Relational Algebra

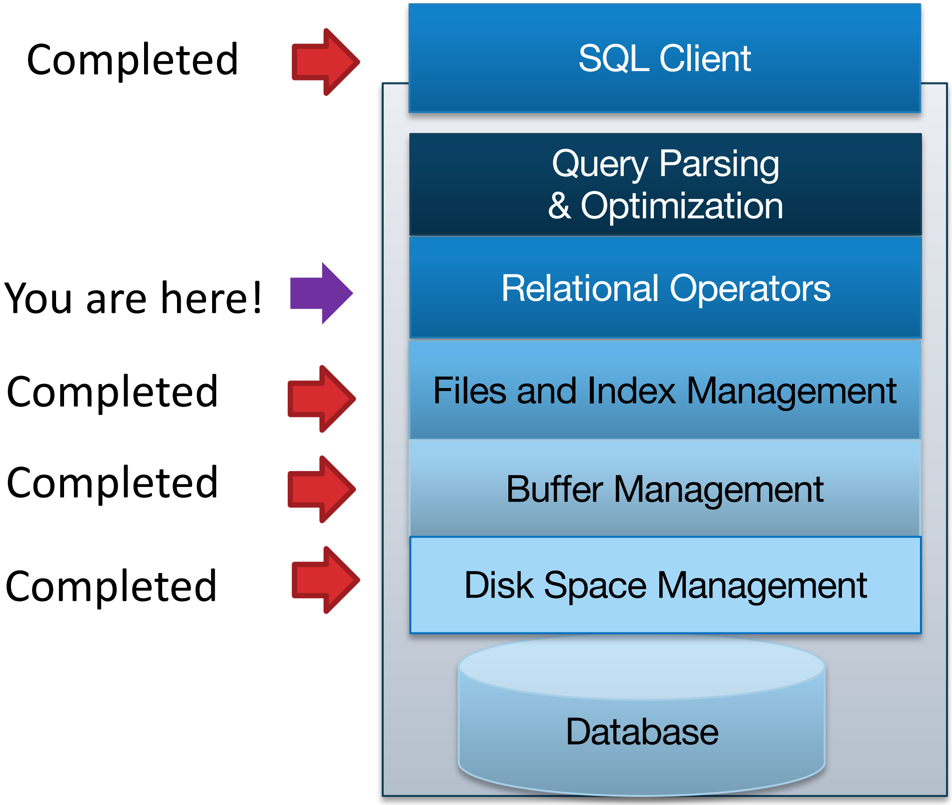
Alvin Cheung

Aditya Parameswaran

R & G, Chapters 4.1 - 4.2

Berkeley
cs186

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
```

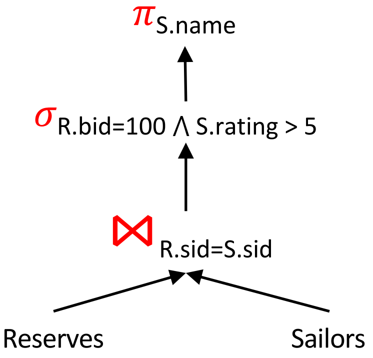
Query Parser & Optimizer

Relational Algebra

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

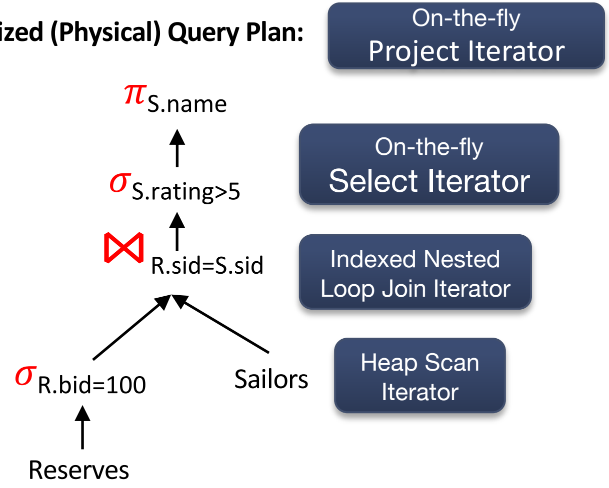
Equivalent to…

(Logical) Query Plan:



But actually will produce…

Optimized (Physical) Query Plan:



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

But actually will produce...

Relational Algebra

Operational description of a computation.

Systems execute relational algebra query plan.

