By Marina Barsky

Relational algebra

Lecture 5

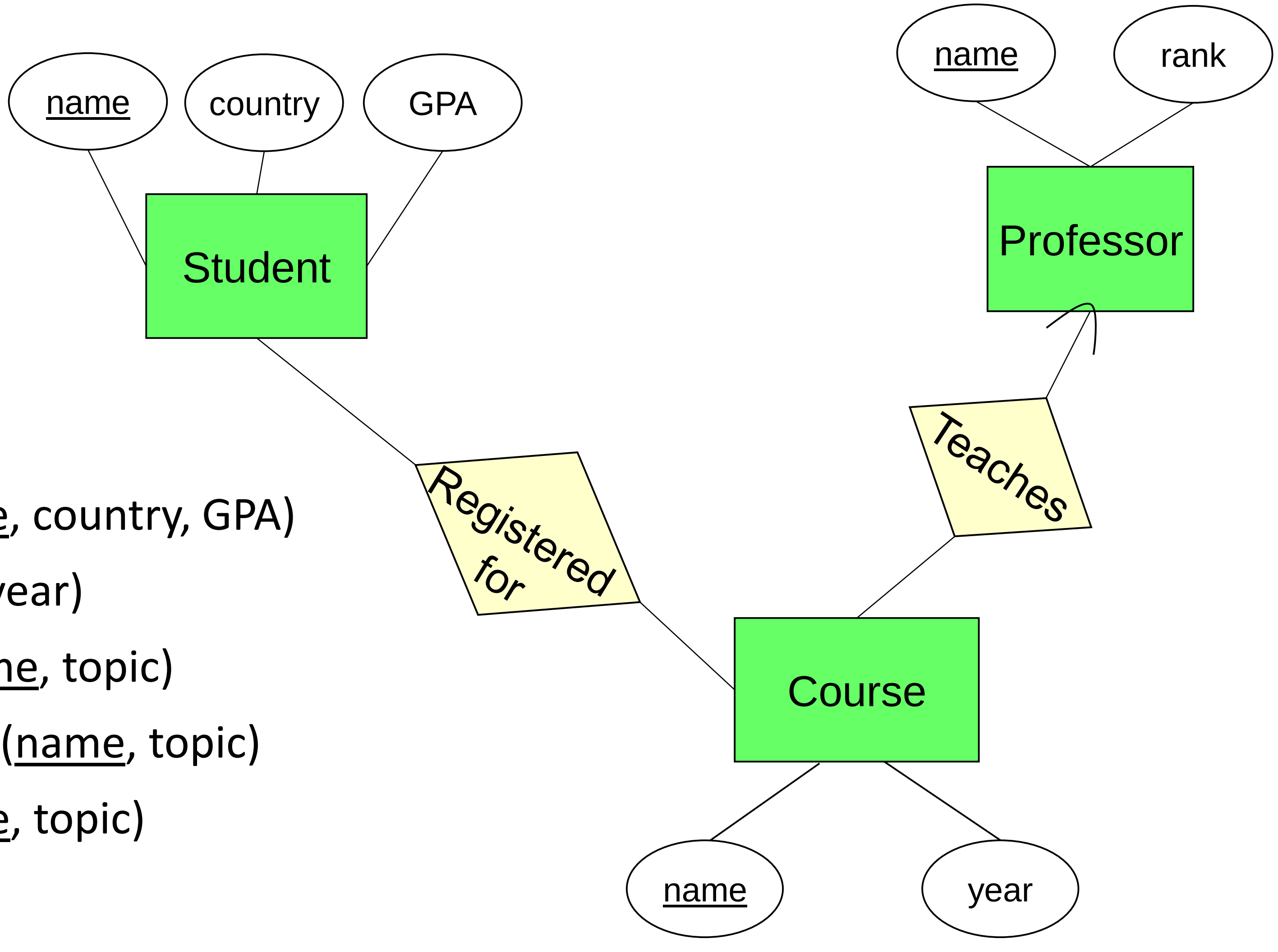
Relations: what are they?

- *Relations* are records of related facts or properties for each entity in the entity set
- How the facts are related is defined through the list of attributes
- The facts themselves are represented as tuples of values – one value for each attribute

Facts required to be different – relation is a SET

- There are no two completely identical tuples in a given relations
- Each relation is a **set** of tuples – no duplicates

Consider an example



Student (<u>name</u>, country, GPA)

Couse (<u>topic</u>, year)

Professor (<u>name</u>, topic)

RegisteredFor (<u>name</u>, topic)

Teaches (<u>name</u>, topic)

Sample instances for each relation

Student		
Name	Country	GPA
Bob	Canada	3
John	Britain	3
Tom	Canada	3.5
Maria	Mexico	4

Course	
Topic	Year
Algorithms	2
Python	2
Databases	3
GUI	3

RegisteredFor	
Name	Topic
Bob	Algorithms
John	Algorithms
Tom	Algorithms
Bob	Python
Tom	Python
Bob	Databases
John	Databases
Maria	Databases
John	GUI
Maria	GUI

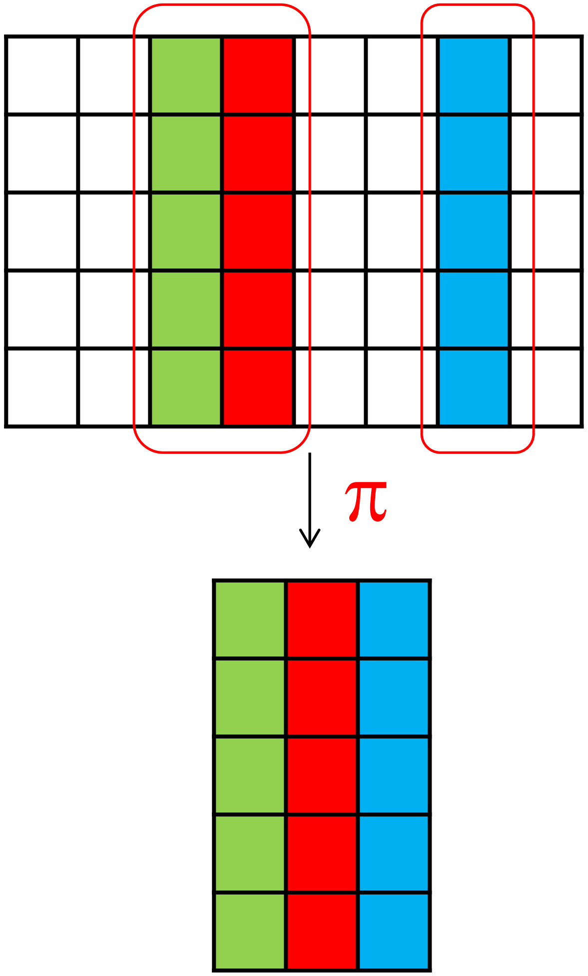
Professor	
Name	Rank
Dr. Monk	Professor
Dr. Pooh	Associate Professor
Dr. Patel	Assistant Professor

Teaches	
Name	Topic
Dr. Monk	Algorithms
Dr. Pooh	Python
Dr. Patel	Databases
Dr. Patel	GUI

Core operators of relational algebra

Slice operations: Projection

Produces from relation **R** a new relation that has only the A_1, …, A_n columns of **R**.



