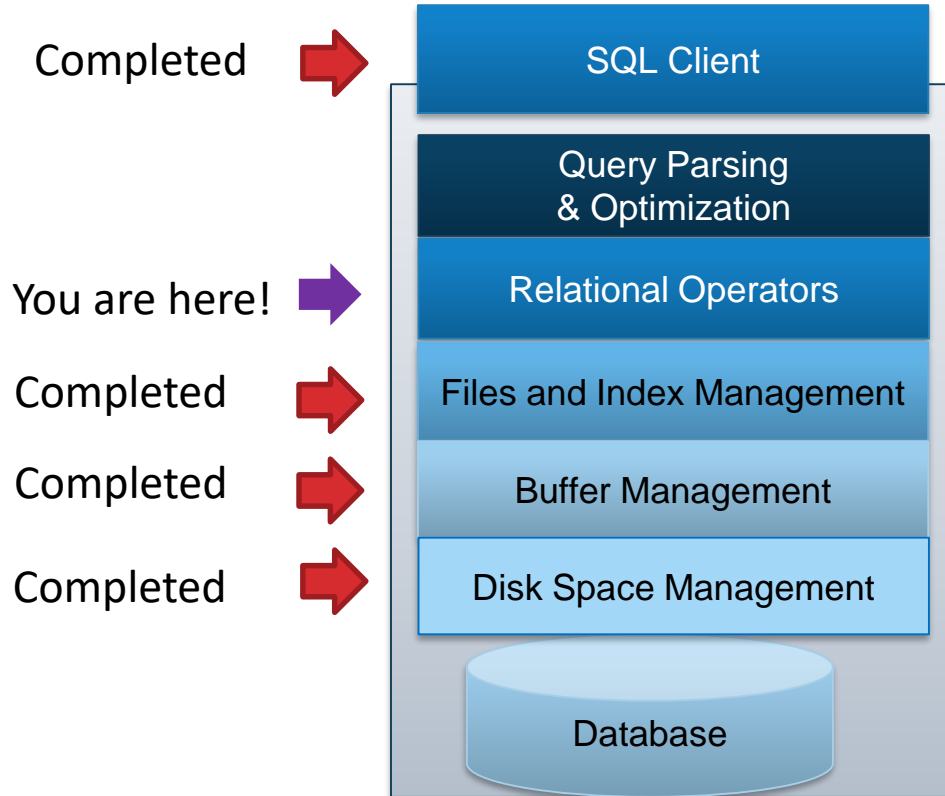


# Relational Algebra

R & G, Chapters 4.1 - 4.2



# Architecture of a DBMS: What we've learned



Today: *definitions* of the relational operators.

Coming soon: *implementations*

# An Overview of the Layer Above

## SQL Query

```
SELECT S.name
FROM Reserves R, Sailors S
WHERE R.sid = S.sid
AND R.bid = 100
AND S.rating > 5
```

Query Parser  
& Optimizer

## Relational Algebra

$$\pi_{S.name}(\sigma_{bid=100 \wedge rating > 5}(\text{Reserves} \bowtie_{R.sid=S.sid} \text{Sailors}))$$

Equivalent to...

## Optimized (Physical) Query Plan:

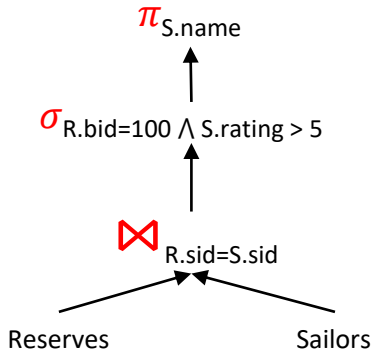
On-the-fly  
Project Iterator

On-the-fly  
Select Iterator

Indexed Nested  
Loop Join Iterator

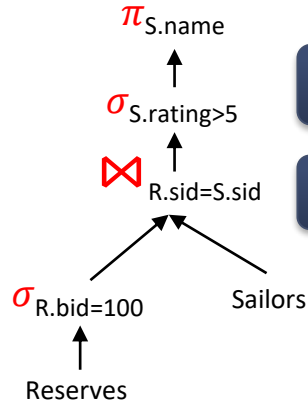
Heap Scan  
Iterator

## (Logical) Query Plan:



But actually will  
produce...

