Relational Algebra and SQL

Slides by:

Joe Hellerstein

hellerstein@berkeley.edu

A bit of computing history

- Pre-1969: databases were more like data *structures*
  - i.e. hackery

- 1969: E.F. Codd's relational model and languages
  - A mathematical abstraction, independent of data structures

1. Mathematical relations with typed attributes
2. A *Relational Algebra* of simple operations on relations
   - In the spirit of abstract algebra (groups, rings, fields, etc)
   - Inspired functional libraries like **Pandas**
3. A *Relational Calculus* of truth expressions over relations
   - Inspired declarative languages like **SQL**, Datalog

Historical Perspective

- 1969: Codd’s Theorem
- 1974: IBM System R and Berkeley Ingres research projects begin
- 1979: Oracle released first commercial SQL system for DEC Vax minicomputer
- 1981: Ted Codd receives Turing Award
- 1983: IBM DB2 released for MVS mainframe
- 1984-87: Teradata, Informix SQL and Sybase released
- 1988: Berkeley Postgres project begins
- 1989: Microsoft SQL Server released (derived from Sybase)
- 1992: First meaningful SQL standard
- 1995: PostgreSQL released (“Postgres 95”), MySQL released
- 2000: Sqlite released
- 2004: Google MapReduce paper
- 2010: Apache Hive (SQL on Hadoop) released
- 2012: Pandas library popularized

Math Review: Relations

Consider two *domains* D_1, D_2

Can define a finite ***set*** $S \subset D_1$

E.g.

$D_1 = \mathbb{R}$, $D_2 = \mathbb{Z}$

$S = \{4.2, 3.6\}$

Math Review: Relations

Consider two *domains* D_1, D_2

Can define a finite ***set*** $S \subset D_1$

Consider the domain $D_1 \times D_2$

Can define a finite ***relation*** $R \subset D_1 \times D_2$

Each element of R is a ***tuple***

E.g.

$D_1 = \mathbb{R}$, $D_2 = \mathbb{Z}$

$S = \{4.2, 3.6\}$

$\mathbb{R} \times \mathbb{Z}$

$R = \{(4.2, 6), (3.6, 6), (4.2, 1)\}$

Math Review: Relations

Consider two *domains* D_1, D_2

Can define a finite ***set*** $S \subset D_1$

Consider the domain $D_1 \times D_2$

Can define a finite ***relation*** $R \subset D_1 \times D_2$

Each element of R is a ***tuple***

A ***function*** $F \subset D_1 \times D_2$ is a relation such that
$((x, y) \in F \land (x, z) \in F) \Rightarrow y = z$

We can say that the value in the second position is *functionally dependent* on the value in the first.

E.g.

$D_1 = \mathbb{R}$, $D_2 = \mathbb{Z}$

$S = \{4.2, 3.6\}$

$\mathbb{R} \times \mathbb{Z}$

$R = \{(4.2, 6), (3.6, 6), (4.2, 1)\}$

$F = \{(4.2, 6), (3.6, 6)\}$

Math Review: Relations

Consider two *domains* D_1, D_2

Can define a finite ***set*** $S \subset D_1$

Consider the domain $D_1 \times D_2$

Can define a finite ***relation*** $R \subset D_1 \times D_2$

Each element of R is a ***tuple***

A ***function*** $F \subset D_1 \times D_2$ is a relation such that
$((x, y) \in F \wedge (x, z) \in F) \Rightarrow y = z$

We can say that the value in the second position is *functionally dependent* on the value in the first.

Consider a relation $R2 \subset D_1 \times D_2 \times D_3 \times D4$

E.g.

$D_1 = \mathbb{R}$, $D_2 = \mathbb{Z}$

S = {4.2, 3.6}

$\mathbb{R} \times \mathbb{Z}$

R = {(4.2, 6), (3.6, 6), (4.2, 1)}

F = {(4.2, 6), (3.6, 6)}

R2 = {(4.2, 6, red, 😸),
(3.6, 6, blue, 🙀)

Isn't this just Vectors and Matrices?

A finite set *could* be encoded as an (infinite, sparse) boolean vector over the domain values

Consider the domain $D_1 \times D_2$

A finite relation *could* be encoded as an (infinite, sparse) boolean matrix over the values of $D_1 \times D_2$

E.g.

