

SDSC5003

Relational Algebra

- **RG Chapter 4.1,4.2**
- **GUW Chapter 2.4**

Relational Query Languages

- ▶ Query languages: Allow manipulation and retrieval of data from a database.
- ▶ Relational model supports simple, powerful QLs:
 - ▶ Strong formal foundation based on algebra/logic.
 - ▶ Allows for much optimization.
- ▶ A query language is not a general purpose programming language
 - ▶ They are not Turing complete, and
 - ▶ Are not intended for complex calculations

Formal Relational Query Languages

- Two mathematical Query Languages form the basis for “real” languages (e.g. SQL), and for implementation:
 - Relational Algebra: More **operational**, very useful for representing execution plans.
 - Relational Calculus: Lets users describe what they want, rather than how to compute it. (**Non-operational**, declarative.) Not covered in course.
- Understanding formal query languages is important for understanding
 - The origins of SQL
 - Query processing and optimization



Procedural vs. Non-procedural

- A procedural query consists of a series of operations
 - Which describe a step-by-step process for calculating the answer
 - e.g. giving precise directions on how to get somewhere
 - "..., turn left at the blue sign, drive for 1.5 miles, turn right at the cat, then left again at the red tower, ..."
- A non-procedural (or declarative) query describes *what* output is desired, and not *how* to compute it
 - e.g. giving the address someone is to go to
 - "go to the HSBC bank at 15th"



Preliminaries

- A query is applied to *relation instances*, and the result of a query is also a relation instance.
 - *Schemas of input* relations for a query are *fixed*
 - The *schema for the result* of a given query is also *fixed!* Determined by definition of query language constructs.



Preliminaries

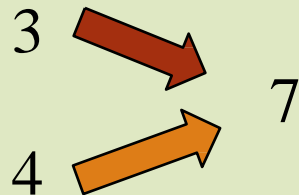
- Fields in an instance can be referred to either by position or by name
- Positional vs. named-attribute notation:
 - Positional notation
 - Ex: Sailor(1,2,3,4)
 - easier for formal definitions
 - Named-attribute notation
 - Ex: Sailor(sid, sname, rating, age)
 - more readable



Relational Algebra

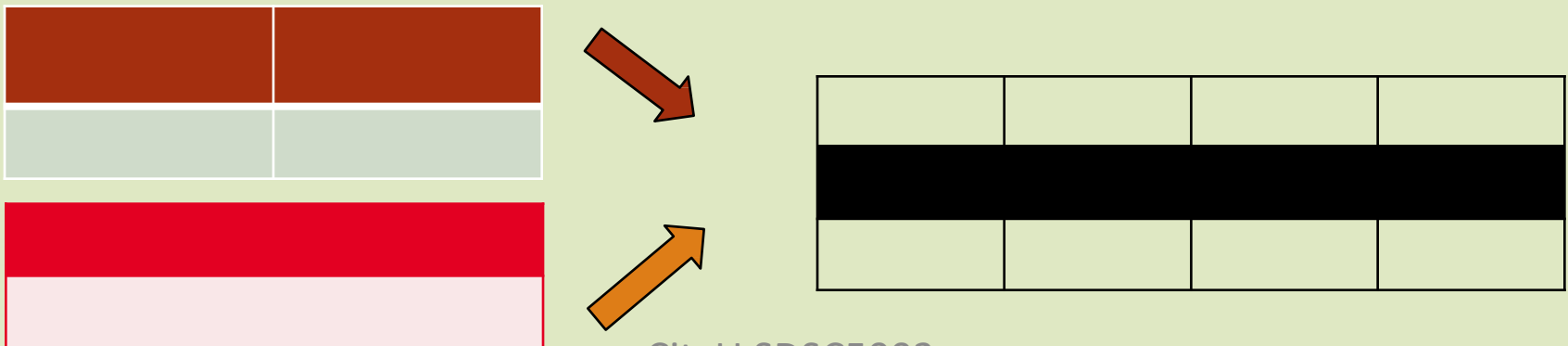
Algebra

- In math, algebraic operations like $+$, $-$, \times , $/$.
- Operate on numbers: input are numbers, output are numbers.
- Can also do Boolean algebra on sets, using union, intersect, difference.
- Focus on **algebraic identities**, e.g.
 - $x(y+z) = xy + xz$.
- (Relational algebra lies between propositional and 1st-order logic.)