$S=\pi_{\text{attribute list}}(R)$

Projection: example

Query: list names of students

Student			
SIN	Name	GPA	Country
111	Bob	3	Canada
222	John	3	Britain
333	Tom	3.5	Canada
444	Maria	4	Mexico

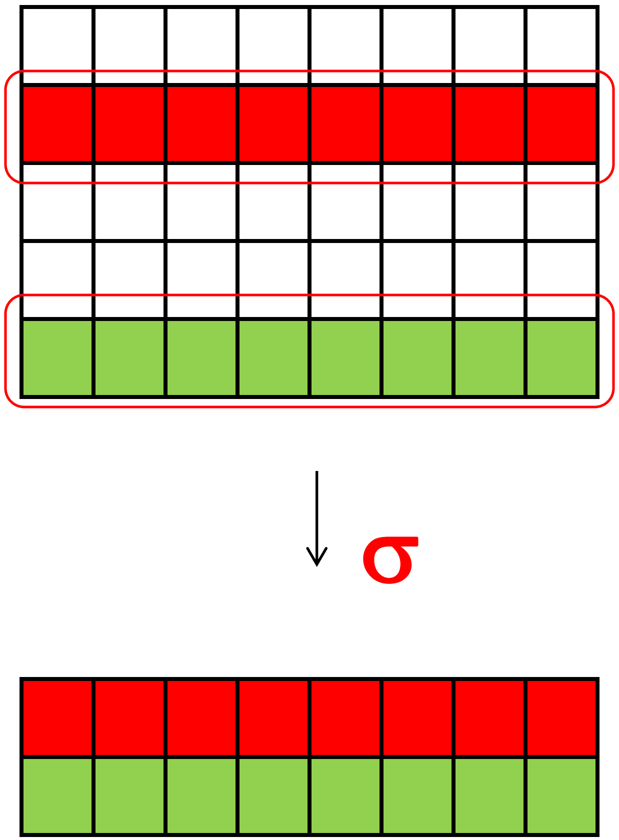
S
Name
Bob
John
Tom
Maria

$S = \pi_{Name}(Student)$

Slice operations: Selection

Produces a new relation with those tuples of **R** which satisfy condition **C**.

$$S = \sigma_{condition}(R)$$



Selection example.

Query: list students with GPA >3

Student		
Name	GPA	Country
Bob	3	Canada
John	3	Britain
Tom	3.5	Canada
Maria	4	Mexico

S		
Name	GPA	Country
Tom	3.5	Canada
Maria	4	Mexico

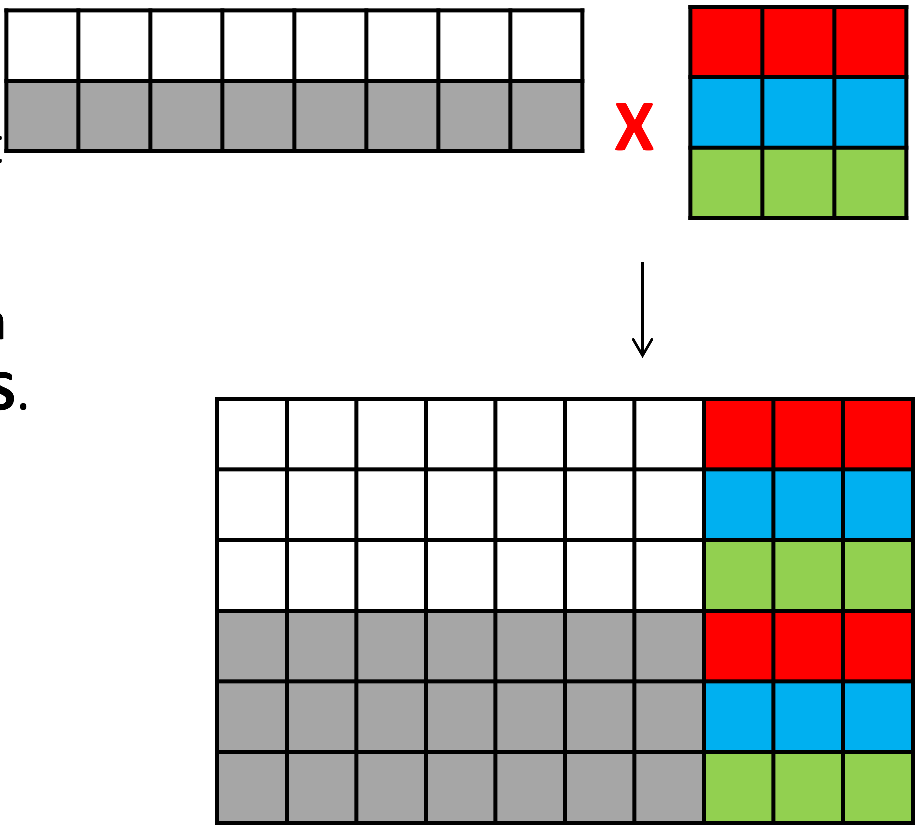
$$S = \sigma_{gpa>3}(Student)$$

Join operation: Cartesian product (Cross-product)

1. Set of tuples ***rs*** that are formed by choosing the first part (***r***) to be any tuple of **R** and the second part (***s***) to be any tuple of **S**.

2.Schema for the resulting relation is the union of schemas for **R** and **S**.

3.If **R** and **S** happen to have some attributes in common, then prefix those attributes by the relation name.



T=R x S

Cartesian product example

T=Course x Professor

Course	
Topic	Year
Algorithms	2
Python	2
Databases	3
GUI	3

Professor	
Name	Rank
Dr. Monk	Professor
Dr. Pooh	Associate Professor
Dr. Patel	Assistant Professor

Cartesian product output

Algorithms	2
Python	2
Databases	3
GUI	3

Dr. Monk	Dr. Pooh	Dr. Patel
Professor	Associate Professor	Assistant Professor

Topic	Y	Name	Rank
Algorithms	2	Dr. Monk	Professor
Algorithms	2	Dr. Pooh	Assoc. Professor
Algorithms	2	Dr. Patel	Assist. Professor
Python	2	Dr. Monk	Professor
Python	2	Dr. Pooh	Assoc. Professor
Python	2	Dr. Patel	Assist. Professor
Databases	3	Dr. Monk	Professor
Databases	3	Dr. Pooh	Assoc. Professor
Databases	3	Dr. Patel	Assist. Professor
GUI	3	Dr. Monk	Professor
GUI	3	Dr. Pooh	Assoc. Professor
GUI	3	Dr. Patel	Assist. Professor

