Relational Algebra 2

Week 5

Relational Algebra (So far)

- Basic operations:
  - *Selection* (σ) Selects a subset of rows from relation.
  - *Projection* (π) Deletes unwanted columns from relation.
  - *Cross-product* ($\times$) Allows us to combine two relations.
  - *Set-difference* (-) Tuples in reln. 1, but not in reln. 2.
  - *Union* ($\cup$) Tuples in reln. 1 and tuples in reln. 2.

  Also,
  - *Rename* (ρ) Changes names of the attributes
  - *Intersection* ($\cap$) Tuples in both reln. 1 and in reln. 2.

- Since each operation returns a relation, operations can be *composed*! (Algebra is "closed".)
- Use of temporary relations recommended.

Additional Operations

We define additional operations that do not add any power to the relational algebra, but that simplify common queries.

- Natural join
- Conditional Join
- Equi-Join
- Division

All joins are really special cases of conditional join

Also, we've already seen "Set intersection":

$$r \cap s = r - (r - s)$$

Quick note on notation

good_customers

customer-name	***loan-number***
Patty	1234
Apu	3421
Selma	2342
Ned	4531

bad_customers

customer-name	***loan-number***
Seymour	3432
Marge	3467
Selma	7625
Abraham	3597

If we have two or more relations which feature the same attribute names, we could confuse them. To prevent this we can use dot notation.
For example

good_customers.loan-number

Natural-Join Operation: Motivation

Very often, we have a query and the answer is not contained in a single relation. For example, I might wish to know where Apu banks.
The classic relational algebra way to do such queries is a cross product, followed by a selection which tests for equality on some pair of fields.

borrower

cust-name	*l-number*
Patty	1234
Apu	3421

loan

l-number	*branch*
1234	Dublin
3421	Irvine

$\sigma_{borrower.l\text{-}number\ =\ loan.l\text{-}number}(borrower \times loan)))$

cust-name	*borrower.l-number*	*loan.l-number*	*branch*
Patty	1234	1234	Dublin
Patty	1234	3421	Irvine
Apu	3421	1234	Dublin
Apu	3421	3421	Irvine

cust-name	*borrower.l-number*	*loan.l-number*	*branch*
Patty	1234	1234	Dublin
Apu	3421	3421	Irvine

Note that in this example the two relations are the same size (2 by 2), this does not have to be the case.

While this works…

- it is unintuitive
- it requires a lot of memory
- the notation is cumbersome

So, we have a more intuitive way of achieving the same effect, the natural join, denoted by the $\bowtie$ symbol

Natural-Join Operation: Intuition

Natural join combines a cross product and a selection into one operation. It performs a selection forcing equality on *those attributes that appear in both relation schemes*. Duplicates are removed as in all relation operations.
So, if the relations have one attribute in common, as in the last slide ("*l-number*"), for example, we have…

$$\textit{borrower} \bowtie \textit{loan} \quad = \sigma_{\textit{borrower.l-number} = \textit{loan.l-number}}(\textit{borrower} \times \textit{loan})))$$

There are two special cases:

- If the two relations have no attributes in common, then their natural join is simply their cross product.
- If the two relations have more than one attribute in common, then the natural join selects only the rows where all pairs of matching attributes match. (let's see an example on the next slide).

A

l-name	*f-name*	*age*
Bouvier	Selma	40
Bouvier	Patty	40
Smith	Maggie	2

B

l-name	*f-name*	*ID*
Bouvier	Selma	1232
Smith	Selma	4423

Both the *l-name* and the *f-name* match, so select.

Only the *f-names* match, so don't select.