On-the-fly

R.sid=S.sid

Reserves

Sailors

Operator Code

B+-Tr

Indexed Iterat

SQL vs. Relational Algebra

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

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

- *Why do humans like SQL*
  - It's declarative
  - Say ***what*** you want, not ***how*** to get it
  - Enables system to optimize the ***how***
- *Why do systems like rel. algebra*
  - It's operational
  - It describes the steps for ***how*** to compute a query result
- DBMSs internally transform SQL into relational algebra expressions, manipulate and simplify it, and figure out the best operational mechanism to compute the SQL query result

Relational Algebra Preliminaries

- Algebra of operators on relation instances
  - Just like other algebras: linear algebra or elementary algebra
    - Operating on matrices or variables
- $\pi_{S.name}(\sigma_{R.bid=100 \wedge S.rating>5}(R \bowtie_{R.sid=S.sid} S))$
- Closed: result is also a relation instance
  - Enables rich composition!
  - Just like a linear algebraic expression on matrices returns a matrix
- Typed: input schema determines output
  - Can statically check whether queries are legal.
  - Same story for linear algebra – input sizes determine output sizes

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** (π): Retains only desired columns (vertical)
- **Selection** (σ): Selects a subset of rows (horizontal)
- **Renaming** (ρ): Rename attributes and relations.

Relational Algebra Operators: Binary

- <u>Binary Operators</u>: on **pairs of relations**
- **Union** ($\cup$): Tuples in r1 or in r2.
- **Set-difference** (—): Tuples in r1, but not in r2.
- **Cross-product** ($\times$): Allows us to combine two relations.

Relational Algebra Operators: Compound

- Compound Operators: common "*macros*" for the 6 unit ops above
- **Intersection** ($\cap$): Tuples in r1 and in r2.
- **Joins** ($\bowtie_{\theta}$, $\bowtie$): Combine relations that satisfy predicates

Relational Algebra contd.

Alvin Cheung
Aditya Parameswaran

R & G, Chapters 4.1 - 4.2

Announcements

- Look at the weekly post!
- We've noticed OH ticket descriptions becoming a bit sparse, making it harder for us to help you
  - We're starting to enforce some minimal requirements to help with OH tickets (see @19) Specify:
    - subpart
    - description of your problem/bug
    - debugging steps taken (i.e. writing tests, running IntelliJ debugger),
    - link to updated GitHub repo
  - Full details in @19
- Turn on your video if you can!

Relational Algebra Operators

- **Projection** (π): Retains only desired columns (vertical)
- **Selection** (σ): Selects a subset of rows (horizontal)
- **Renaming** (ρ): Rename attributes and relations
- **Union** ($\cup$): Tuples in r1 or in r2.
- **Set-difference** ($-$): Tuples in r1, but not in r2.
- **Cross-product** ($\times$): Allows us to combine two relations.

- **Intersection** ($\cap$): Tuples in r1 and in r2.
- **Joins** ($\bowtie_\theta$, $\bowtie$): Combine relations that satisfy predicates

Projection (π)

- 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)*

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

List of Attributes

Relational *Instance* **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

sname	age
yuppy	35.0
lubber	55.5
guppy	35.0
rusty	35.0

Projection (π), cont.

- Set semantics → results in fewer rows
  - Real systems don't automatically remove duplicates
    - Why? (Semantics and Performance reasons)

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

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(σ)

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

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

Selection Condition (Boolean Expression)

Relational *Instance* **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

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

Composing Select and Project

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

<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

$\sigma_{rating>8}$

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

π_{sname}

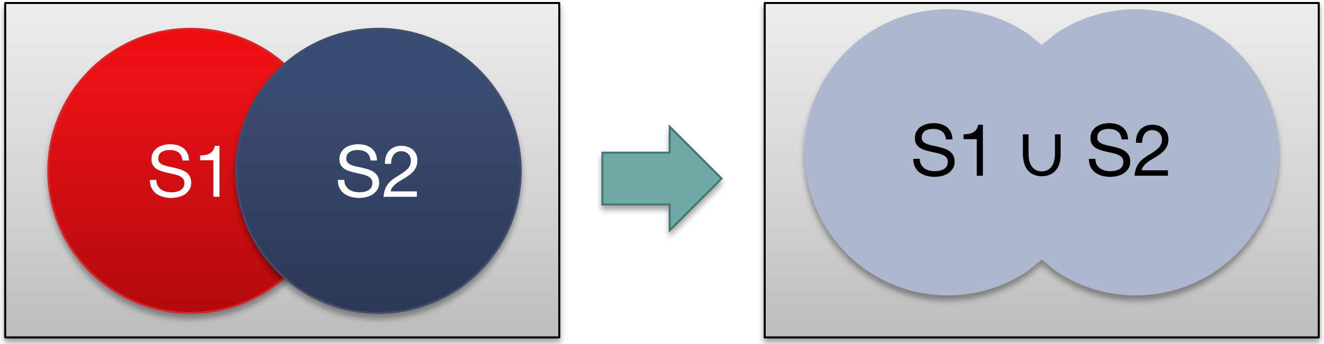
sname
yuppy
rusty

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

Union (∪)

- *Takes the set union of two sets*
- The two input relations must be *compatible*:
  - Same sequence of attributes and types thereof
- SQL Expression: UNION

S1 U S2



Union (∪) VS Union ALL

- In union under set semantics, duplicate elimination is needed
- SQL Expression: UNION (get rid of duplicates) vs. UNION ALL (keep dup.)