Operator Code  
B+-Tree  
Indexed Scan  
Iterator



# SQL vs Relational Algebra

## SQL Query

```
SELECT S.name
FROM Reserves R, Sailors S
WHERE R.sid = S.sid
AND R.bid = 100
AND S.rating > 5
```

Query Parser  
& Optimizer

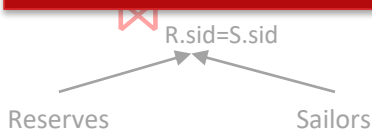
## Relational Algebra

$$\pi_{S.name}(\sigma_{bid=100 \wedge rating > 5}(\text{Reserves} \bowtie_{R.sid=S.sid} \text{Sailors}))$$

Equivalent to...

SQL

A **declarative** expression  
of the query result



Relational Algebra

**Operational** description of  
a computation.

But actually will  
produce...

Operator Code

B+-Tree  
Indexed  
Iterator

Systems execute relational algebra  
query plan.

On-the-fly

tor

# SQL (Structured Query Language)

```
SELECT S.name  
FROM Reserves R, Sailors S  
WHERE R.sid = S.sid  
AND R.bid = 100  
AND S.rating > 5
```

- Key **System** Features: *Why do we like SQL*
  - Declarative:
    - Say **what** you want, not **how** to get it
    - Enables system to optimize the **how**
- Foundation in formal Query Languages
  - **Relational Calculus**

# History: Formal Relational QL's

- **Relational Calculus:** (Basis for SQL)

- Describe the result of computation
- Based on first order logic
- Tuple Relational Calculus (**TRC**)
  - $\{S \mid S \in \text{Sailors} \exists R \in \text{Reserves} (R.\text{sid} = S.\text{sid} \wedge R.\text{bid} = 103)\}$



Are these equivalent?

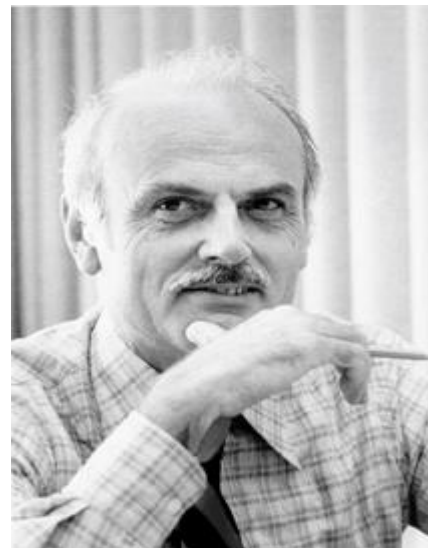
Can we go from one to the other?

- **Relational Algebra:**

- Algebra on sets
- Operational description of transformations

# Codd's Theorem

- Established equivalence in expressivity between :
  - Relational Calculus
  - Relational Algebra
- Why an important result?
  - Connects declarative representation of queries with operational description
  - Constructive: we can compile SQL into relational algebra



Edgar F. "Ted" Codd  
(1923 - 2003)  
Turing Award 1981





# Relational Algebra Preliminaries

- Algebra of operators on relation instances
- $\pi_{S.name}(\sigma_{R.bid=100 \wedge S.rating>5}(R \bowtie_{R.sid=S.sid} S))$ 
  - Closed: result is also a relation instance
    - Enables rich composition!
  - Typed: input schema determines output
    - Can statically check whether queries are legal.

# Relational Algebra and Sets

- Pure relational algebra has set semantics
  - No duplicate tuples in a relation instance
  - vs. SQL, which has multiset (bag) semantics
  - We will switch to multiset in the system discussion

# Relational Algebra Operators: Unary

- Unary Operators: on **single relation**
- **Projection** ( $\pi$ ): Retains only desired columns (vertical)
- **Selection** ( $\sigma$ ): Selects a subset of rows (horizontal)
- **Renaming** ( $\rho$ ): Rename attributes and relations.

# Relational Algebra Operators: Binary

- Binary Operators: on **pairs of relations**
- **Union** ( $\cup$ ): Tuples in  $r_1$  or in  $r_2$ .
- **Set-difference** ( $-$ ): Tuples in  $r_1$ , but not in  $r_2$ .
- **Cross-product** ( $\times$ ): Allows us to combine two relations.