$D = \mathbb{J}$

S = (0,0,0,0,1,0,0,0,0,0,0,1,…)

R = ((0,0,0,0,1,0,0,0,0,0,0,1,…),
(0,0,0,0,0,0,0,0,0,0,0,0,…),
….
)

Concerns?

Possible concerns

- Matrix/vector notation won't work nicely with continuous domains like $\mathbb{R}$
- Linear algebra may not provide the operations we want in a natural way.
  - E.g. union, intersection, predicates…
- Notation could become unwieldy
  - Finite Sets/Relations are typically *sparse*
  - End up representing non-zero entries as tuples anyhow!

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

Keep a lot of tools in your toolbox

Relational Terminology

- *Database*: Set of Relations
- *Relation* (*Table*):
  - *Schema (metadata)*
    - A unique name for the relation
    - A list of *k* distinct Attribute names, each associated with a type.
    - Optional *constraints* (*key constraints*)
  - *Instance (data)*
    - Set of *k*-tuples satisfying the schema
- *Attribute* (*Column, Field*)
- *Tuple* (*Row, Record*)

The schema of a database is the set of schemas of its relations.

Boat Club Schema

sailors(<u>sid integer</u>, sname text, rating integer, age float)

boats(<u>bid integer</u>, bname text, color text)

reserves(<u>sid</u> integer, <u>bid</u> integer, <u>day</u> date)

Boat Club

Example Instances

R1

<u>sid</u>	<u>bid</u>	<u>day</u>
22	101	10/10/16
58	103	11/12/96

Boats

<u>bid</u>	bname	color
101	Interlake	blue
102	Interlake	red
104	Marine	red
103	Clipper	green

S1

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

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

Note: primary keys <u>underlined</u>

Why learn Relational Algebra

- Intuitive for programmers
  - Imperative: apply this, then apply that
  - Set-oriented: no need for for-loops, low-level reasoning
- Basis of functional libraries like Pandas
  - Pandas (over-?) complicates things
  - Nice to have a clean foundation
- Common currency
  - Most data folk know the relational algebra operators

Relational Algebra Preliminaries

- Algebra of *operators* on *relational instances*

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

  - **Closed**: result is also a relational instance
    - Enables rich composition!
  - **Typed**: input schema and operator determines output
    - Why is this important?

- Pure relational algebra has ***set* semantics**
  - **No duplicate** tuples in a relation instance
  - vs. SQL, which has *multiset* (bag) semantics

Relational Algebra Operators

Unary Operators: *operate on **single** relation instance*

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

Binary Operators: *operate on **pairs** of relation instances*

- **Union ($\cup$):** *Tuples in r1 or in r2.*
- **Intersection ($\cap$):** *Tuples in r1 and in r2.*
- **Set-difference (—):** *Tuples in r1, but not in r2.*
- **Cross-product ($\times$):** *Allows us to combine two relations.*
- **Joins ($\bowtie_\theta$, $\bowtie$):** *Combine relations that satisfy predicates*

Projection (π)

Selects a subset of columns (vertical)

$$\pi_{\underbrace{sname,rating}_{\text{List of Attributes}}}(S2)$$

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

- Schema determined by schema of attribute list
  - Names and types correspond to input attributes

Projection (π)

Selects a subset of columns (vertical)

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

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

Multiset

age
35.0
55.5
35.0
35.0

Set

age
35.0
55.5

- Set semantics → results in fewer rows
  - SQL systems don't automatically remove duplicates
  - Why?

Selection(σ)

Selects a subset of rows (horizontal)

$$\sigma_{\underbrace{rating>8}_{\text{Selection Condition (Boolean Expression)}}}(S2)$$

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

- Output schema same as input
- Duplicate Elimination?