Relational *Instance* **S1**

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

Relational *Instance* **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

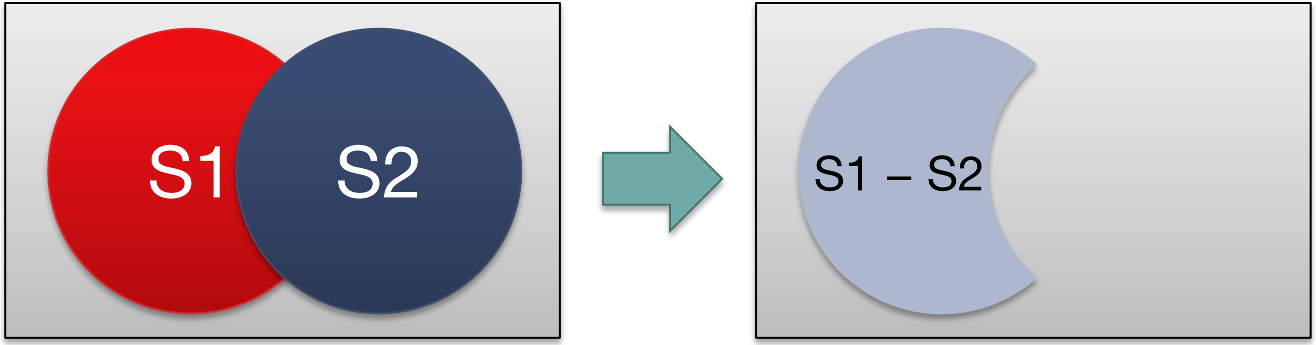
S1 ∪ S2

sid	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

S1 – S2



Set Difference (–), cont.

- Q: Do we need to eliminate duplicates like in UNION?
  - Not required if inputs are sets
- SQL Expression: EXCEPT vs EXCEPT ALL
  - Same as UNION/UNION ALL
  - In EXCEPT duplicates are eliminated if they exist

Relational *Instance* **S1**

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

Relational *Instance* **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

S1 – S2

sid	sname	rating	age
22	dustin	7	45

S2 – S1

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

Cross-Product (×)

- **R1 × S1:** *Each row of* ***R1*** *paired with each row of* ***S1***

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

=

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|*|S1|*
- *Do we need to worry about schema compatibility?*
  - *Not needed.*
- *Do we need to do duplicate elimination?*
  - *None generated.*

Renaming (ρ = "rho")

- *Renames relations and their attributes*
- Convenient to avoid confusion when two relations overlap in attributes
- Can omit output name if we don't want to rename the output

R1:

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

$$\rho_{R(sid2,\ bid2,\ day)}\ R1$$

$$\rho_{R(sid\text{->}sid2,\ bid\text{->}bid2)}\ R1$$

R:

sid2	bid2	day
22	101	10/10/96
58	103	11/12/96

Renaming (ρ = “rho”) contd.

