CS 450

Relational Algebra 3

Prof. Jessica Lin

More Examples on Sailors Relations

Sailors(sid, sname, rating, age)

Boats(bid, bname, color)

Reserves(sid, bid, day)

Find names of sailors who've reserved boat #103

- Solution 1: Find those who reserved boat 103, join with Sailors to find the names, and project out the names

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

- Solution 2: Join Reserves and Sailors to get all information, and find those who reserved boat 103. Project out the names.

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

Which one is more efficient?

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: Find the bids of red boats first before doing the join.

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

☛ *A query optimizer can find this given the first solution!*

Find sailors who've reserved a red <u>or</u> a green boat

- Can identify all red or green boats, then find sailors who've reserved one of these boats:

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:

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

$$\text{Re}sult = \pi_{sname}(Tempboats \bowtie \text{Re}serves \bowtie Sailors)$$

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:

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

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

- Can also define Tempboats using union:

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

- What happens if "or" is replaced by "and"?

Find sailors who've reserved a red <u>and</u> a green boat

- Previous first approach won't work! (Why not?) Must use intersection.

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

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

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

Consider yet another query

- Find the sailor(s) who reserved all the red boats.

R1

sid	*bid*	*day*
22	101	10/10/96
22	103	10/11/96
56	102	11/12/96

B

bid	*color*
101	Red
102	Green
103	Red

Start an attempt

- Who reserved what boat:

$$S1 = \pi_{sid,bid}(R1) =$$

sid	*bid*
22	101
22	103
56	102

- All the red boats:

$$S2 = \pi_{bid}(\sigma_{color=red}(B)) =$$

bid
101
103

Now what?

Find the sailor(s) who reserved all the red boats.

- We will solve the problem the "hard" way, and then will introduce an operator specifically for this kind of problem.
- *Idea*: Compute the sids of sailors who *didn't* reserve all red boats.
  1. Find all possible reservations that could be made on red boats.
  2. Find *actual* reservations on red boats
  3. Find the possible reservations on red boats that were not actually made (#1 – #2) (set difference)
  4. Project out the sids from 3 – these are the sailors who didn't have reservation on some red boat(s).

Find the sailor(s) who reserved all the red boats.

- *Idea*: Compute the sids of sailors who *didn't* reserve all red boats (then find the difference between this set and set of all sailors).
  1. Find all possible reservations that could be made on red boats.

     $AllPossibleRes = \pi_{sid} (R1) \times \pi_{bid} \sigma_{color=\text{“red”}} (B)$
  2. Find *actual* reservations on red boats

     $AllRedRes = \pi_{sid,bid} (R1) \bowtie \pi_{bid} \sigma_{color=\text{“red”}} (B)$

  3. 4. Find the possible reservations on red boats that were not actually made, and project out the sids.

     $\pi_{sid} (AllPossibleRes - AllRedRes)$
  5. Find sids that are not in the result from above (sailors such that there is no red boat that's not reserved by him/her)