Relational Algebra

- Every operator takes one or two relation instances
- A relational algebra expression is recursively defined to be a relation
 - Result is also a relation
 - Can apply operator to
 - Relation from database
 - Relation as a result of another operator



Relational Algebra Operations

Basic operations:

- Selection (σ) Selects a subset of rows from relation.
- Projection (π) Selects a subset of 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 in reln. 2.

Additional derived operations:

- Intersection, join, division, renaming.
Not essential, but very useful.
- Since each operation returns a relation, *operations can be composed!*

Example Instances

- “Sailors” and “Reserves” relations for our examples.
- We’ll use positional or named field notation.
- Assume that names of fields in query results are inherited from names of fields in query input relations.

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

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

Projection

- The projection operator, π (π), specifies the columns to be retained from the input relation
- A selection has the form:

$\pi_{columns}(relation)$

- Where *columns* is a comma separated list of column names
- The list contains the names of the columns to be retained in the result relation

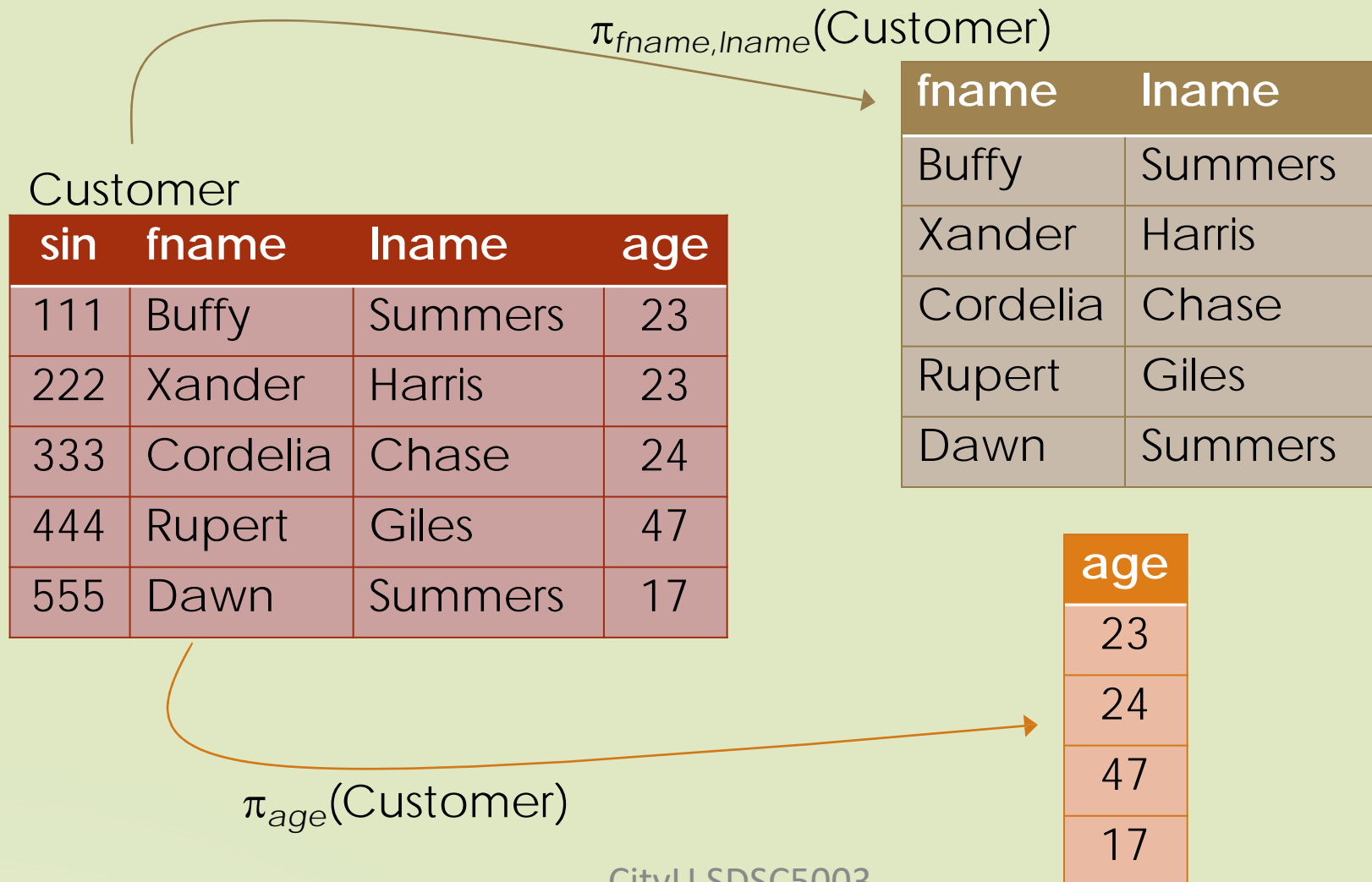
sname	rating
yuppy	9
lubber	8
guppy	5
rusty	10