- Yet another shorthand for renaming

$$\rho_{Temp1(1 \rightarrow sid1,\ 4 \rightarrow sid2)}(R1 \times S1)$$

Output Name: Temp1

Renaming List position → New Name: (1 → sid1, 4 → sid2)

Input Relation: (R1 × S1)

- Again, can omit output name if we don’t want to rename the output
- For this case, can equivalently name each relation and then do cross-product

R1 × 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

→

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

Relational Algebra Operators

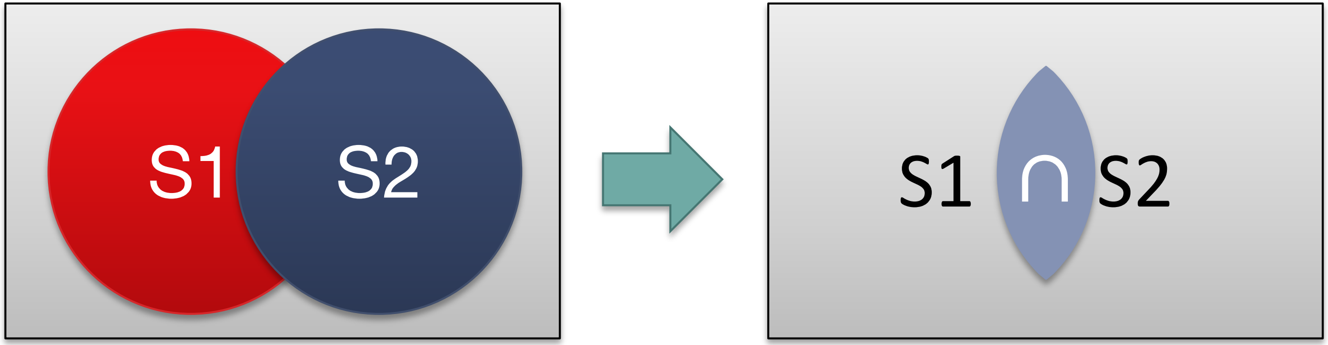
- **Projection** (π): Retains only desired columns (vertical)
- **Selection** (σ): Selects a subset of rows (horizontal)
- **Renaming** (ρ): Rename attributes and relations
- **Union** ($\cup$): Tuples in r1 or in r2.
- **Set-difference** ($-$): Tuples in r1, but not in r2.
- **Cross-product** ($\times$): Allows us to combine two relations.

- **Intersection** ($\cap$): Tuples in r1 and in r2. *Next*
- **Joins** ($\bowtie_\theta$, $\bowtie$): Combine relations that satisfy predicates

Compound Operator: Intersection

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

S1 ∩ S2



Intersection (∩)

- Same story as — with respect to duplicates
  - Duplicates don't need to be eliminated if inputs are sets

Relational *Instance* **S1**

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

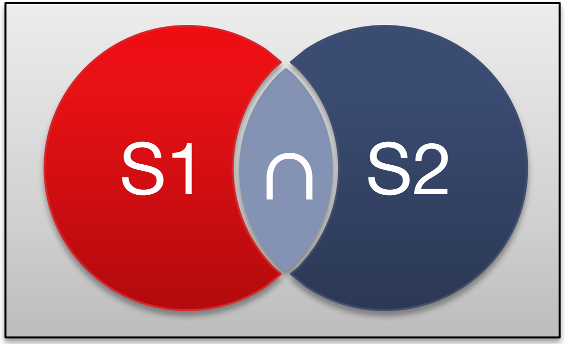
Relational *Instance* **S2**

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

S1 ∩ S2

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

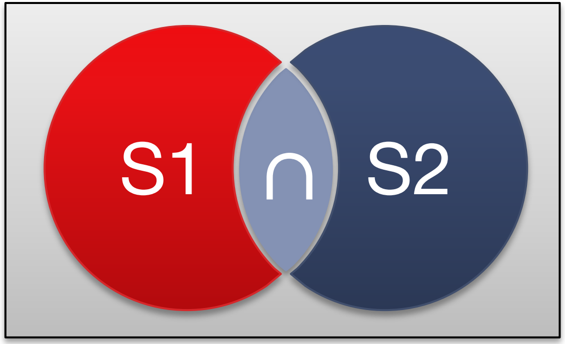
Intersection (∩), Pt 2

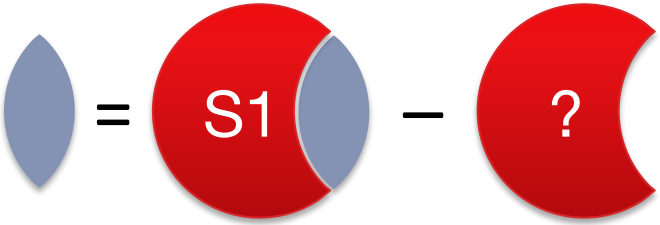


- We saw that ∩ is a compound operator.
- S1 ∩ S2 = ?