Only the *l-names* match, so don't select.

l-name	*f-name*	*age*	*l-name*	*f-name*	*ID*
Bouvier	**Selma**	40	**Bouvier**	**Selma**	1232
Bouvier	**Selma**	40	Smith	**Selma**	4423
Bouvier	Patty	2	**Bouvier**	Selma	1232
Bouvier	Patty	40	Smith	Selma	4423
Smith	Maggie	2	Bouvier	Selma	1232
Smith	Maggie	2	**Smith**	Selma	4423

We remove duplicate attributes…

l-name	*f-name*	*age*	*l-name*	*f-name*	*ID*
Bouvier	Selma	40	Bouvier	Selma	1232

The natural join of *A* and *B*

Note that this is just a way to visualize the natural join, we don't really have to do the cross product as in this example

$A \bowtie B =$

l-name	*f-name*	*age*	*ID*
Bouvier	Selma	40	1232

Natural-Join Operation

- Notation: $r \bowtie s$
- Let r and s be relation instances on schemas R and S respectively.The result is a relation on schema $R \cup S$ which is obtained by considering each pair of tuples t_r from r and t_s from s.
- If t_r and t_s have the same value on each of the attributes in $R \cap S$, a tuple t is added to the result, where
  - t has the same value as t_r on r
  - t has the same value as t_s on s
- Example:

  $R = (A, B, C, D)$

  $S = (E, B, D)$

- Result schema = (A, B, C, D, E)
- $r \bowtie s$ is defined as:

$$\pi_{r.A,\ r.B,\ r.C,\ r.D,\ s.E}\ (\sigma_{r.B\ =\ s.B\ \ r.D\ =\ s.D}\ (r \times s))$$

Natural Join Operation – Example

- Relation instances *r*, *s*:

A	*B*	*C*	*D*
α	1	α	a
β	2	γ	a
γ	4	β	b
α	1	γ	a
δ	2	β	b

r

B	*D*	*E*
1	a	α
3	a	β
1	a	γ
2	b	δ
3	b	$\in$

s

$r \bowtie s$

A	*B*	*C*	*D*	*E*
α	1	α	a	α
α	1	α	a	γ
α	1	γ	a	α
α	1	γ	a	γ
δ	2	β	b	δ

How did we get here?

Lets do a trace over the next few slides…

A	B	C	D
α	1	α	a
β	2	γ	a
γ	4	β	b
α	1	γ	a
δ	2	β	b

r

B	D	E
1	a	α
3	a	β
1	a	γ
2	b	δ
3	b	$\in$

s

First we note which attributes the two relations have in common…

A	B	C	D
α	1	α	a
β	2	γ	a
γ	4	β	b
α	1	γ	a
δ	2	β	b

r

B	D	E
1	a	α
3	a	β
1	a	γ
2	b	δ
3	b	∈

s

A	B	C	D	E
α	1	α	a	α
α	1	α	a	γ

There are two rows in s that match our first row in r, (in the relevant attributes) so both are joined to our first row…

A	B	C	D
α	1	α	a
β	2	γ	a
γ	4	β	b
α	1	γ	a
δ	2	β	b

r

B	D	E
1	a	α
3	a	β
1	a	γ
2	b	δ
3	b	∈

s

A	B	C	D	E
α	1	α	a	α
α	1	α	a	γ

…there are no rows in s that match our second row in r, so do nothing…

A	B	C	D
α	1	α	a
β	2	γ	a
γ	4	β	b
α	1	γ	a
δ	2	β	b

r

B	D	E
1	a	α
3	a	β
1	a	γ
2	b	δ
3	b	$\in$

s

A	B	C	D	E
α	1	α	a	α
α	1	α	a	γ

…there are no rows in s that match our third row in r, so do nothing…

A	B	C	D
α	1	α	a
β	2	γ	a
γ	4	β	b
α	1	γ	a
δ	2	β	b

r

B	D	E
1	a	α
3	a	β
1	a	γ
2	b	δ
3	b	∈

s

A	B	C	D	E
α	1	α	a	α
α	1	α	a	γ
α	1	γ	a	α
α	1	γ	a	γ

There are two rows in s that match our fourth row in r, so both are joined to our fourth row…

A	B	C	D
α	1	α	a
β	2	γ	a
γ	4	β	b
α	1	γ	a
δ	2	β	b

r

B	D	E
1	a	α
3	a	β
1	a	γ
2	b	δ
3	b	$\in$

s

A	B	C	D	E
α	1	α	a	α
α	1	α	a	γ
α	1	γ	a	α
α	1	γ	a	γ
δ	2	β	b	δ

There is one row that matches our fifth row in r,.. so it is joined to our fifth row and we are done!