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

age
35.0
55.5

$\pi_{age}(S2)$

More Projection Example



Selection

- ▶ The selection operator, σ (sigma), specifies the rows to be retained from the input relation
- ▶ A selection has the form: $\sigma_{condition}(relation)$, where *condition* is a Boolean expression
 - ▶ Terms in the condition are comparisons between two fields (or a field and a constant)
 - ▶ Using one of the comparison operators: $<$, \leq , $=$, \neq , \geq , $>$
 - ▶ Terms may be connected by \wedge (and), or \vee (or),
 - ▶ Terms may be negated using \neg (not)

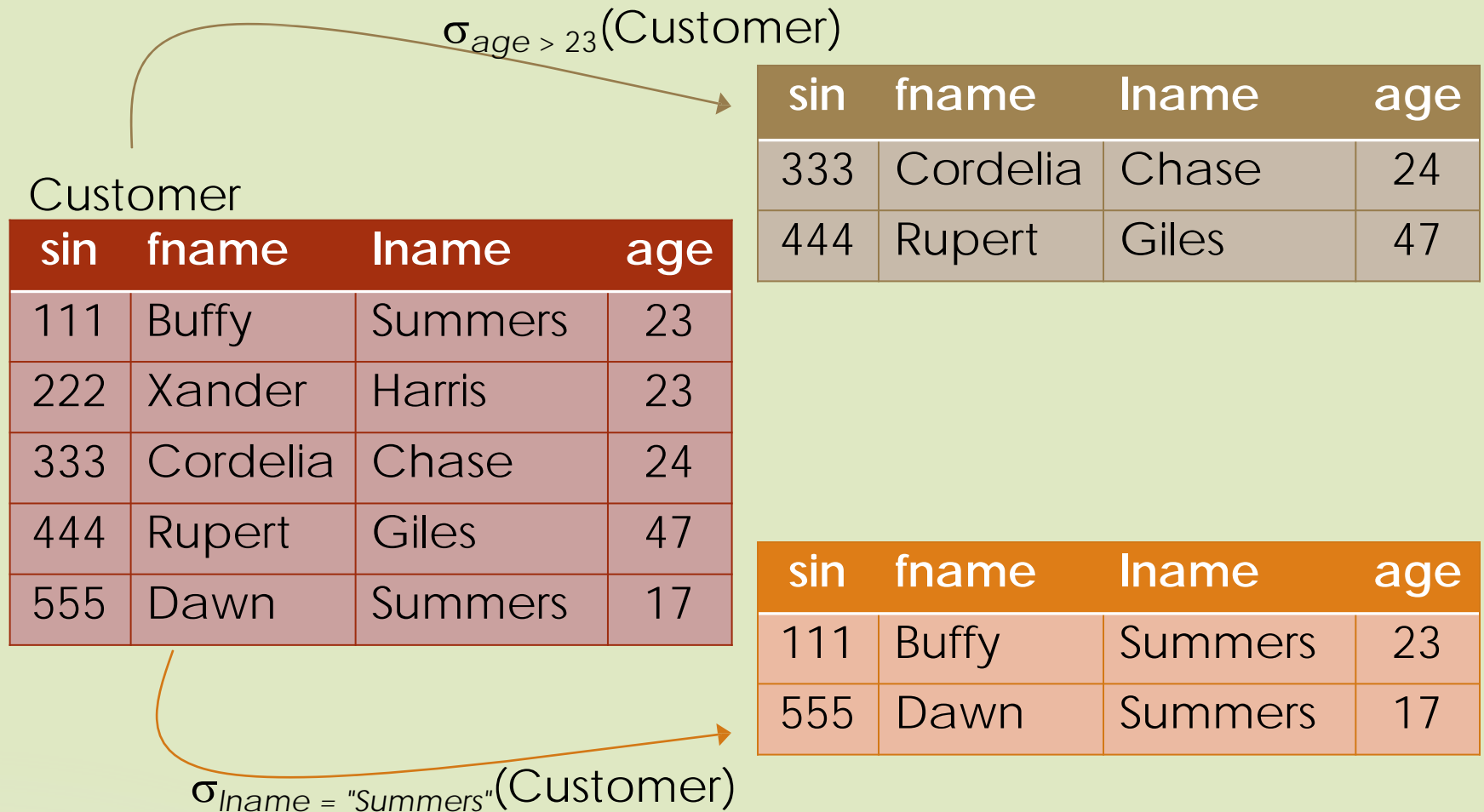
sid	sname	rating	age
28	yuppy	9	35.0
58	rusty	10	35.0