     $\pi_{sid} (R1) - \pi_{sid} (AllPossibleRes - AllRedRes)$

Division Operation

$r\ /\ s$

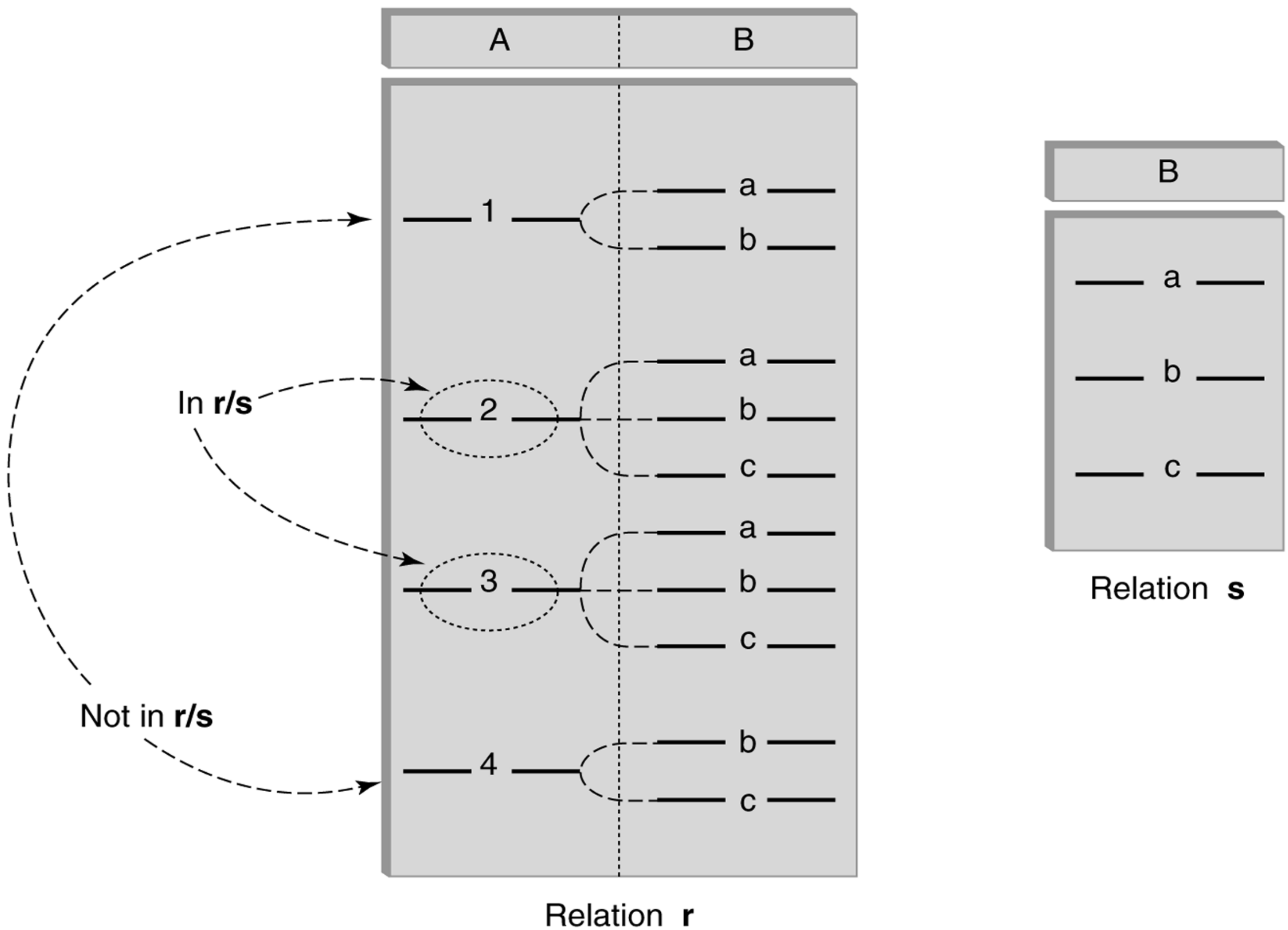
- Suited to queries that include the phrase "for all", e.g. *Find sailors who have reserved **all** red boats*.
- Produce the tuples in one relation, r, that match *all* tuples in another relation, s
- Let *S1* have 2 fields, *x* and *y*; *S2* have only field *y*:
  - *S1/S2* = $\left\{ \langle x \rangle \mid \forall \ \langle y \rangle \ in \ S2 \ (\exists \ \langle x,y \rangle \ in \ S1) \ \right\}$
  - i.e., ***S1/S2*** **contains all *x* tuples (sailors) such that for *every* *y* tuple (redboat) in *S2*, there is an *xy* tuple in *S1* (i.e, x reserved y).**

- In general, *x* and *y* can be any lists of fields; *y* is the list of fields in *S2*, and $x \cup y$ is the list of fields of *S1*.
- Let r and s be relations on schemas R and S respectively where
  - $R = (A_1, \ldots, A_m, B_1, \ldots, B_n)$,
  - $S = (B_1, \ldots, B_n)$,

  The result of $r\ /\ s$ is a relation on schema

  $R - S = (A_1, \ldots, A_m)$

Division (cont'd)



Division Operation – Example

Relations *r, s*:

A	**B**
α	*1*
α	*2*
α	*3*
β	*1*
γ	*1*
δ	*1*
δ	*3*
δ	*4*
$\in$	*6*
$\in$	*1*
β	*2*

r

B
1
2

s

r / s:

A
α
β

α occurs in the presence of both ***1*** and ***2***, so it is returned.
β occurs in the presence of both ***1*** and ***2***, so it is returned.
γ does not occur in the presence of both ***1*** and ***2***, so is ignored.
...

Another Division Example

Relations *r, s*:

A	*B*	*C*	*D*	*E*
α	a	α	a	*1*
α	a	γ	**a**	**1**
α	a	γ	**b**	**1**
β	a	γ	a	*1*
β	a	γ	b	*3*
γ	a	γ	**a**	**1**
γ	a	γ	**b**	**1**
γ	a	β	b	*1*

r

D	*E*
a	**1**
b	**1**

s

r / *s*:

A	*B*	*C*
α	a	γ
γ	a	γ

<α, a ,γ > occurs in the presence of both <***a,1***> and <***b,1***>, so it is returned.
< γ, a ,γ > occurs in the presence of both <***a,1***> and <***b,1***>, so it is returned.
<β, a ,γ > does not occur in the presence of both <***a,1***> and <***b,1***>, so it is ignored.