Natural Join on Sailors Example

S1

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

R1

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

$S1 \bowtie R1 =$

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

Earlier We Saw…

Query: Find the name of the sailor who reserved boat 101.

$$Temp = \rho(sid \rightarrow sid1, S1) \times \rho(sid \rightarrow sid2, R1)$$

$$\text{Result} = \pi_{Sname}(\sigma_{sid1 = sid2 \wedge bid = 101}(Temp))$$

* Note my use of "temporary" relation Temp.

Query revisited using natural join

Query: Find the name of the sailor who reserved boat 101.

$$\text{Result} = \pi_{Sname}(\sigma_{bid=101}(S1 \bowtie R1))$$

Or

$$\text{Result} = \pi_{Sname}(S1 \bowtie \sigma_{bid=101}(R1))$$

What' s the difference between these two approaches?

Conditional-Join Operation:

The conditional join is actually the most general type of join. I introduced the natural join first only because it is more intuitive and... natural!

Just like natural join, conditional join combines a cross product and a selection into one operation. However instead of only selecting rows that have equality on those attributes that appear in both relation schemes, we allow selection based on any predicate.

$$r \bowtie_c s = \sigma_c(r \times s)$$

Where *c* is any predicate the attributes of r and/or s

Duplicate rows are removed as always, but duplicate columns are not removed!

Conditional-Join Example:

We want to find all women that are younger than their husbands…

r

l-name	*f-name*	*marr-Lic*	*age*
Simpson	Marge	777	35
Lovejoy	Helen	234	38
Flanders	Maude	555	24
Krabappel	Edna	978	40

s

l-name	*f-name*	*marr-Lic*	*age*
Simpson	Homer	777	36
Lovejoy	Timothy	234	36
Simpson	Bart	*null*	9

$$r \bowtie_{r.age < s.age \text{ AND } r.Marr\text{-}Lic = s.Marr\text{-}Lic} s$$

r.l-name	*r.f-name*	*r.Marr-Lic*	*r.age*	*s.l-name*	*s.f-name*	*s.marr-Lic*	*s.age*
Simpson	Marge	777	35	Simpson	Homer	777	36

Note we have removed ambiguity of attribute names by using "dot" notation
Also note the redundant information in the *marr-lic* attributes

Equi-Join

- Equi-Join: Special case of conditional join where the conditions consist only of equalities.
- Natural Join: Special case of equi-join in which equalities are specified on ALL fields having the same names in both relations.

Equi-Join

r

l-name	*f-name*	*marr-Lic*	*age*
Simpson	Marge	777	35
Lovejoy	Helen	234	38
Flanders	Maude	555	24
Krabappel	Edna	978	40

s

l-name	*f-name*	*marr-Lic*	*age*
Simpson	Homer	777	36
Lovejoy	Timothy	234	36
Simpson	Bart	*null*	9

$r \bowtie_{r.Marr\text{-}Lic \,=\, s.Marr\text{-}Lic} s$

r.l-name	*r.f-name*	*Marr-Lic*	*r.age*	*s.l-name*	*s.f-name*	*s.age*
Simpson	Marge	777	35	Simpson	Homer	36
Lovejoy	Helen	234	38	Lovejoy	Timothy	36

Review on Joins

- All joins combine a cross product and a selection into one operation.
- Conditional Join
  - the selection condition can be of any predicate (e.g. rating1 > rating2)
- Equi-Join:
  - Special case of conditional join where the conditions consist only of equalities.
- Natural Join
  - Special case of equi-join in which equalities are specified on ALL fields having the same names in both relations.