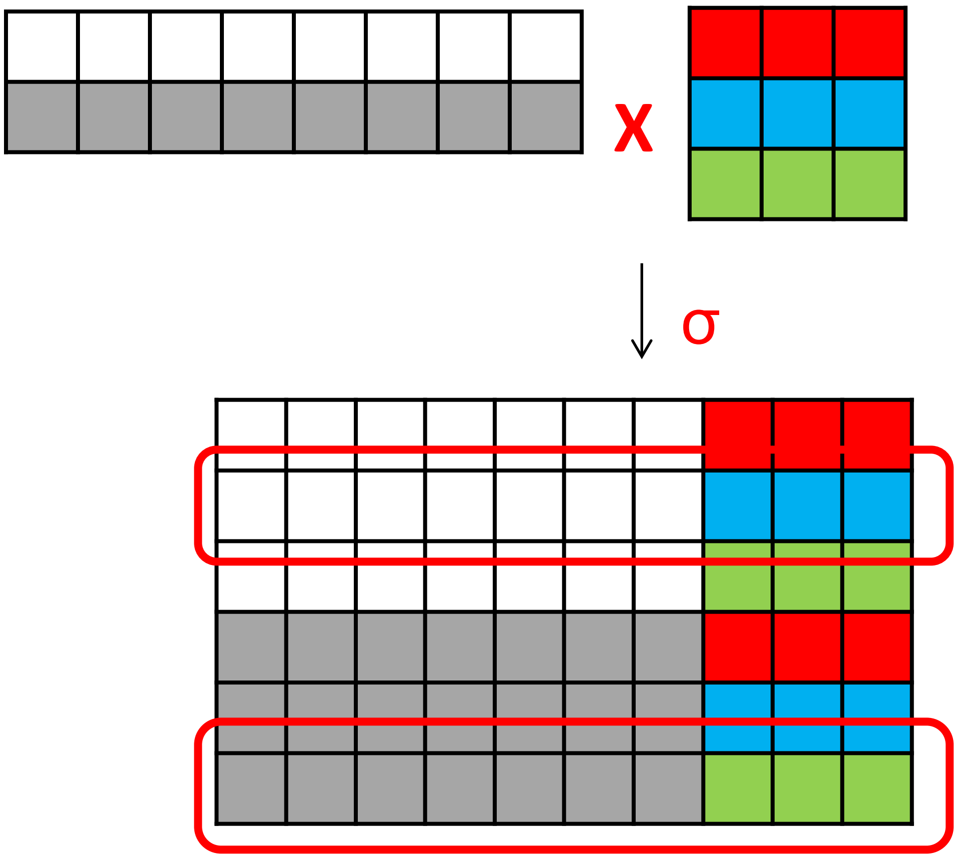
Combining Cross-product with selection

1. The result is constructed as follows:

    a) Take the Cartesian product of **R** and **S**.

    b) Select from the product only those tuples that satisfy the condition **C**.

2. Schema for the result is the union of the schema of **R** and **S,** with **“R”** or **“S”** prefix as necessary.



$$T=\sigma_{condition}\ (R \times S)$$

Example.

Query: Dr. Monk wonders whether he has to teach a multi-cultural group of students

Student		
Name	Country	GPA
Bob	Canada	3
John	Britain	3
Tom	Canada	3.5
Maria	Mexico	4

Teaches	
Name	Topic
Dr. Monk	Algorithms
Dr. Pooh	Python
Dr. Patel	Databases
Dr. Patel	GUI

RegisteredFor	
Name	Topic
Bob	Algorithms
John	Algorithms
Tom	Algorithms
Bob	Python
Tom	Python
Bob	Databases
John	Databases
Maria	Databases
John	GUI
Maria	GUI

Multi-cultural class

Student		
Name	Country	GPA
Bob	Canada	3
John	Britain	3
Tom	Canada	3.5
Maria	Mexico	4

AlgoList	
Name	Topic
Bob	Algorithms
John	Algorithms
Tom	Algorithms

AlgoList $=\sigma_{Topic=Algorithms}$ (RegisteredFor)

Multi-cultural class

Student		
Name	Country	GPA
Bob	Canada	3
John	Britain	3
Tom	Canada	3.5
Maria	Mexico	4

AlgoList	
Name	Topic
Bob	Algorithms
John	Algorithms
Tom	Algorithms

ClassInfo		
Name	Country	GPA
Bob	Canada	3
John	Britain	3
Tom	Canada	3.5

AlgoList $= \sigma_{\text{Topic=Algorithms}}$ (RegisteredFor)

ClassInfo= $\sigma_{\text{Student.name=AlgoList.name}}$ AlgoList x Student

Multi-cultural class

Student		
Name	Country	GPA
Bob	Canada	3
John	Britain	3
Tom	Canada	3.5
Maria	Mexico	4

AlgoList	
Name	Topic
Bob	Algorithms
John	Algorithms
Tom	Algorithms

ClassInfo		
Name	Country	GPA
Bob	Canada	3
John	Britain	3
Tom	Canada	3.5

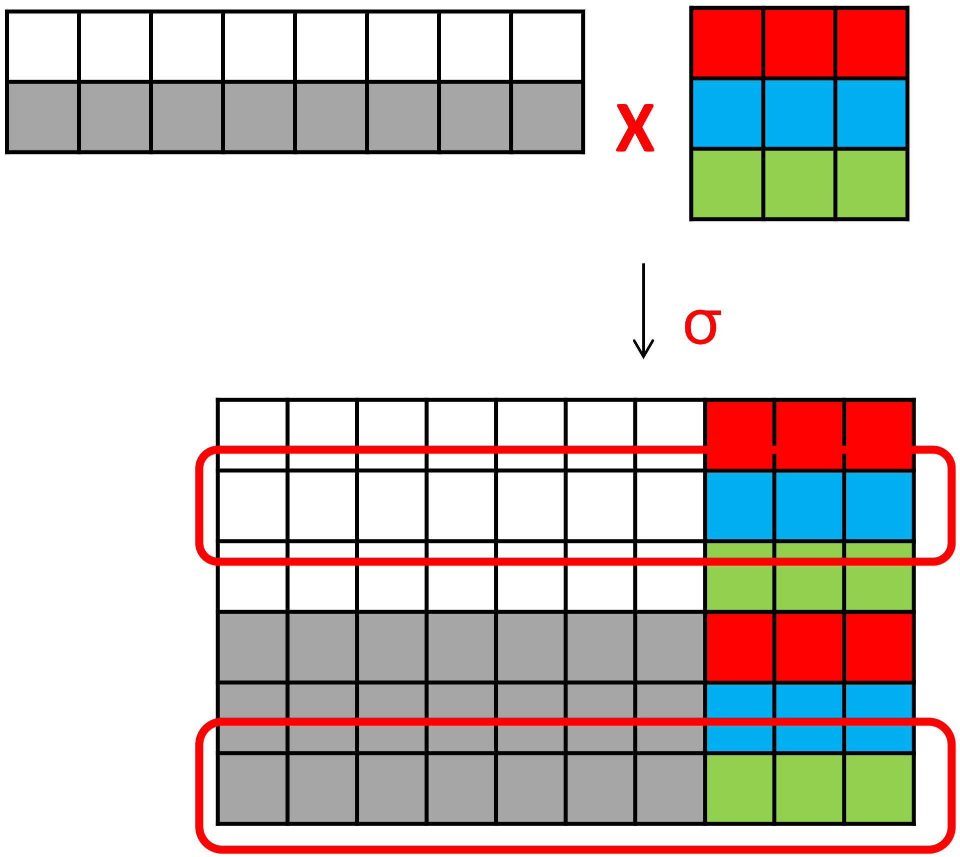
Countries
Country
Canada
Britain

AlgoList $=\sigma_{\text{Topic=Algorithms}}$ (RegisteredFor)