$$\sigma_{rating > 8}(S_2)$$

sname	rating
yuppy	9
rusty	10

$$\pi_{sname, rating}(\sigma_{rating > 8}(S_2))$$

More Selection Example



Selection and Projection

$\pi_{sin, fname}(\sigma_{age > 18 \wedge lname = "Summers"}(Customer))$

Customer

sin	fname	lname	age
111	Buffy	Summers	23
222	Xander	Harris	23
333	Cordelia	Chase	24
444	Rupert	Giles	47
555	Dawn	Summers	17

intermediate relation

sin	fname	lname	age
111	Buffy	Summers	23

sin	fname
111	Buffy



Selection and Projection Notes

- Selection and projection eliminate duplicates
 - Because relations are sets
 - In practice (SQL), duplicate elimination is expensive, therefore it is only done when explicitly requested
- Both operations require one input relation
- The schema of the result of a selection is *the same as* the schema of the input relation
- The schema of the result of a projection contains just those attributes in the projection list

Set Operations

- Relational algebra includes the standard set operations:

- Union, \cup
- Intersection, \cap
- Set Difference, $-$
- Cartesian product (Cross product), \times

These are all binary operators

- All relational algebra operations can be implemented using these five basic operations:
 - *Selection, projection, union, set difference and Cartesian product*

Set Operations Review

$A = \{1, 3, 6\}$	$B = \{1, 2, 5, 6\}$	
Union (\cup)	$A \cup B \equiv B \cup A$	$A \cup B = \{1, 2, 3, 5, 6\}$
Intersection (\cap)	$A \cap B \equiv B \cap A$	$A \cap B = \{1, 6\}$
Set Difference ($-$)	$A - B \neq B - A$	$A - B = \{3\}$ $B - A = \{2, 5\}$
Cartesian Product (\times)	$A \times B \equiv B \times A^*$	

$$A \times B = \{(1,1), (1,2), (1,5), (1,6), (3,1), (3,2), (3,5), (3,6), (6,1), (6,2), (6,5), (6,6)\}$$

* not strictly true, as pairs are ordered

Example Instances

- “Sailors” and “Reserves” relations for our examples.
- We’ll use positional or named field notation.
- Assume that names of fields in query results are inherited from names of fields in query input relations.

R1

sid	bid	day
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

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

Union, Intersection, Set-Difference

- All these operations take two input relations, which must be union-compatible:

- Same number of fields.
- Corresponding fields have the same type.