A Note on Precedence

- Unary operators have the highest precedence: [σ, π, ρ]
- Then "multiplicative" operators: [$\times$, $\bowtie$]
- Then "additive" operators: [$\cap$, $\cup$, $-$]

Banking Examples

branch (branch-id, branch-city, assets)

customer (customer-id, customer-name, customer-city)

account (account-number, branch-id, balance)

loan (loan-number, branch-id, amount)

depositor (customer-id, account-number)

borrower (customer-id, loan-number)

Example Queries 1

- Find all loans over $1200

"select from the relation *loan*, only the rows which have a *amount* greater than 1200"

loan

loan-number	***branch-id***	***amount***
1234	001	1,923.03
3421	002	123.00
2342	004	56.25
4531	005	120.03

$\sigma_{amount > 1200}(loan)$

1234	001	1,923.03

Example Queries 2

- Find the loan number for each loan of an amount greater than $1200

"select from the relation *loan*, only the rows which have a *amount* greater than 1200, then project out just the *loan_number*"

loan

loan-number	*branch-id*	*amount*
1234	001	1,923.03
3421	002	123.00
2342	004	56.25
4531	005	120.03

$\sigma_{amount > 1200}\ (loan)$

1234	001	1,923.03

$\pi_{loan\text{-}number}\ (\sigma_{amount > 1200}\ (loan))$

1234

Example Queries 3

- Find all loans greater than $1200 or less than $75

"select from the relation *loan*, only the rows which have a *amount* greater than 1200 or an *amount* less than 75

loan

loan-number	*branch-id*	*amount*
1234	001	1,923.03
3421	002	123.00
2342	004	56.25
4531	005	120.03

$\sigma_{amount > 1200 \vee amount < 75}(loan)$

1234	001	1,923.03
2342	004	56.25

Example Queries 4

- Find the IDs of all customers who have a loan, an account, or both, from the bank

borrower

customer-id	***loan-number***
201	1234
304	3421
425	2342
109	4531

depositor

customer-id	***account-number***
333	3467
304	2312
201	9999
492	3423

$\pi_{customer\text{-}id}(borrower)$

201
304
425
109

201
304
425
109
333
492

$\pi_{customer\text{-}id}(depositor)$

333
304
201
492

$\pi_{customer\text{-}id}(borrower) \cup \pi_{customer\text{-}id}(depositor)$

Example Queries 5

Note this example is split over two slides!

Find the IDs of all customers who have a loan at branch 001.

We retrieve *borrower* and *loan*…

borrower

customer-id	***loan-number***
201	1234
304	3421

loan

loan-number	***branch-id***	***amount***
1234	001	1,923.03
3421	002	123.00

…we calculate their cross product…

customer-id	***borrower.loan-number***	***loan.loan-number***	***branch-id***	***amount***
201	1234	1234	001	1,923.03
201	1234	3421	002	123.00
304	3421	1234	001	1,923.03
304	3421	3421	002	123.00

…we calculate their cross product…

customer-id	*borrower.loan-number*	*loan.loan-number*	*branch-id*	*amount*
201	1234	1234	001	1,923.03
201	1234	3421	002	123.00
304	3421	1234	001	1,923.03
304	3421	3421	002	123.00

…we select the rows where *borrower.loan-number* is equal to *loan.loan-number*…

customer-id	*borrower.loan-number*	*loan.loan-number*	*branch-id*	*amount*
201	1234	1234	001	1,923.03
304	3421	3421	002	123.00

…we select the rows where *branch-id* is equal to *"001"*

customer-id	*borrower.loan-number*	*loan.loan-number*	*branch-id*	*amount*
201	1234	1234	001	1,923.03

…we project out the *customer-id.*

201

$$\pi_{customer\text{-}id}\,(\sigma_{branch\text{-}id=\text{'001'}}\,(\sigma_{borrower.loan\text{-}number\,=\,loan.loan\text{-}number}(borrower \times loan)))$$

Now Using Natural Join

Find the IDs of all customers who have a loan at branch 001.

We retrieve *borrower* and *loan*…

borrower

customer-id	***loan-number***
201	1234
304	3421

loan

loan-number	***branch-id***	***amount***
1234	001	1,923.03
3421	002	123.00

1234 in *borrower* is matched with 1234 in *loan*…

3421 in *borrower* is matched with 3421 in *loan*…

customer-id	***loan-number***	***branch-id***	***amount***
201	1234	001	1,923.03
304	3421	002	123.00

The rest is the same.

customer-id	***loan-number***	***branch-id***	***amount***
201	1234	001	1,923.03

$\pi_{customer\text{-}id}\ (\sigma_{branch\text{-}id='001'}\ (\sigma_{borrower.loan\text{-}number\ =\ loan.loan\text{-}number}(borrower \times loan)))$

$= \quad \pi_{customer\text{-}id}\ (\sigma_{branch\text{-}id='001'}\ (borrower \bowtie loan))$

Example Queries 7

Note this example is split over two slides!