ClassInfo= $\sigma_{\text{Student.name=AlgoList.name}}$ AlgoList x Student

Countries=π_{country} (ClassInfo)

Cross-product with selection



$T=\sigma_{condition}\ (R \times S)$

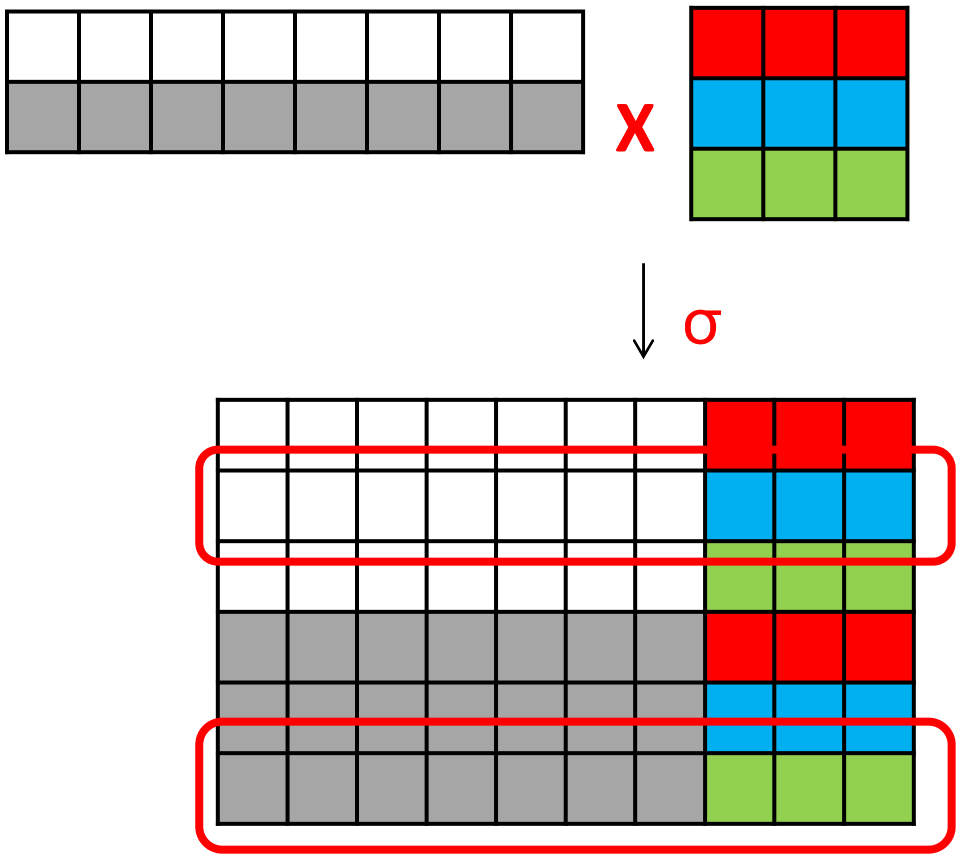
Shortcut: Theta-join

1. The result of this operation is constructed as follows:

    a) Take the Cartesian product of **R** and **S**.

    b) Select from the product only those tuples that satisfy the condition **C**.

2. Schema for the result is the union of the schema of **R** and **S,** with **“R”** or **“S”** prefix as necessary.



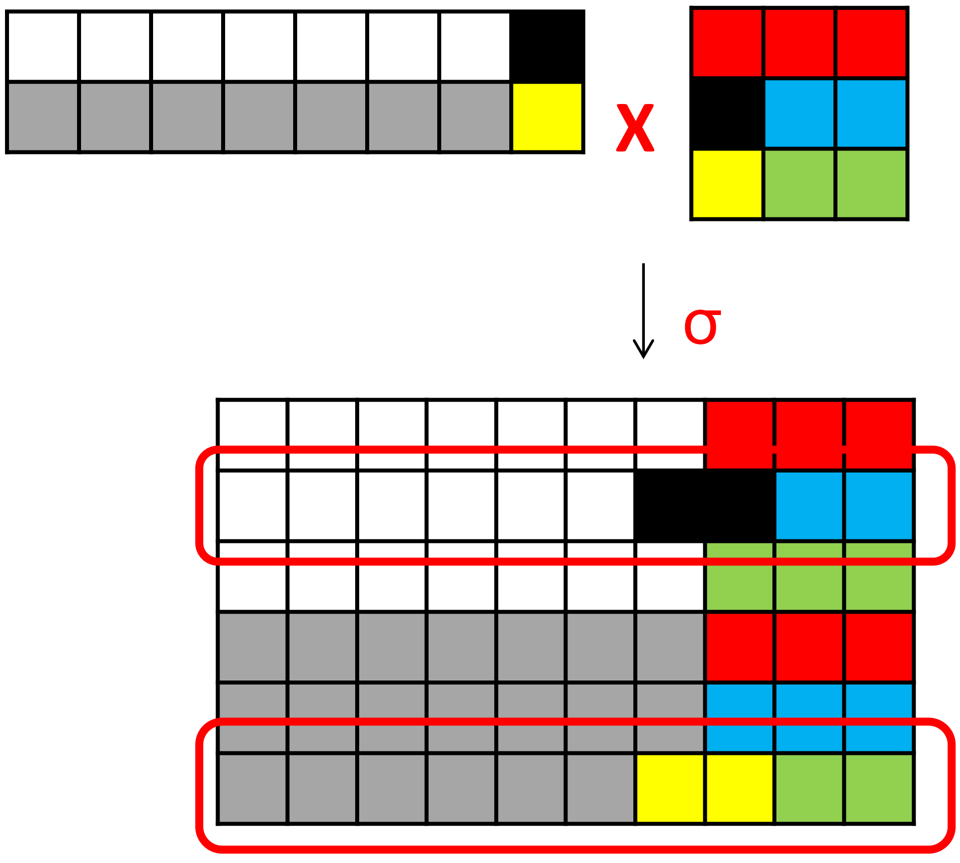
$T= R \bowtie_{condition} S$

Shortcut for

$T=\sigma_{condition} (R \times S)$

Subtype of theta-join: Equijoin

1.Equijoin is a subset of theta-joins where the join condition is equality



T= R ⋈$_{R.A = S.B}$ S

Shortcut for

T=σ$_{R.A = S.B}$ (R x S)

Special case of equijoin: Natural Join

R ⋈ S

Let $\mathbf{A_1, A_2, \ldots, A_n}$ be the attributes in both the schema of **R** and the schema of **S**.

Then a tuple ***r*** from **R** and a tuple ***s*** from **S** are successfully paired if and only if ***r*** and ***s*** agree on each of their common attributes $\mathbf{A_1, A_2, \ldots, A_n}$.

Still the same meaning as:

$T = \sigma_{R.A = S.A} (R \times S)$,

but common attributes are not duplicated as in Cartesian Product

Set Operations on Relations

R ∪ S, the **union** of **R** and **S**, is the set of tuples that are in **R** or **S** or both.

R − S, the **difference** of **R** and **S**, is the set of tuples that are in **R** but not in **S**.

Note that **R − S** is different from **S − R**.

R ∩ S, the **intersection** of **R** and **S**, is the set of tuples that are in both **R** and **S**.