# Relational Algebra Operators: Compound

- Compound Operators: common “*macros*” for the above
- **Intersection** (  $\cap$  ): Tuples in r1 and in r2.
- **Joins** (  $\bowtie_{\theta}$  ,  $\bowtie$  ): Combine relations that satisfy predicates

# Projection ( $\pi$ )

- Corresponds to the SELECT list in SQL
- Schema determined by schema of attribute list
  - Names and types correspond to input attributes
- *Selects a subset of columns (vertical)*

$\pi_{\text{sname, age}}(S2)$

Relational Instance S2

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

List of Attributes



sname	age
yuppy	35.0
lubber	55.5
guppy	35.0
rusty	35.0

# Projection ( $\pi$ ), cont.

- Set semantics  $\rightarrow$  results in fewer rows
  - Real systems don't automatically remove duplicates
    - Why? (Semantics and Performance reasons)
  - *Selects a subset of columns (vertical)*

$$\pi_{\text{age}}(S2)$$

Relational Instance S2

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

Multiset

age
35.0
55.5
35.0
35.0

Set

age
35.0
55.5

# Selection( $\sigma$ )


- Corresponds to the WHERE clause in SQL
- Output schema same as input
- Duplicate Elimination? Not needed.
- *Selects a subset of rows (horizontal)*

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

Relational Instance S2

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

Selection Condition (Boolean Expression)

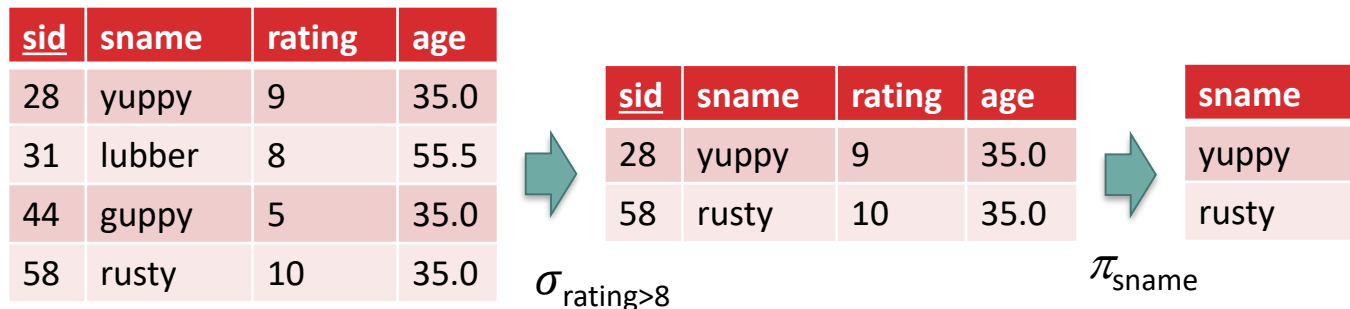


<u>sid</u>	sname	rating	age
28	yuppy	9	35.0
58	rusty	10	35.0



# Composing Select and Project

- Names of sailors with rating > 8:  $\pi_{\text{sname}}(\sigma_{\text{rating}>8}(S2))$



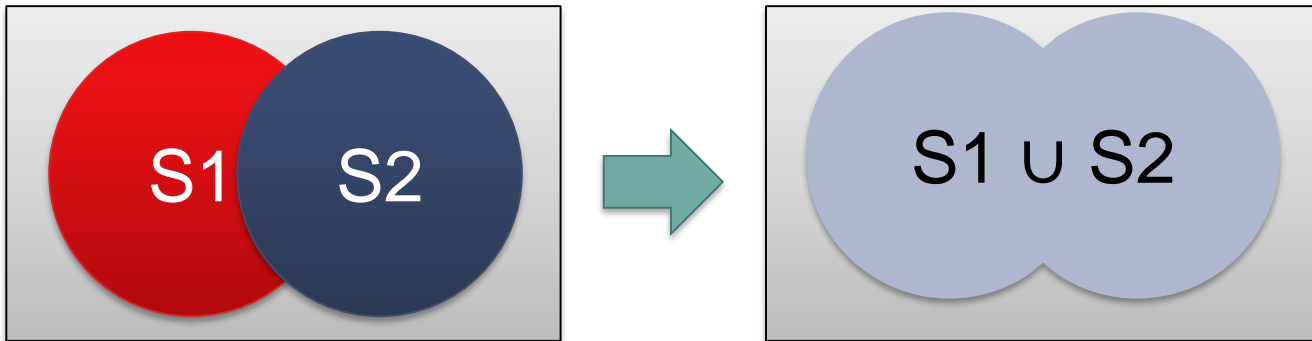
- What about:  $\sigma_{\text{rating}>8}(\pi_{\text{sname}}(S2))$ 
  - Invalid types. Input to  $\sigma_{\text{rating}>8}$  does not contain rating.*