- What is the *schema* of result?

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0
44	guppy	5	35.0
28	yuppy	9	35.0

$S1 \cup S2$

sid	sname	rating	age
22	dustin	7	45.0

$S1 - S2$

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

$S1 \cap S2$

Exercise on Union

Num ber	shape	holes
1	round	2
2	square	4
3	rectangle	8

Blue blocks (BB)

Stacked(S)

bottom	top
4	2
4	6
6	2

CityU SDSC5003

Num ber	shape	holes
4	round	2
5	square	4
6	rectangle	8

Yellow blocks(YB)

1. Which tables are union-compatible?
2. What is the result of the possible unions?

More Union Compatible Relations

Intersection of the Employee and Customer relations

Customer

sin	fname	lname	age
111	Buffy	Summers	23
222	Xander	Harris	23
333	Cordelia	Chase	24
444	Rupert	Giles	47
555	Dawn	Summers	17

Employee

sin	fname	lname	salary
208	Clark	Kent	80000.55
111	Buffy	Summers	22000.78
412	Carol	Danvers	64000.00

The two relations are not union compatible as age is an INTEGER and salary is a REAL

More Union Compatible Relations

Intersection of the Employee and Customer relations

Customer

sin	fname	lname	age
111	Buffy	Summers	23
222	Xander	Harris	23
333	Cordelia	Chase	24
444	Rupert	Giles	47
555	Dawn	Summers	17

Employee

sin	fname	lname	salary
208	Clark	Kent	80000.55
111	Buffy	Summers	22000.78
412	Carol	Danvers	64000.00

The two relations are not union compatible as age is an INTEGER and salary is a REAL

However we can carry out some preliminary operations to make the relations union compatible:

$$\pi_{sin, fname, lname}(\text{Customer}) \cap \pi_{sin, fname, lname}(\text{Employee})$$

More Union Example

R

sin	fname	lname
111	Buffy	Summers
222	Xander	Harris
333	Cordelia	Chase
444	Rupert	Giles
555	Dawn	Summers

S

sin	fname	lname
208	Clark	Kent
111	Buffy	Summers
412	Carol	Danvers

sin	fname	lname
111	Buffy	Summers
222	Xander	Harris
333	Cordelia	Chase
444	Rupert	Giles
555	Dawn	Summers
208	Clark	Kent
412	Carol	Danvers

$R \cup S$

Returns all records in either relation (or both)

More Intersection Example

R

sin	fname	lname
111	Buffy	Summers
222	Xander	Harris
333	Cordelia	Chase
444	Rupert	Giles
555	Dawn	Summers

sin	fname	lname
111	Buffy	Summers

S

sin	fname	lname
208	Clark	Kent
111	Buffy	Summers
412	Carol	Danvers

$R \cap S$

Only returns records that are in both relations

More Set Difference Example

R

sin	fname	lname
111	Buffy	Summers
222	Xander	Harris
333	Cordelia	Chase
444	Rupert	Giles
555	Dawn	Summers

$R - S$

sin	fname	lname
222	Xander	Harris
333	Cordelia	Chase
444	Rupert	Giles
555	Dawn	Summers

$R - S$ returns all records in R that are not in S

S

sin	fname	lname
208	Clark	Kent
111	Buffy	Summers
412	Carol	Danvers

$S - R$

sin	fname	lname
208	Clark	Kent
412	Carol	Danvers



Cartesian (Cross) Product

- The *schema* of the result of $R \times S$ has one attribute for each attribute of the input relations
 - All of the fields of R , followed by all of the fields of S
 - Names are inherited if possible (i.e. if not duplicated)
 - If two field names are the same, a *naming conflict* occurs and the affected columns are referred to by position
- The result *relation* contains one record for each pair of records $r \in R, s \in S$
 - Each record in R is paired with each record in S
 - If R contains m records, and S contains n records, the result relation will contain $m * n$ records

Example Instances

- “Sailors” and “Reserves” relations for our examples.
- We’ll use positional or named field notation.
- Assume that names of fields in query results are inherited from names of fields in query input relations.