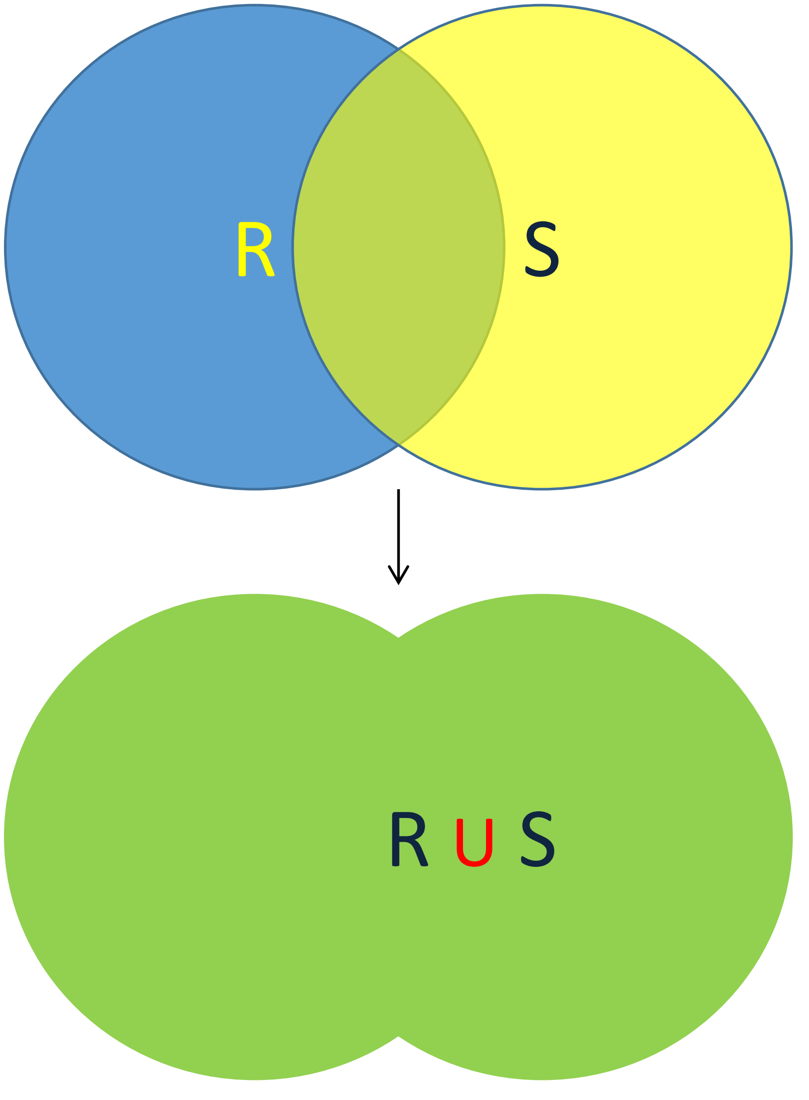
Condition for set operators

Set operators can operate only on two union-compatible relations

Two relations are **union-compatible** if they have the same number of attributes and each attribute must be from the same domain

Union

T=R ∪ S



Union example.

Query: list names of all people in the department

Student		
Name	Country	GPA
Bob	Canada	3
John	Britain	3
Tom	Canada	3.5
Maria	Mexico	4

Professor	
Name	Rank
Dr. Monk	Professor
Dr. Pooh	Associate Professor
Dr. Patel	Assistant Professor

Can we do ?

T=Student $\cup$ Professor

Union example.

Query: list names of all people in the department

Student
Name
Bob
John
Tom
Maria

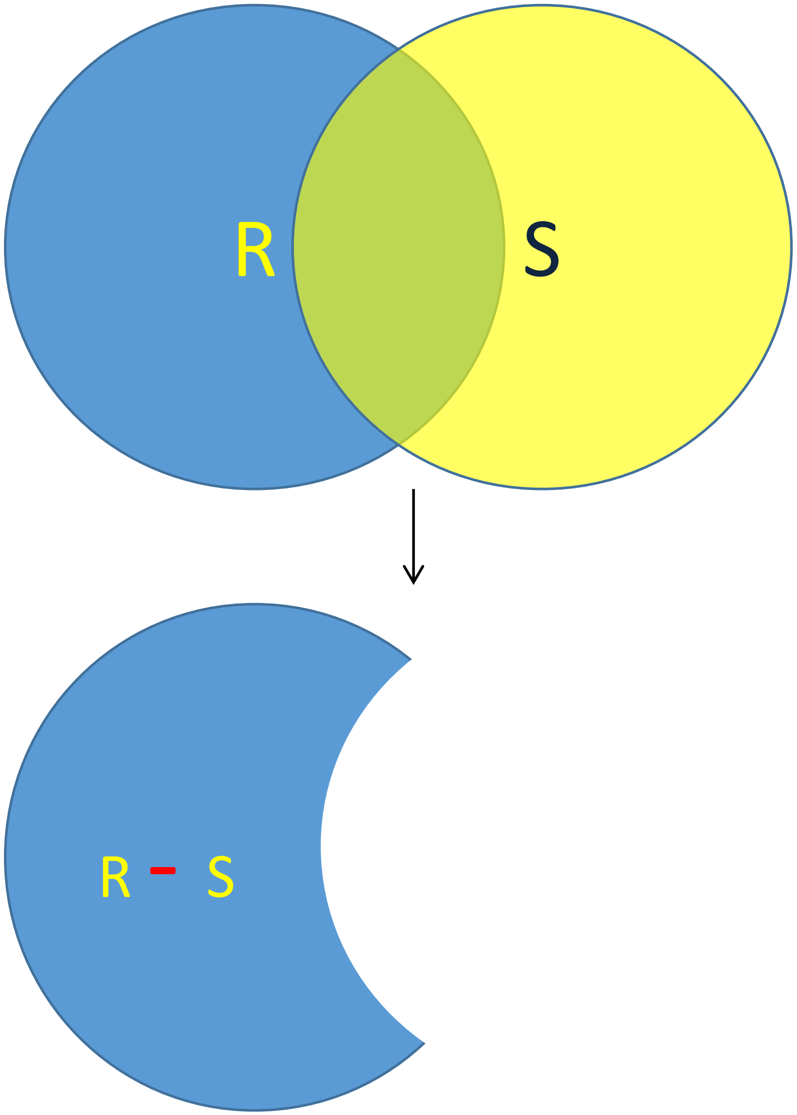
Professor
Name
Dr. Monk
Dr. Pooh
Dr. Patel

$$T = \pi_{name}(Student) \cup \pi_{name}(Professor)$$

Note: if attributes in 2 operands have different names, the names of the left relation are used in the union (PostgreSQL)

Difference

R – S



Difference example.

Query: Who is registered in the Database course but not in the Algorithms?