# Union ( $\cup$ )

- Two input relations, must be *compatible*:
  - Same number of fields
  - Fields in corresponding positions have same **type**
- SQL Expression: UNION

$S1 \cup S2$



# Union (U) VS Union ALL

- Duplicate elimination in practice?
  - SQL's UNION vs UNION ALL

Relational Instance **S1**

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

Relational Instance **S2**

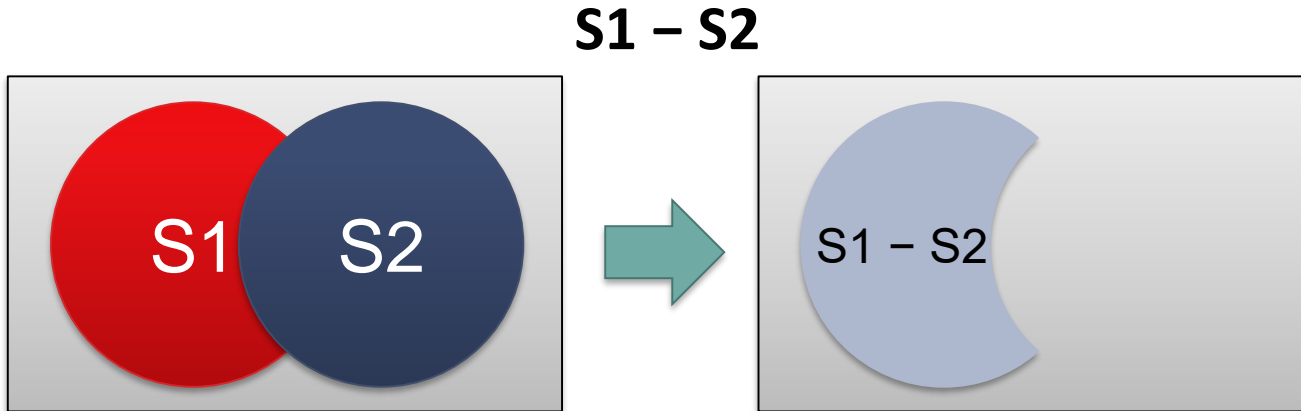
<u>sid</u>	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

**S1 U S2**

<u>sid</u>	sname	rating	age
22	dustin	7	45
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

# Set Difference ( $-$ )

- Same as with union, both input relations must be *compatible*.
- SQL Expression: EXCEPT



# Set Difference ( – ), cont.

- Duplicate elimination?
  - Not required
- EXCEPT vs EXCEPT ALL

Relational Instance S1

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

Relational Instance S2

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

## S1 – S2

<u>sid</u>	sname	rating	age
22	dustin	7	45

## S2 – S1

<u>sid</u>	sname	rating	age
28	yuppy	9	35.0
44	guppy	5	35.0

# Cross-Product (×)

- $R1 \times S1$ : Each row of  $R1$  paired with each row of  $S1$

R1:

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

×

S1:

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

=

sid	bid	day	sid	sname	rating	age
22	101	10/10/96	22	dustin	7	45.0
22	101	10/10/96	31	lubber	8	55.5
22	101	10/10/96	58	rusty	10	35.0
58	103	11/12/96	22	dustin	7	45.0
58	103	11/12/96	31	lubber	8	55.5
58	103	11/12/96	58	rusty	10	35.0

- *How many rows in result?  $|R1| \times |R2|$*
- *Schema compatability? Not needed.*
- *Duplicates? None generated.*