More Division Examples: A/B

A

sno	pno
s1	p1
s1	p2
s1	p3
s1	p4
s2	p1
s2	p2
s3	p2
s4	p2
s4	p4

B1

pno
p2

B2

pno
p2
p4

B3

pno
p1
p2
p4

A/B1

sno
s1
s2
s3
s4

A/B2

sno
s1
s4

A/B3

sno
s1

Find the sailor(s) who reserved ALL red boats

- who reserved what boat:

$$S1=\pi_{sid,bid}(R1)=$$

sid	bid
22	101
22	103
58	102

- All the red boats:

$$S2=\pi_{bid}(\sigma_{color=red}(B))=$$

bid
101
103

=> S1/S2

Find the names of sailors who' ve reserved all boats

- Uses division; schemas of the input relations to "divide" must be carefully chosen:

$$\text{Tempsids} = (\pi_{sid,bid}(\text{Re}\mathit{serves})) / (\pi_{bid}(\mathit{Boats}))$$

$$\text{Result} = \pi_{sname}(\mathit{Tempsids} \bowtie \mathit{Sailors})$$

- SALES(supId, prodId);
- PRODUCTS(prodId);
- SALES/PRODUCTS = ?

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. Division is NOT implemented in SQL).
- *Idea*: For *SALES/PRODUCTS*, compute the IDs of suppliers that don't supply all products.

$$A = \pi_{sid}((\pi_{sid}(Sales) \times Products) - Sales)$$

The answer is $\pi_{sid}(\text{Sales}) - A$

Additional Operator: Outer Join

- An extension of the join operation that avoids loss of information.
- Computes the join and then adds tuples from one relation that does not match tuples in the other relation to the result of the join.
- Uses *null* values:
  - *null* signifies that the value is unknown or does not exist
  - All comparisons involving *null* are (roughly speaking) **false** by definition.
    - Will study precise meaning of comparisons with nulls later

Outer Join – Example

Relation *loan*

loan-number	*branch-name*	*amount*
L-170	Springfield	3000
L-230	Shelbyville	4000
L-260	Dublin	1700

Relation *borrower*

customer-name	*loan-number*
Simpson	**L-170**
Wiggum	**L-230**
Flanders	L-155

Outer Join – Example

- **Inner Join**

loan ⋈ *Borrower*

loan-number	***branch-name***	***amount***	***customer-name***
L-170	Springfield	3000	Simpson
L-230	Shelbyville	4000	Wiggum

loan-number	***branch-name***	***amount***
L-170	Springfield	3000
L-230	Shelbyville	4000
L-260	Dublin	1700

customer-name	***loan-number***
Simpson	**L-170**
Wiggum	**L-230**
Flanders	L-155

- **Left Outer Join**

loan ⟕ *borrower*

loan-number	***branch-name***	***amount***	***customer-name***
L-170	Springfield	3000	Simpson
L-230	Shelbyville	4000	Wiggum
L-260	Dublin	1700	*null*

Outer Join – Example

Right Outer Join

loan-number	*branch-name*	*amount*	*customer-name*
L-170	Springfield	3000	Simpson
L-230	Shelbyville	4000	Wiggum
L-155	*null*	*null*	Flanders

loan ⟖ *borrower*

loan-number	*branch-name*	*amount*
L-170	Springfield	3000
L-230	Shelbyville	4000
L-260	Dublin	1700

customer-name	*loan-number*
Simpson	**L-170**
Wiggum	**L-230**
Flanders	L-155

Full Outer Join

loan ⟗ *borrower*

loan-number	*branch-name*	*amount*	*customer-name*
L-170	Springfield	3000	Simpson
L-230	Shelbyville	4000	Wiggum
L-260	Dublin	1700	*null*
L-155	*null*	*null*	Flanders

Null Values

- It is possible for tuples to have a null value, denoted by *null*, for some of their attributes
- *null* signifies an unknown value or that a value does not exist.
- The result of any arithmetic expression involving *null* is *null.*
- Aggregate functions simply ignore null values
  - Is an arbitrary decision. Could have returned null as result instead.
  - We follow the semantics of SQL in its handling of null values
- For duplicate elimination and grouping, null is treated like any other value, and two nulls are assumed to be the same
  - Alternative: assume each null is different from each other
  - Both are arbitrary decisions, so we simply follow SQL

Null Values

- Comparisons with null values return the special truth value *unknown*
- Three-valued logic using the truth value *unknown*:
  - OR: (*unknown* **or** *true*) = *true*,
    (*unknown* **or** *false*) = *unknown*
    (*unknown* **or** *unknown*) = *unknown*
  - AND: (*true* **and** *unknown*) = *unknown*,
    (*false* **and** *unknown*) = *false*,
    (*unknown* **and** *unknown*) = *unknown*
  - NOT: (**not** *unknown*) = *unknown*
  - In SQL "*P* **is unknown**" evaluates to true if predicate *P* evaluates to *unknown*
- Result of select predicate is treated as *false* if it evaluates to *unknown*

Summary

- The relational model has rigorously defined query languages that are simple and powerful.
- Relational algebra is more operational; useful as internal representation for query evaluation plans.
- Several ways of expressing a given query; a query optimizer should choose the most efficient version.
- Operations covered: 5 basic operations (selection, projection, union, set difference, cross product), rename, joins (natural join, equi-join, conditional join, outer joins), division