RegisteredFor	
Name	Topic
Bob	Algorithms
John	Algorithms
Tom	Algorithms
Bob	Python
Tom	Python
Bob	Databases
John	Databases
Maria	Databases
John	GUI
Maria	GUI

First do some selections:

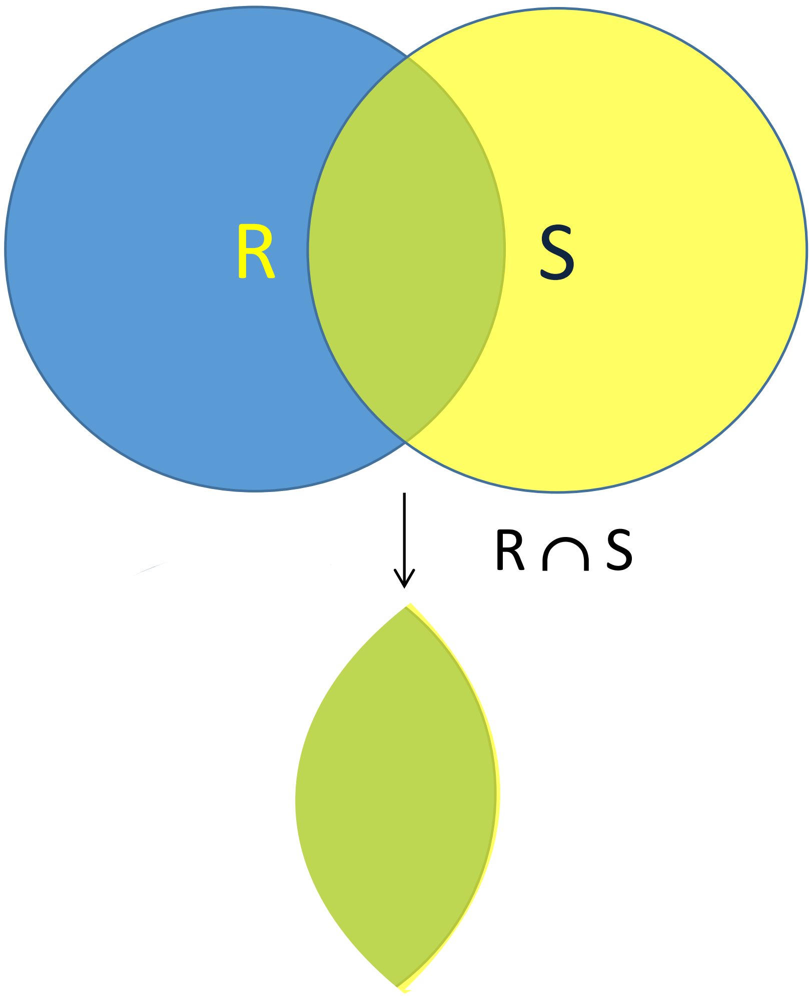
$A = \sigma_{topic=algorithms}$ (RegisteredFor)

$D = \sigma_{topic=databases}$ (RegisteredFor)

Then take $D - A$

Intersection

T=R ∩ S



Intersection example.

Query: Which courses are taught at both Universities?

Alright University

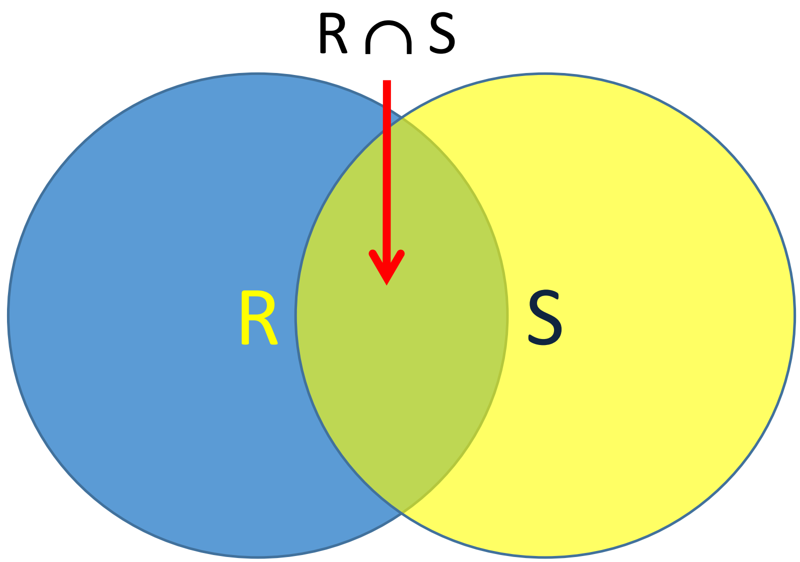
Course
Topic
Algorithms
Python
Databases
GUI

EvenBetter University

Course
Topic
Algorithms
Java
Databases
Networks
Human-Computer Interaction

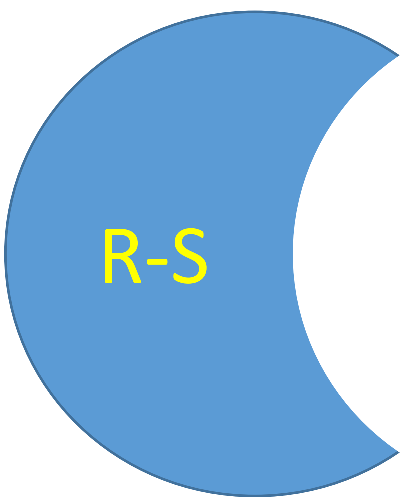
$T = \pi_{topic}(A.course) \cap \pi_{topic}(B.course)$

Intersection is a shortcut for R – (R – S)

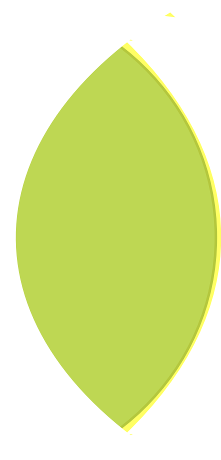


R ∩ S can be derived using 2 difference operators

R – (R – S)



R - S (are in R but not in S)



R - (R-S)

Renaming Operator

$\rho_{S(A1,A2,\ldots,An)}$ **(R)**

1. Resulting relation has exactly the same tuples as **R**, but the name of the relation is **S**.
2. Moreover, the attributes of the result relation **S** can be re-named A_1, A_2, …, A_n, in order from the left.
3. If not all attributes are renamed, can specify renamed attributes:

$\rho_{S,\ a \rightarrow a1,\ b \rightarrow b1}$ **(R)**

Renaming: example

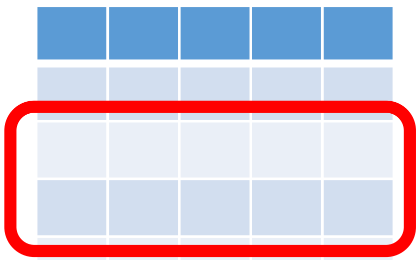
T (uid1, uid2)