# Renaming ( $\rho$ = “rho” )

- *Renames relations and their attributes:*
- Note that relational algebra doesn't require names.
  - We could just use positional arguments.

$$\rho(\underbrace{\text{Temp1}}_{\text{Output Relation Name}}(\underbrace{1 \rightarrow \text{sid1}, 4 \rightarrow \text{sid2}}_{\substack{\text{Renaming List} \\ \text{position} \rightarrow \text{New Name}}}, \underbrace{R1 \times S1}_{\text{Input Relation}})$$

$R1 \times S1$

sid	bid	day	sid	sname	rating	age
22	101	10/10/96	22	dustin	7	45.0
22	101	10/10/96	31	lubber	8	55.5
22	101	10/10/96	58	rusty	10	35.0
58	103	11/12/96	22	dustin	7	45.0
58	103	11/12/96	31	lubber	8	55.5
58	103	11/12/96	58	rusty	10	35.0



$\text{Temp1}$

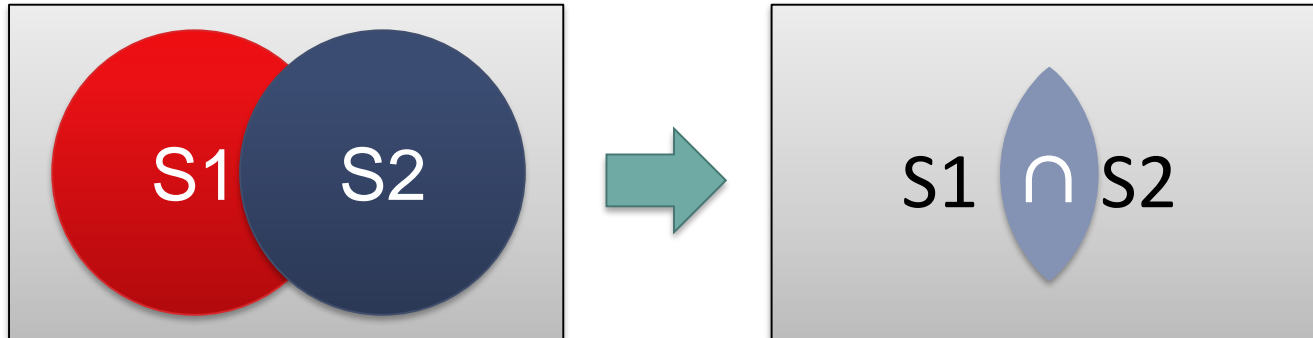
sid1	bid	day	sid2	sname	rating	age
22	101	10/10/96	22	dustin	7	45.0
22	101	10/10/96	31	lubber	8	55.5
22	101	10/10/96	58	rusty	10	35.0
58	103	11/12/96	22	dustin	7	45.0
58	103	11/12/96	31	lubber	8	55.5
58	103	11/12/96	58	rusty	10	35.0



# Compound Operator: Intersection

- Same as with union, both input relations must be *compatible*.
- SQL Expression: INTERSECT

$$S1 \cap S2$$



# Intersection ( $\cap$ )

- Equivalent to:  
 $S1 \text{ --- } (S1 \text{ --- } S2)$

## $S1 \cap S2$

<u>sid</u>	sname	rating	age
31	lubber	8	55.5
58	rusty	10	35.0

Relational Instance **S1**

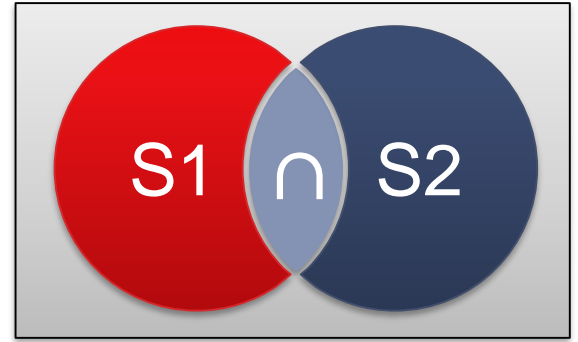
<u>sid</u>	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