Composing Select and Project

- Names of sailors with rating > 8

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

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

$\sigma_{rating>8}$

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

π_{name}

sname
yuppy
rusty

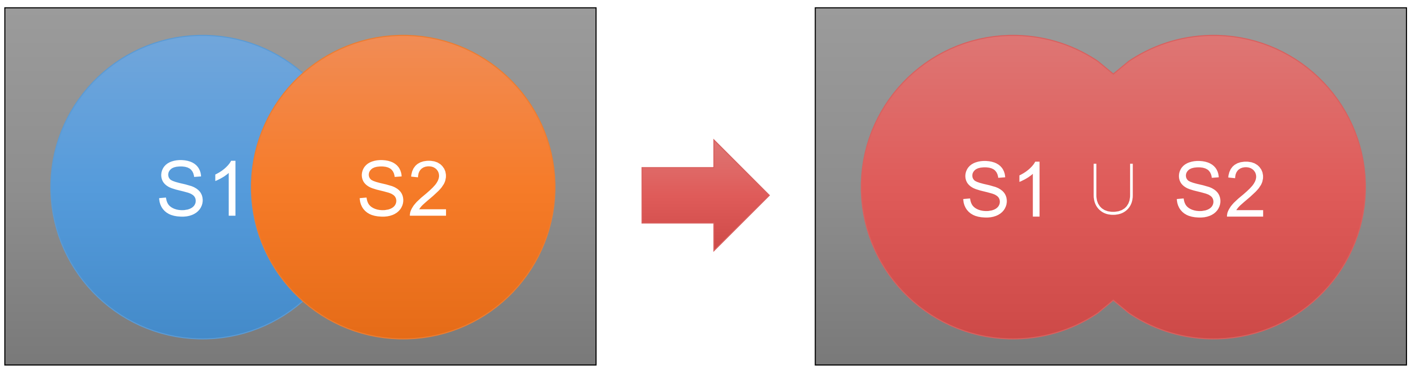
- What about:

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

Invalid types. Input to $\sigma_{rating>8}$ does not contain rating.

Union (∪)

S1 ∪ **S2**



Two input relations, must be *compatible*:

- Same number of fields.
- Fields in the same position have same **type**

Union (∪)

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

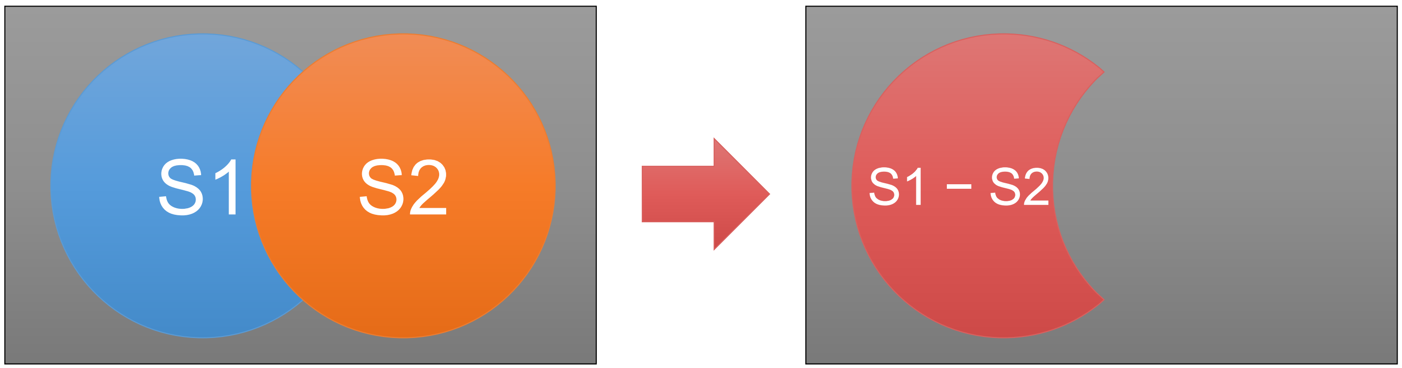
S1 ∪ S2

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

Duplicate elimination?

Set Difference (−)

S1 − S2



Same as with union, both input relations must be *compatible*.

Set Difference (–)

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

Symmetric?