- Find the *names* of all customers who have a loan, an account, or both, from the bank

$\pi_{customer\text{-}id}\,(borrower) \cup \pi_{customer\text{-}id}\,(depositor)$

201
304
425
109
333
492

customer

customer-id	***customer-name***	***customer-city***
101	Carol	Fairfax
109	David	Fairfax
201	John	Vienna
304	Mary	McLean
333	Ben	Chantilly
425	David	Manassas
492	Jason	Fairfax
501	Adam	Burke

Example Queries

- Find the *names* of all customers who have a loan, an account, or both, from the bank

customer-id	***customer-name***	***customer-city***
109	David	Fairfax
201	John	Vienna
304	Mary	McLean
333	Ben	Chantilly
425	David	Manassas
492	Jason	Fairfax

customer-name
David
John
Mary
Ben
David
Jason

customer-name
David
John
Mary
Ben
Jason

$$\pi_{customer\text{-}name}((\pi_{customer\text{-}id}(borrower) \cup \pi_{customer\text{-}id}(depositor)) \bowtie \text{customer})$$

Example Queries 8

Note this example is split over three slides!

Find the largest account balance

account

account-number	***branch-id***	***balance***
7777	001	100.30
8888	003	12.34
6666	004	45.34

We do a rename to get a "copy" of the balance column from *account*. We call this copy *d*…

d

balance
100.30
12.34
45.34

… next we will do a cross product…

… do a cross product…

account-number	*branch-id*	*account.balance*	*d.balance*
7777	001	100.30	100.30
7777	001	100.30	12.34
7777	001	100.30	45.34
8888	003	12.34	100.30
8888	003	12.34	12.34
8888	003	12.34	45.34
6666	004	45.34	100.30
6666	004	45.34	12.34
6666	004	45.34	45.34

…select out all rows where *account.balance* is less than *d.balance*…

account-number	*branch-id*	*account.balance*	*d.balance*
8888	003	12.34	100.30
8888	003	12.34	45.34
6666	004	45.34	100.30

account-number	*branch-id*	*account.balance*	*d.balance*
8888	003	12.34	100.30
8888	003	12.34	45.34
6666	004	45.34	100.30

.. next we project out *account.balance*…

…then we do a set difference between it and the original *account.balance* from the account relation…

… the set difference leaves us with one number, the largest value!

balance from account

balance
100.30
12.34
45.34

—

account.balance
12.34
45.34

100.30

$\pi_{balance}(account) - \pi_{account.balance}(\sigma_{account.balance < d.balance}(account \times \rho(d, \pi_{balance}(account))))$

Now Using Conditional Join

Find the largest account balance

$\pi_{balance}(account) - \pi_{account.balance}(\sigma_{account.balance < d.balance}(account \times \rho(d, \pi_{balance}(account))))$

$\rho(d, \pi_{account.balance}(account))$

$\pi_{balance}(account) - \pi_{account.balance}(account \bowtie_{account.balance < d.balance} d)$

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

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

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

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

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:

- A more efficient solution: Find the bids of red boats first before doing the join.

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

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:

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:

- Can also define Tempboats using union:

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

Find sailors who’ve reserved a red and a green boat

Find sailors who've reserved a red and a green boat

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

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

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

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

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 <u>all</u> 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) <- that is a minus sign.
  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 <u>all</u> 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 = π_{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.