Relational Instance **S2**

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

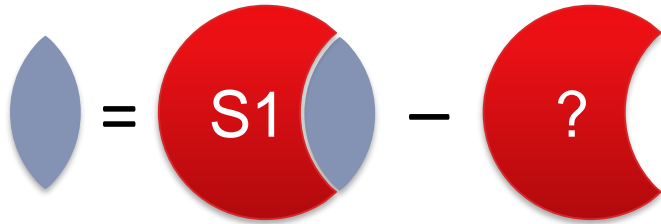
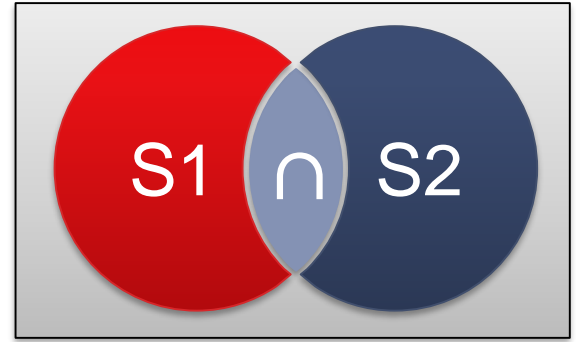
# Intersection ( $\cap$ ), Pt 2

- $S1 \cap S2 = ?$



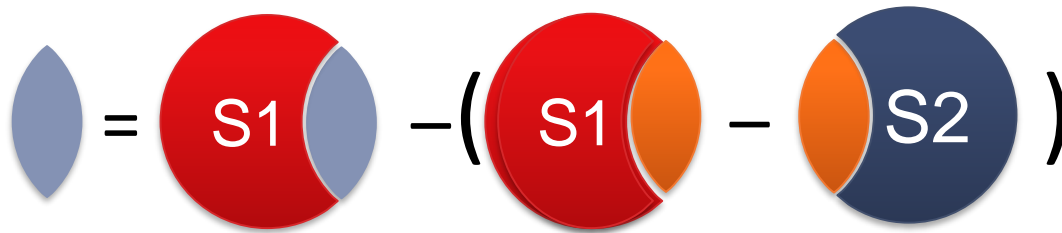
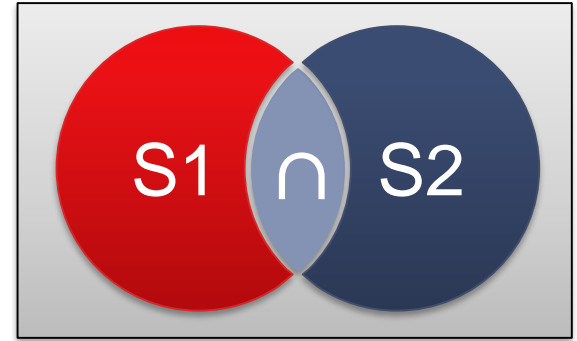
# Intersection ( $\cap$ ), Pt 3

- $S1 \cap S2 = S1 - ?$



# Intersection ( $\cap$ ), Pt 4

- $S1 \cap S2 = S1 - (S1 - S2)$



- Is intersection monotonic?

$$R_1 \subseteq R_2 \Rightarrow S \cap R_1 \subseteq S \cap R_2$$



# Compound Operator: Join

- Joins are compound operators (like intersection):
  - Generally,  $\sigma_{\theta}(R \times S)$
- Hierarchy of common kinds:
  - **Theta Join** ( $\bowtie_{\theta}$ ): join on logical expression  $\theta$ 
    - **Equi-Join**: theta join with theta being a conjunction of equalities
      - **Natural Join** ( $\bowtie$ ): equi-join on all matching column names

Note: we will need to learn a good join algorithm.

Avoid cross-product if we can!!

# Theta Join ( $\bowtie_{\theta}$ ) Example

- $R1 \bowtie_{\text{sid}=\text{sid}} S1$

**R1:**

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

**S1:**

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

$\bowtie_{\text{sid}=\text{sid}}$

**Result:**

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

- Note that output needs a rename operator!