A → B
B → A
B → C
A → C
C → B

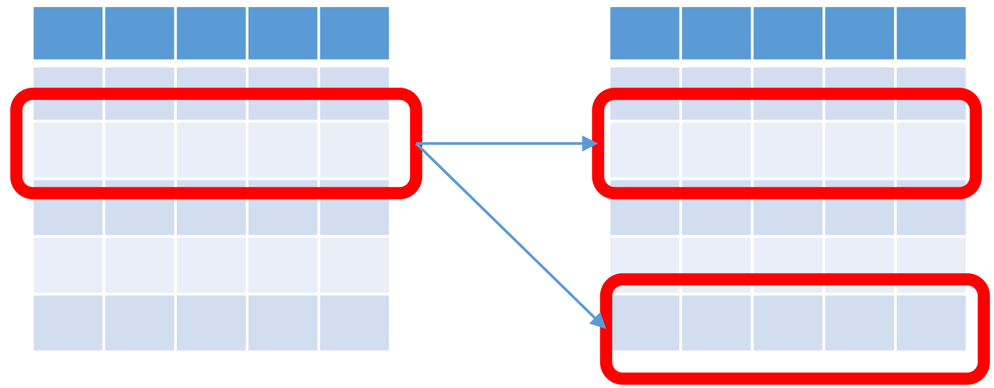
- Find all true friends in twitter dataset
- By renaming T we created two identical relations R and S, and we now extract all tuples where for each pair X → Y in R there is a pair Y → X in S

$$\pi_{R.uid1,\ R.uid2}\ \sigma_{R.uid1=S.uid2\ AND\ R.uid2\ =\ S.uid1}(\rho_R\ (T)\ x\ \rho_S\ (T))$$

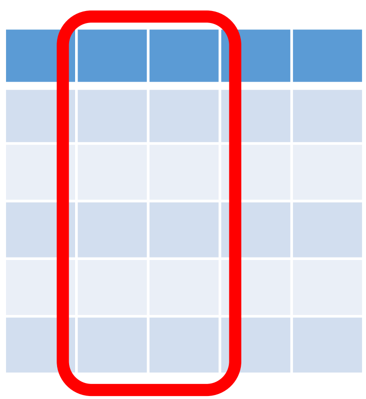
Core operators of relational algebra



Selection σ



Cross-product x



Projection π

Union U
Difference –
Renaming ρ

Core operators – sufficient to express any query in relational model

Edgar "Ted" Codd, a mathematician at IBM in 1970, proved that any query can be expressed using these core operators:
σ, π, x, U, –, ρ

> A Relational Model of Data for Large Shared Data Banks". *Communications of the ACM* **13** (6): 377–387

The Relational model is **precise**, **implementable**, and we can operate on it (combine, optimize)

Relational algebra: closure

In regular algebra the result of every operator is another number, and we can compose complex expressions using basic operators +,-,x,/:

$$a^2 - b^2 = (a-b)x(a+b)$$

The same applies to relational algebra: any RA operator returns a relation, so we can compose complex queries by operating on these intermediate results:

$$\pi_{name,gpa}(\sigma_{gpa>3.5}(Student))$$

$$\Updownarrow$$

Are these logically equivalent?

$$\sigma_{gpa>3.5}(\pi_{name,gpa}(Student))$$

Relational algebra equivalences

- Commutative: $R \bowtie S = S \bowtie R$
- Associative: $(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$
- Splitting: $\sigma_{C \cap D} (R) = \sigma_C (\sigma_D (R))$
- Pushing selections: $\sigma_C (R \bowtie_D S) = \sigma_C (R) \bowtie_D (S)$, if condition C applies only to R
- ...

Example of a valid RA transformation

- Consider ***R*(*A*,*B*)** and ***S*(*B*,*C*)** and the expression below:

  $\boldsymbol{\sigma}_{A=1 \cap B<C}(R \bowtie S)$

1. Splitting **AND** $\boldsymbol{\sigma}_{A=1}(\boldsymbol{\sigma}_{B<C}(R \bowtie S))$
2. Push **σ** to *S* $\boldsymbol{\sigma}_{A=1}(R \bowtie \boldsymbol{\sigma}_{B<C}(S))$
3. Push **σ** to *R* $\boldsymbol{\sigma}_{A=1}(R) \bowtie \boldsymbol{\sigma}_{B<C}(S)$

Intermediate variables

As in traditional algebra,

$x^2 + 2x + 1 = 0$

$D = 4 – 4 = 0$

$x = -2 \underline{+} \sqrt{D} = -2$

we can use *temporary variables* to store the results of intermediate queries. These temporary variables hold results of what is called a ***subquery***

$T_1 = \sigma_{A=1}(R)$

$T_2 = \sigma_{B < C}(S)$

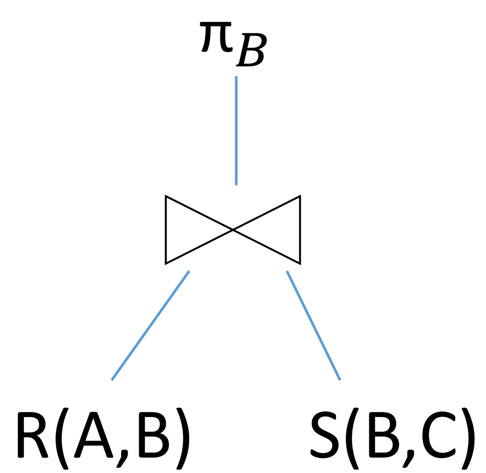
$Result = T_1 \bowtie T_2$

We can visualize an RA expression as a tree

Linear notation

$$\pi_B(R(A,B) \bowtie S(B,C))$$

Tree notation



Bottom-up tree traversal = order of operation execution!

PCRS notation for RA exercises

$$\pi_B(R(A,B) \bowtie S(B,C))$$

```
\project_{B} (R \natural_join S);
```

- To write more complex queries – you would need to learn and exercise a special PCRS syntax
- However, this syntax does not have any further use except of making the marking of your homework easier
- Thus, we will do PCRS exercises only for real SQL

Why do we care about relational algebra?

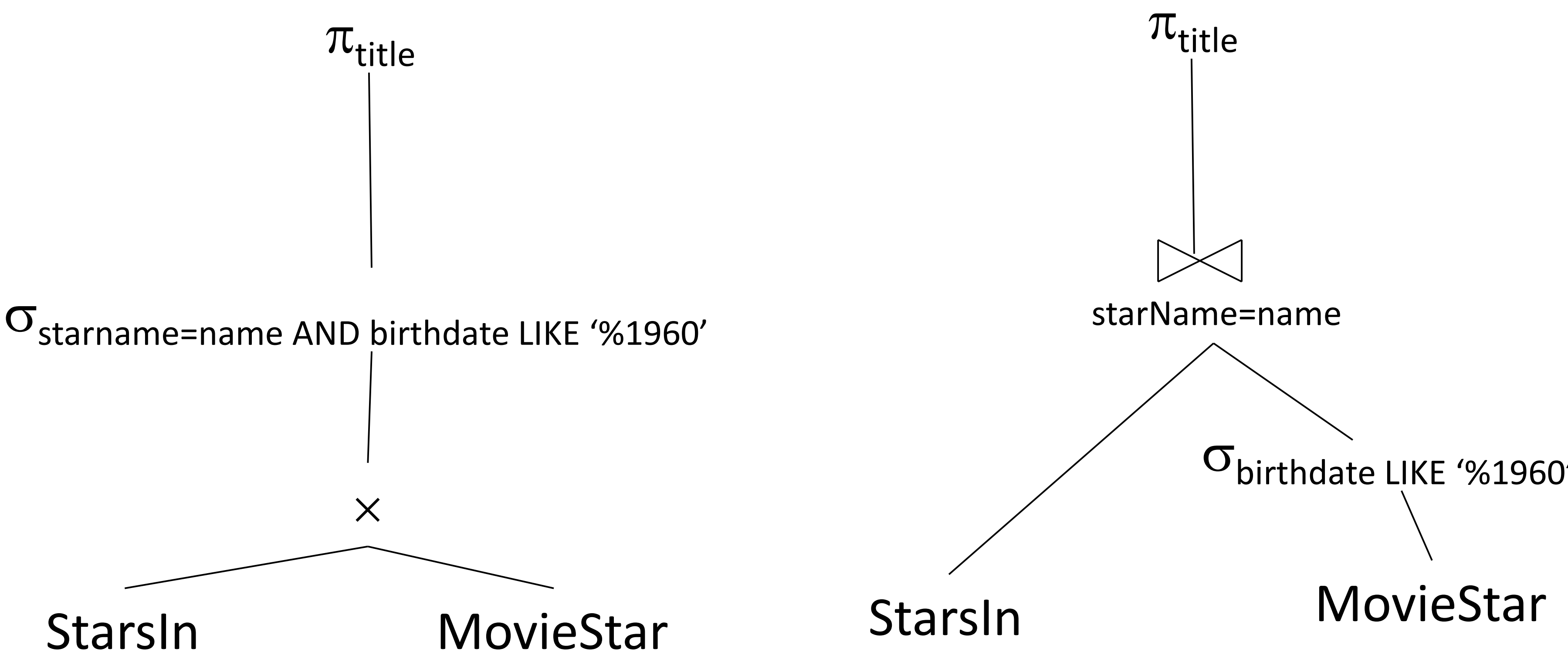
Why not learn just SQL? 