R1

sid	bid	day
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

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

Cross-Product

- Each row of S1 is paired with each row of R1.
- Result schema* has one field per field of S1 and R1, with field names inherited if possible.
- Conflict*: Both S1 and R1 have a field called *sid*.

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

- Renaming operator*: $\rho (C(1 \rightarrow sid1, 5 \rightarrow sid2), S1 \times R1)$



Renaming

- It is sometimes useful to assign names to the results of a relational algebra query
- The rename operator, ρ (rho) allows a relational algebra expression to be renamed
 - $\rho_x(E)$ names the result of the expression, or
 - $\rho_{x(A1,A2,\dots,A_n)}(E)$ names the result of the expression, and its attributes
- For example, find the account with the largest balance

Largest Balance

- To find the largest balance first find accounts which are less than some other account
 - By performing a comparison on the Cartesian product of the account table with itself
 - The *Account* relation is referred to twice so with no renaming we have an ambiguous expression:
 - $\sigma_{\text{account.balance} < \text{account.balance}} (\text{Account} \times \text{Account})$
 - So rename one version of the Account relation
 - $\sigma_{\text{account.balance} < d.\text{balance}} (\text{Account} \times \rho_d (\text{Account}))$
- Then use set difference to find the largest balance

Exercise on Cross-Product

Num ber	shape	holes
1	round	2
2	square	4
3	rectangle	8

Num ber	shape	holes
4	round	2
5	square	4
6	rectangle	8

Blue blocks (BB)

Stacked(S)

bottom	top
4	2
4	6
6	2

1. Write down 2 tuples in $BB \times S$.
2. What is the cardinality of $BB \times S$?



Joins

- It is often useful to simplify some queries that require a Cartesian product
- There is often a natural way to join two relations
 - e.g., finding data about customers and their accounts
 - Assume *Account* has a foreign key, that references *Customer*
 - Get the Cartesian product of *Account* and *Customer*
 - Select the tuples where the primary key of *Customer* equals the foreign key attribute in *Account*

Example Instances

- “Sailors” and “Reserves” relations for our examples.
- We’ll use positional or named field notation.
- Assume that names of fields in query results are inherited from names of fields in query input relations.

R1

sid	bid	day
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

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

Joins (cont.)

► Condition Join: $R \bowtie_c S = \sigma_c (R \times S)$

(sid)	sname	rating	age	(sid)	bid	day
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.5	58	103	11/12/96

$$S1 \bowtie_{S1.sid < R1.sid} R1$$

- *Result schema* same as that of cross-product.
- Fewer tuples than cross-product, might be able to compute more efficiently. How?
- Sometimes called a *theta-join*.

Exercise on Join

Num ber	shape	holes
1	round	2
2	square	4
3	rectangle	8

Blue blocks (BB)

Num ber	shape	holes
4	round	2
5	square	4
6	rectangle	8

Yellow blocks(YB)

$$BB \bowtie_{BB.holes < YB.holes} YB$$

Write down 2 tuples in this join.

Joins (cont.)

- Equi-Join: A special case of condition join where the condition c contains only *equalities*.

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

$$S1 \bowtie_{Rsid=S.sid} R1$$

- Result schema similar to cross-product, but only one copy of fields for which equality is specified.
- Natural Join: Equi-join on *all* common fields.
Without specified condition
means the natural join of A and B.

$$A \bowtie B$$

Example for Natural Join

Num ber	shape	holes
1	round	2
2	square	4
3	rectangle	8

Blue blocks (BB)

shape	holes
round	2
square	4
rectangle	8

Yellow blocks(YB)

What is the natural join of BB and YB?