# Another Theta Join ( $\bowtie_{\theta}$ ) Example<sub>s1:</sub>

- $R \bowtie_{\theta} S = \sigma_{\theta}(R \times S)$
- **Example:** *More senior sailors for each sailor.*
- $S1 \bowtie_{f4 < f8} S1$

f1	f2	f3	f4
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

S1				S1			
f1	f2	f3	f4	f5	f6	f7	f8
22	dustin	7	45.0	22	dustin	7	45.0
22	dustin	7	45.0	31	lubber	8	55.5
22	dustin	7	45.0	58	rusty	10	35.0
31	lubber	8	55.5	22	dustin	7	45.0
31	lubber	8	55.5	31	lubber	8	55.5
31	lubber	8	55.5	58	rusty	10	35.0
58	rusty	10	35.0	22	dustin	7	45.0
58	rusty	10	35.0	31	lubber	8	55.5
58	rusty	10	35.0	58	rusty	10	35.0

# Another Theta Join ( $\bowtie_{\theta}$ ), Pt 2

- $R \bowtie_{\theta} S = \sigma_{\theta}(R \times S)$
- **Example:** *More senior sailors for each sailor.*
- $S1 \bowtie_{\text{age} < \text{age2}} S1$

S1:

f1	f2	f3	f4
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

S1				S1			
f1	f2	f3	f4	f5	f6	f7	f8
22	dustin	7	45.0	22	dustin	7	45.0
22	dustin	7	45.0	31	lubber	8	55.5
22	dustin	7	45.0	58	rusty	10	35.0
31	lubber	8	55.5	22	dustin	7	45.0
31	lubber	8	55.5	31	lubber	8	55.5
31	lubber	8	55.5	58	rusty	10	35.0
58	rusty	10	35.0	22	dustin	7	45.0
58	rusty	10	35.0	31	lubber	8	55.5
58	rusty	10	35.0	58	rusty	10	35.0

# Another Theta Join ( $\bowtie_{\theta}$ ), Pt 3

- $R \bowtie_{\theta} S = \sigma_{\theta}(R \times S)$
- **Example:** *More senior sailors for each sailor.*
- $S1 \bowtie_{f4 < f8} S1$

S1:

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

S1				S1			
sid	sname	rating	age	sid	sname	rating	age2
22	dustin	7	45.0	31	lubber	8	55.5
58	rusty	10	35.0	22	dustin	7	45.0
58	rusty	10	35.0	31	lubber	8	55.5

- Result *schema* same as that of cross-product.
- Special Case:
  - **Equi-Join:** *theta join with AND of = predicates*
    - Special special case **Natural Join** ...

# Natural Join ( $\bowtie$ )

- Special case of equi-join in which equalities are specified for all matching fields and duplicate fields are projected away

$$R \bowtie S = \pi_{\text{unique fld.}} \sigma_{\text{eq. matching fld.}} (R \times S)$$

- Compute  $R \times S$
- Select rows where fields appearing in both relations have equal values
- Project onto the set of all unique fields.

# Natural Join ( $\bowtie$ ) Pt 2

- $R \bowtie S = \pi_{\text{unique fld.}} \sigma_{\text{eq. matching fld.}} (R \times S)$

$R1 \bowtie S1$

sid	bid	day	sid	sname	rating	age
22	101	10/10/96	22	dustin	7	45.0
22	101	10/10/96	31	lubber	8	55.5
22	101	10/10/96	58	rusty	10	35.0
58	103	11/12/96	22	dustin	7	45.0
58	103	11/12/96	31	lubber	8	55.5
58	103	11/12/96	58	rusty	10	35.0

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

# Natural Join ( $\bowtie$ ), Pt 3

- $R \bowtie S = \pi_{\text{unique fld.}} \sigma_{\text{eq. matching fld.}} (R \times S)$

$R1 \bowtie S1$

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

R1:

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

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

- Commonly used for foreign key joins (as above).



# Extended Relational Algebra

- Group By / Aggregation Operator ( $\gamma$ ):
  - $\gamma_{\text{age, AVG(rating)}}(\text{Sailors})$
  - With selection (HAVING clause):
    - $\gamma_{\text{age, AVG(rating), COUNT(*) > 2}}(\text{Sailors})$
- Textbook uses two operators:
  - GROUP BY age, AVG(rating) (Sailors)
  - HAVING COUNT(\*) > 2  
(GROUP BY age, AVG(rating)(Sailors))



# 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! Mix and match.
- Basic ops include:  $\sigma$ ,  $\pi$ ,  $\times$ ,  $\cup$ ,  $-$
- Important compound ops:  $\cap$ ,  $\bowtie$