S2 – S1

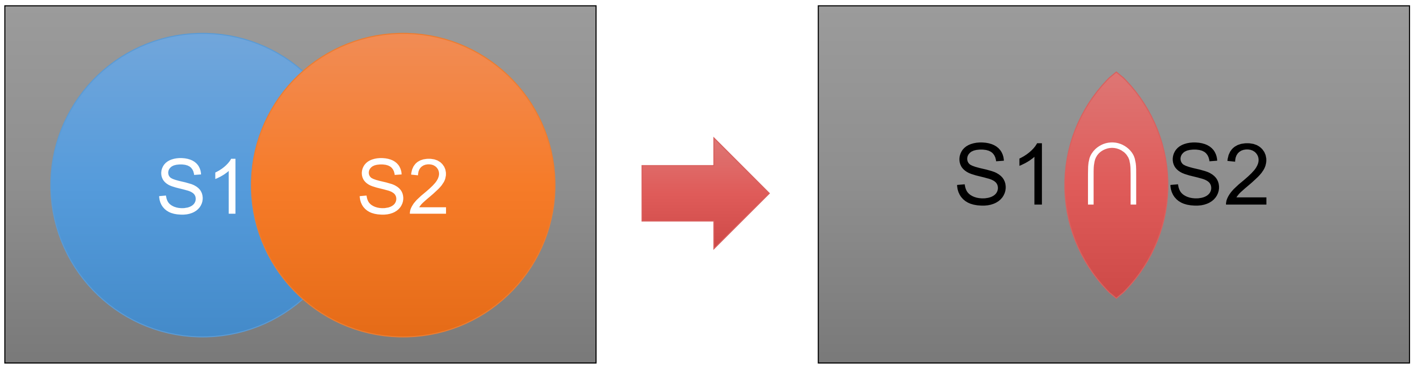
<u>sid</u>	sname	rating	age
28	yuppy	9	35.0
44	guppy	5	35.0

Duplicate elimination?

- Not required

Intersection (∩)

S1 ∩ S2



Same as with union, both input relations must be *compatible*.

Intersection (⋂)

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

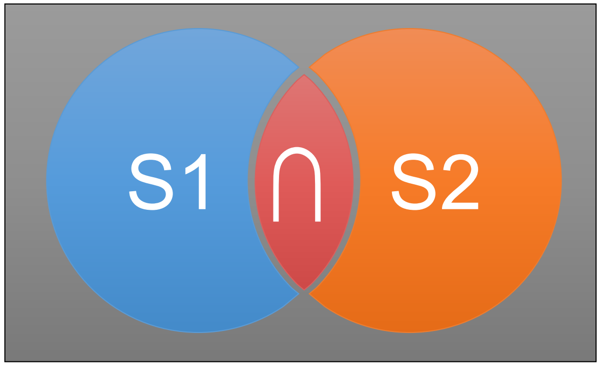
S1 ⋂ S2

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

Is intersection essential?

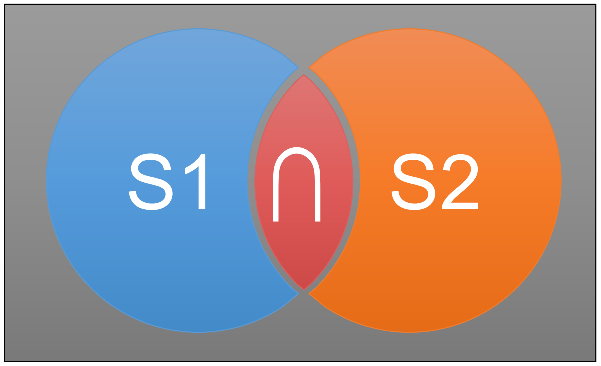
- Implement it with earlier ops. ?

Intersection (∩)

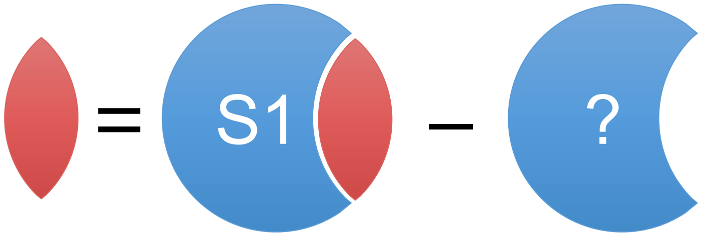


$$S1 \cap S2 = S1 - ?$$

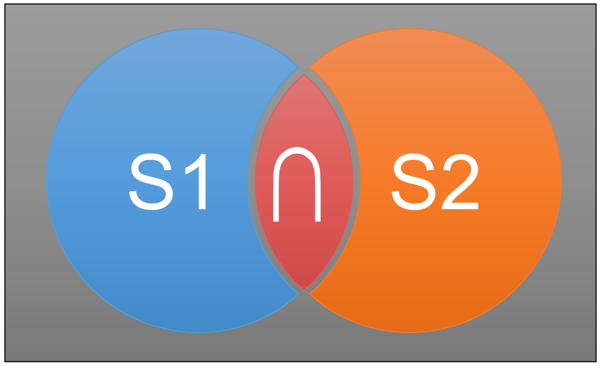
Intersection (∩)