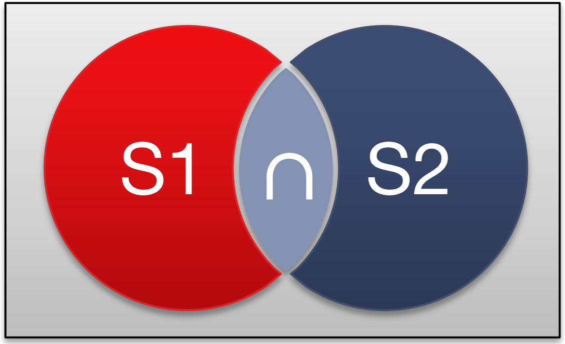
Intersection (∩), Pt 3

- S1 ∩ S2 = S1 – ?
- Q: What is "?"

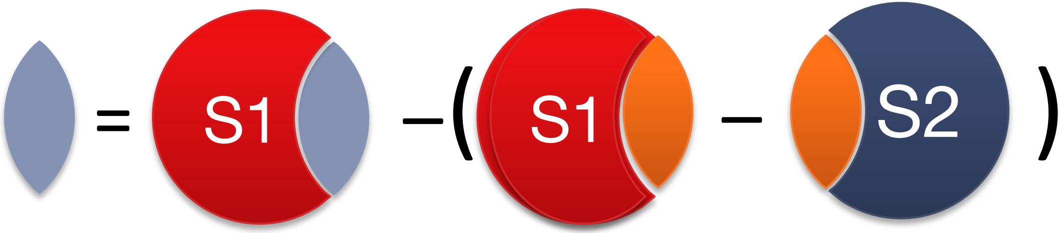




Intersection (∩), Pt 4



- S1 ∩ S2 = S1 – (S1 – S2)



Compound Operator: Join

- Joins are compound operators (like intersection):
  - Generally, σ_θ(R × S)
  - With possibly a rename in there (for natural join)

- Increasing degree of specialization
  - **Theta Join** ($\bowtie_\theta$): join on logical expression θ
  - **Equi-Join:** theta join with theta being a conjunction of equalities
  - **Natural Join** ($\bowtie$): equi-join on all matching column names
    - (note: only one copy per column preserved!)
- Relating information across tables using joins/cross-products is super useful and important
  - Want to avoid cross-products
  - We’ll learn efficient join algorithms

Theta Join ($\bowtie_\theta$) Semantics

- R $\bowtie_\theta$ S = σ_θ(R × S)
- **Apply a cross-product, then filter out tuples that don't match.**
- If θ only contains equality conditions (with an AND between them), this is called an *equi-join*

Theta Join ($\bowtie_\theta$) Example

- **Say we want to find boats that people have reserved**
- **R1** $\bowtie_{sid=sid}$ **S1**
  - Confusing… hard to interpret!

R1:

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

$\bowtie_{sid=sid}$

S1:

sid	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
58	103	11/12/96	58	rusty	10	35.0

  - *Q: How do we fix?*

Theta Join ($\bowtie_\theta$) Example

- $\rho_{(sid\text{->}sid1)}$**R1** $\bowtie_{sid1=sid}$ **S1**

R1:

sid1	bid	day
22	101	10/10/96
58	103	11/12/96

$\bowtie_{sid1=sid}$

S1:

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

=

sid1	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

Another Theta Join ($\bowtie_\theta$) Self Join Example

- $\mathbf{R \bowtie_\theta S = \sigma_\theta(R \times S)}$
- **Example:** *More senior sailors for each sailor.*
- $\rho_{\text{(sid1, sname1, rating1, age1)}}$ S1 $\bowtie_{\text{age1 < age}}$ S1

S1				S1			
sid1	sname1	rating1	age1	sid	sname	rating	age
~~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~~

S1:

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

Natural Join (⋈)

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

- 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 (⋈) Pt 2

- **R ⋈ S**

R1 ⋈ 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:

<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

Natural Join (⋈), Pt 3

- **R ⋈ S**

R1 ⋈ 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

S1:

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).

Natural Join (⋈), Pt 3

- **R ⋈ S**

R1 ⋈ 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

S1:

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

- **Q: How do we express R1 ⋈ S1 using the other operators?**
- **R1 ⋈ S1 =** $\pi_{\text{sid, bid, day, sname, rating, age}}\ \sigma_{\text{sid = sid1}}(\ \mathbf{R1} \times \rho_{\text{(sid->sid1)}}\ \mathbf{S1}\ )$
- **R1 ⋈ S1 =** $\pi_{\text{unique fld.}}\ \sigma_{\text{eq. matching fld.}}(\ \mathbf{R1} \times \rho_{\text{eq. matching fld. renamed}}\ \mathbf{S1}\ )$

Other Natural Join Variants

- We have convenient symbols for Outer joins:
    - Left Outer join
        - R ⟕ S
    - Right Outer join
        - R ⟖ S
    - Full Outer join
        - R ⟗ S

Complex Relational Algebra Expressions

- Algebras allow us to express sequences of operations in a natural way.