More Natural Join Example

Customer

sin	fname	lname	age
111	Buffy	Summers	23
222	Xander	Harris	23
333	Cordelia	Chase	24
444	Rupert	Giles	47
555	Dawn	Summers	17

Employee

sin	fname	lname	salary
208	Clark	Kent	80000.55
111	Buffy	Summers	22000.78
396	Dawn	Allen	41000.21
412	Carol	Danvers	64000.00

Customer ⋈ Employee

sin	fname	lname	age	salary
111	Buffy	Summers	23	22000.78

More Theta Join Example

Customer

sin	fname	lname	age
111	Buffy	Summers	23
222	Xander	Harris	23
333	Cordelia	Chase	24
444	Rupert	Giles	47
555	Dawn	Summers	17

Employee

sin	fname	lname	salary
208	Clark	Kent	80000.55
111	Buffy	Summers	22000.78
412	Carol	Danvers	64000.00

Customer ⋈_{Customer.sin < Employee.sin} Employee

1	2	3	age	5	6	7	salary
111	Buffy	Summers	23	208	Clark	Kent	80000
111	Buffy	Summers	23	412	Carol	Danvers	64000
222	Xander	Harris	23	412	Carol	Danvers	64000
333	Cordelia	Chase	17	412	Carol	Danvers	64000

Complex Exercises


- ▶ “Sailors” and “Reserves” relations for our examples.
- ▶ We’ll use positional or named field notation.
- ▶ Assume that names of fields in query results are inherited from names of fields in query input relations.

Reserves

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

Sailors

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




Find names of sailors who've reserved boat #103

Reserves

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

Sailors

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



Find names of sailors who've reserved boat #103

➤ Solution 1:

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

➤ Solution 2:

$$\rho(Temp1, \sigma_{bid=103} Reserves)$$

$$\rho(Temp2, Temp1 \bowtie Sailors)$$

$$\pi_{sname}(Temp2)$$

➤ Solution 3:

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

Exercise: Find names of sailors
who've reserved a red boat

Reserves

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

Sailors

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

Boats

<u>bid</u>	color
101	red
102	green
103	green



Exercise: Find names of sailors who've reserved a red boat

- Information about boat color only available in Boats; so need an extra join:

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



Exercise: 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:

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

A query optimizer can find this, given the first solution!



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

Reserves

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

Sailors

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

Boats

<u>bid</u>	color
101	red
102	green
103	green

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

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

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

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


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

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

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

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

- Can also define Tempboats using union! (How?)
- What happens if \vee is replaced by \wedge in this query?



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

Reserves


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

Sailors

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

Boats

<u>bid</u>	color
101	red
102	green
103	green



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

- ▶ Previous approach won't work! Must identify sailors who've reserved red boats, sailors who've reserved green boats, then find the intersection (note that *sid* is a key for *Sailors*):

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

- Previous approach won't work! Must identify sailors who've reserved red boats, sailors who've reserved green boats, then find the intersection (note that *sid* is a key for *Sailors*):

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

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

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



Division

- Division is useful for queries which return records associated with *all* of the records in some subset
 - Find people who like all types of music
 - Country *and* Western
 - Find tourists who have visited all of the provinces in Canada
- This operator is always implemented in DBMSs
- But it can be expressed in terms of the basic set operators
 - Implementing a division query in SQL is a *fun* exercise

Examples of Division 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

sno
s1
s2
s3
s4

A/B1

pno
p2
p4

B2

sno
s1
s4


A/B2

pno
p1
p2
p4

B3

sno
s1

A/B3



Find the names of sailors who've reserved all boats

Reserves


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

Sailors

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

Boats

<u>bid</u>	color
101	red
102	green
103	green



Find the names of sailors who've reserved all boats

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

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

$$\pi_{sname} (Tempsids \bowtie Sailors)$$

Find the names of sailors who've reserved all boats

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

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

$$\pi_{sname} (Tempsids \bowtie Sailors)$$

- To find sailors who have reserved all red boats:

$$\dots / \pi_{bid} (\sigma_{color='red'} Boats)$$



Summary

- Relational algebra is a procedural language that is used as the internal representation of SQL
- There are five basic operators: selection, projection, cross-product, union and set difference
 - Additional operators are defined in terms of the basic operators: intersection, join, and division
- There are often several equivalent ways to express a relational algebra
 - These equivalencies are exploited by query optimizers to re-order the operations in a relational algebra expression (choose the most efficient one)