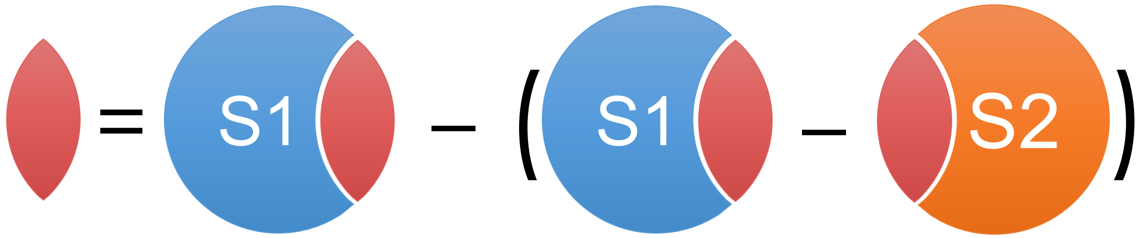
$$S1 \cap S2 = S1 - ?$$



Intersection (∩)



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



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:**

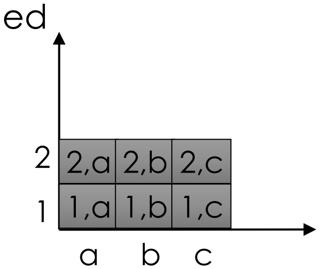
sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

=

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

Sometimes also called
Cartesian Product:



How many rows in the result? | R1 | * | R2 |

Schema compatibility? No requirements.

One field per field in original schemas.

What about duplicate names? Renaming operator

Renaming (ρ = "*rho*")

Renames relations and their attributes:

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

R1 × 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 can also be defined *positionally*, without names.
- Difficult to read …

$$\pi_{f5}(\sigma_{f6>f8}(S2))$$

Compound Operator: Join

- Joins are compound operators (like intersection):
  - **Cross product** followed by **selection** and possibly **projection** (for natural join)
- Hierarchy of common kinds:
  - **Theta Join ($\bowtie_\theta$):** *join on logical expression* θ
    - **Equi-Join:** *theta join with conjunction equalities*
      - **Natural Join ($\bowtie$):** *equi-join on all matching column names*
- Note: we should use a join, not a cross-product, if we can! Easier to read, clarifies opportunities for using efficient join algorithms.

Theta Join ($\bowtie_\theta$)

$$\mathbf{R \bowtie_\theta S = \sigma_\theta(R \times S)}$$

Example: *Pair each sailor with older sailors.*

$$S1 \bowtie_{age < age} S1$$

S1:

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

Theta Join ($\bowtie_\theta$)

$$\mathbf{R} \bowtie_\theta \mathbf{S} = \sigma_\theta(\mathbf{R} \times \mathbf{S})$$

Example: *Pair each sailor with older sailors.*

$$S1 \bowtie_{age < age} S1$$

S1:

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

S1 × S1

S1				S1			
sid	sname	rating	age	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~~

Theta Join ($\bowtie_\theta$)

$$\mathbf{R} \bowtie_\theta \mathbf{S} = \sigma_\theta(\mathbf{R} \times \mathbf{S})$$

Example: *Pair each sailor with older sailors.*

$$S1 \bowtie_{age < age} S1$$

S1:

sid	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	age
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	58	rusty	10	35.0

- *Result schema* same as that of cross-product.
- Special Case:
  - **Equi-Join:** *theta join with conjunction equalities*
    - *Special special case* ***Natural Join*** …

Natural Join (⋈)

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

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

- Compute **R × S**
- **Select** rows where attributes appearing in both relations have equal values
- **Project** onto the set of all unique attributes.

Natural Join (⋈)

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

Example:

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

R1 ⋈ S1

$\boxed{\text{sid}}$	bid	day	$\boxed{\text{sid}}$	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

Natural Join ($\bowtie$)

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

Example:

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

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

Commonly used for foreign key joins (as above).