     π_{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)

     π_{sid} (R1) – π_{sid} (AllPossibleRes – AllRedRes)

Division Operation

r / s

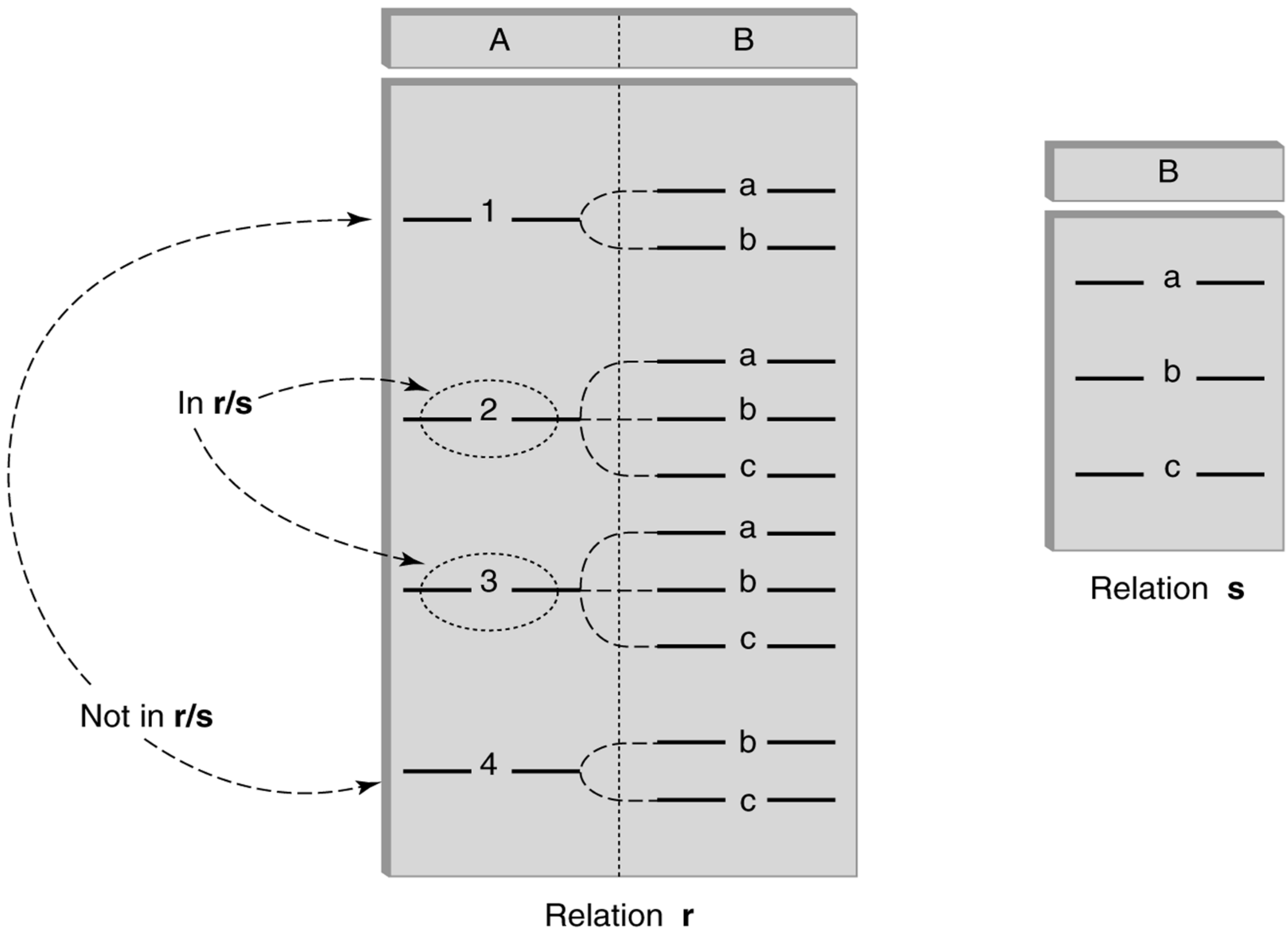
- Suited to queries that include the phrase "for all", e.g. *Find sailors who have reserved **all** red boats*.
- Produce the tuples in one relation, r, that match *all* tuples in another relation, s
- Let *S1* have 2 fields, *x* and *y*; *S2* have only field *y*:
  - $S1/S2 = \left\{ \langle x \rangle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

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

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:

$$\text{Tempsids} = (\pi_{sid,bid}(\text{Reserves})) / (\pi_{bid}(Boats))$$

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

- 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