- Example
  - in arithmetic algebra: $(x + 4)*(y - 3)$
- Relational algebra allows the same.
- Three notations:
  1. Sequences of assignment statements.
  2. Expressions with several operators.
  3. Expression trees.

Sequences of Assignments

- Create temporary relation names.
- Renaming can be implied by giving relations a list of attributes.
  - R3(X, Y) := R1
- Example: R3 := R1 $\bowtie_C$ R2 can be written:

  R4 := R1 x R2

  R3 := σ_C (R4)

Expressions with Several Operators

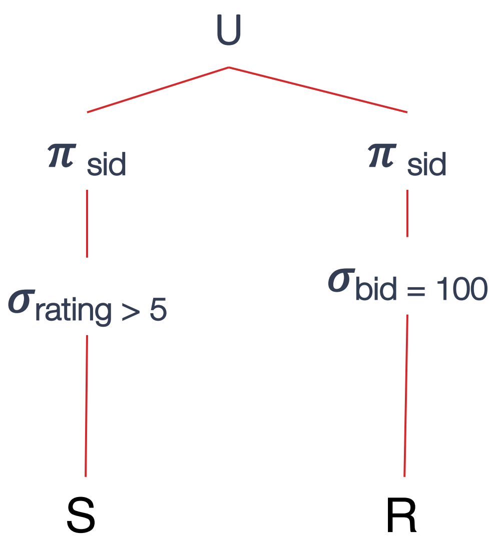
Precedence of relational operators:

1. Unary operators --- select, project, rename --- have highest precedence, bind first.
2. Then come products and joins.
3. Then set operations bind last.

But you can always insert parentheses to force the order you desire.

Expression Trees

- Leaves are operands (relations).
- Interior nodes are operators, applied to their child or children.
- Given R(sid, bid, day), S(sid, sname, rating, age), find the sids of all the sailors whose rating>5 or have reserved boat 100.



A Step Back: Why Did We Study This?

- Relational algebra expressions, just like linear algebra or elementary algebra expressions are easy to manipulate for the DBMS
- Also the number of operators is small so it’s easy to work with.
- To figure out how to rewrite and simplify rel alg expressions, the DBMS uses:
    - Various heuristics
    - Various cost functions

Simple Rewritings

- Example: Changing the order of predicate evaluation
  - $\sigma_{exp1 \wedge exp\ 2}$ **R** = σ_{exp1} **(**σ_{exp2} **R)** = σ_{exp2} **(**σ_{exp1} **R)**

- Example: Changing the order of joins
  - $(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$

An Example of a "Rewrite": Push-Down

- Want reservations for sailors whose age > 40

$\sigma_{age > 40}$ (R1 ⋈ 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

S1:

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

Q: Any other expressions?

Another equiv. exp: R1 ⋈ $\sigma_{age > 40}$ S1

➔ This may be cheaper to compute!

An Example of a “Rewrite”: Eliminating Nesting

- *Names of sailors who’ve **not** reserved boat #103:*

```
SELECT  S.sname
FROM  Sailors S
WHERE  S.sid NOT IN
        (SELECT  R.sid
        FROM  Reserves R
        WHERE  R.bid=103)
```

R:

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

S:

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

An Example of a “Rewrite”: Eliminating Nesting

- *Names of sailors who’ve **not** reserved boat #103:*

One approach:

$$\pi_{sname}\, \mathbf{R} - \pi_{sname}\, ((\sigma_{bid=103}\, \mathbf{R}) \bowtie S))$$

R:

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

S:

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

Extended Relational Algebra

- Group By / Aggregation Operator (γ):
  - $\gamma_{\text{age, AVG(rating)}}$(Sailors)
  - With selection (HAVING clause):
    - $\gamma_{\text{age, AVG(rating), COUNT(*)>2}}$(Sailors)

- Implicitly combines GROUP BY, HAVING and SELECT

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.
  - Easy to manipulate/rewrite/simplify
  - Super powerful! Can encapsulate a lot of SQL functionality
- Basic ops include: σ, π, $\times$, $\cup$, $-$
- Important compound ops: $\cap$, $\bowtie$