Exercise:

Find names of sailors who've reserved boat #103

Boats(<u>bid</u>,bname,color)
Sailors(<u>sid</u>, sname, rating, age)
Reserves(<u>sid</u>, <u>bid</u>, <u>day</u>)

- Solution 1:

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

- Solution 2:

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

Exercise:

Find names of sailors who've reserved a red boat

Boats(<u>bid</u>,bname,color)
Sailors(<u>sid</u>, sname, rating, age)
Reserves(<u>sid</u>, <u>bid</u>, <u>day</u>)

- Solution 1:

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

- More "efficient" Solution 2:

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

In general many possible equivalent expressions: **algebra**…

Relational Algebra Rules

- **Selections:**
  - $\sigma_{c1\wedge\ldots\wedge cn}(R) \equiv \sigma_{c1}(\ldots(\sigma_{cn}(R))\ldots)$ *(cascade)*
  - $\sigma_{c1}(\sigma_{c2}(R)) \equiv \sigma_{c2}(\sigma_{c1}(R))$ *(commute)*
- **Projections:**
  - $\pi_{a1}(R) \equiv \pi_{a1}(\ldots(\pi_{a1,\ \ldots,\ an-1}(R))\ldots)$ *(cascade)*
- **Cartesian Product**
  - $R \times (S \times T) \equiv (R \times S) \times T$ (associative)
  - $R \times S \equiv S \times R$ (commutative)
  - *Applies for joins as well but* be careful with join predicates …

Caution with Join Ordering

Boats(<u>bid</u>,bname,color)
Sailors(<u>sid</u>, sname, rating, age)
Reserves(<u>sid</u>, <u>bid</u>, <u>day</u>)

- Consider the following:

$$\text{Boats} \bowtie_{bid} \text{Reserves} \bowtie_{sid} \text{Sailors}$$

- Commute and Associate:

$$\text{Boats} \bowtie_{bid} \left(\text{Sailors} \bowtie_{sid} \text{Reserves} \right)$$

  - Incompatible join predicate:

$$\left(\text{Boats} \bowtie_{bid} \text{Sailors} \right) \bowtie_{sid} \text{Reserves}$$

Caution with Join Ordering

Boats(<u>bid</u>,bname,color)
Sailors(<u>sid</u>, sname, rating, age)
Reserves(<u>sid</u>, <u>bid</u>, <u>day</u>)

- Consider the following:

$$\text{Boats} \bowtie_{bid} \text{Reserves} \bowtie_{sid} \text{Sailors}$$

- Commute and Associate:

$$\text{Boats} \bowtie_{bid} \left(\text{Sailors} \bowtie_{sid} \text{Reserves}\right)$$

  - Incompatible join predicate:

$$\left(\text{Boats} \times \text{Sailors}\right) \bowtie_{sid,bid} \text{Reserves}$$

More Relational Algebra Rules

Commuting of selection operators

- $\sigma_c(R \times S) \equiv \sigma_c(R) \times S$ (c only has fields in R)
- $\sigma_c(R \bowtie S) \equiv \sigma_c(R) \bowtie S$ (c only has fields in R)

Commuting of projection operators

- $\pi_a(R \times S) \equiv \pi_{a1}(R) \times \pi_{a2}(S)$
  - a_1 is subset of a that mentions R and a_2 is subset of a that mentions S
  - Similar result holds for joins

A Standard Extension

- Group By / Aggregation Operator (γ):

$$\gamma_{age,\ AVG(rating)}(Sailors)$$

  - *With selection (HAVING clause):*

$$\gamma_{age,\ AVG(rating),\ COUNT(*)>2}(Sailors)$$

Recall Codd also had a Relational Calculus

- A *declarative* logic language
  - Find all tuples such that the following properties hold …
  - Says "what" the output should be, not "how" to get it.
- SQL is based on the relational calculus
  - Even though, under the hood, database engines translate to algebra expressions!

SQL Language

- Two sublanguages:
  - DDL – Data Definition Language
    - Define and modify schema
  - DML – Data Manipulation Language
    - Queries can be written intuitively.

- Relational Database Management System (RDBMS) responsible for efficient evaluation.
  - Choose and run algorithms for declarative queries

We will learn SQL interactively

- Frontend: psql command line, Jupyter Notebook
- Backend: PostgreSQL