SQL is a query language that implements Relational Algebra

Why not learn how to solve quadratic equations looking only at a java implementation?

```
double discriminant = Math.pow(b,2) - 4*a*c;
double x1 = (-b + Math.sqrt(discriminant))/(2*a);
double x2 = (-b - Math.sqrt(discriminant))/(2*a);
double i=Math.sqrt(-1);
double x3 = (-b + (Math.sqrt(discriminant))*i)/(2*a);
double x4 = (-b + (Math.sqrt(discriminatn))*i)/(2*a);
if (discriminat > 0 ){
    System.out.println("there are two solutions:" +x1+"and"+x2);
}
```

RA is a basis for logical query optimization



Which query is more efficient?

Extended operators of Relational Algebra

can be derived from core operators

Outer join

Motivation

- Suppose we join $R \bowtie S$.
- A tuple of R which doesn't join with any tuple of S is said to be *dangling*.
  - Similarly for a tuple of S.
  - **Problem**: We loose dangling tuples.

Outerjoin

- Preserves dangling tuples by padding them with a special **NULL** symbol in the result.

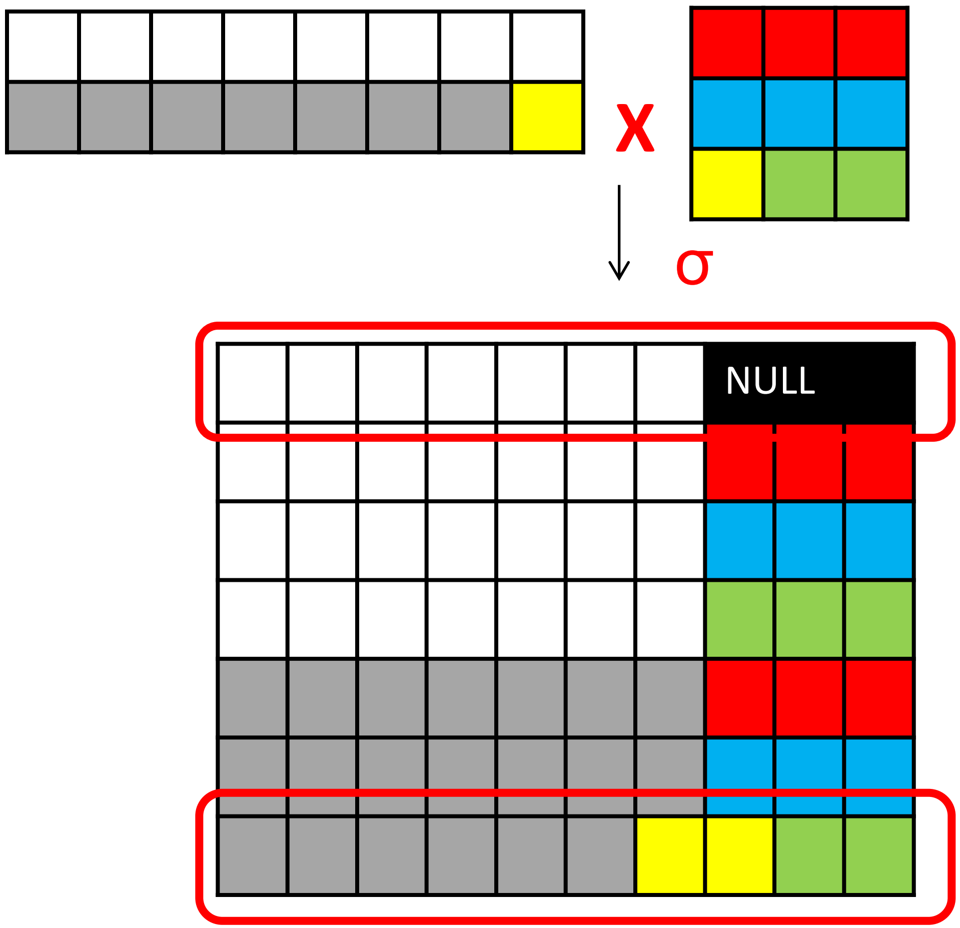
Types of outer join

- R ⟗$_C$ S – This is the **full outerjoin**: Pad dangling tuples from both tables.
- R ⟕$_C$ S – **left outerjoin**: Only pad dangling tuples from the left table.
- R ⟖$_C$ S – **right outerjoin**: Only pad dangling tuples from the right table.

Left outer join

1. For each tuple in R, include all tuples in S which satisfy join condition, but include also tuples of R that do not have matches in S

2. For this case, pair tuples of R with NULL



T= R ⟕$_{condition}$ S

Outer join: example

Anonymous patient P

age	zip	disease
54	99999	heart
20	44444	flue
33	66666	lung

Anonymous occupation O

age	zip	job
54	99999	lawyer
20	44444	cashier

T= P ⟕ O

age	zip	disease	job
54	99999	heart	lawyer
20	44444	flue	cashier
33	66666	lung	**NULL**

Expressing constraints in Relational Algebra

Relational algebra as a constraint language

- If R is a query in relational algebra, then $R=\emptyset$ is a constraint – no results of such query should exist

- If R and S are expressions of relational algebra, then $R \subseteq S$ is a constraint that says that every tuple in the result of R should be also in S (R-results are a subset of S-results)

Expressing foreign key constraints

- If we expect that every value of attribute A in R appears as attribute B in S (R(A) is a foreign key referencing S(B)), then we express this in RA as follows:

$$\pi_A(R) \subseteq \pi_B(S)$$

- Or a shortcut:

$$R[A] \subseteq S[B]$$

Example: movies

MovieStar (<u>name</u>, address, gender, birthdate)

StarsIn (<u>movieTitle</u>, <u>movieYear</u>, <u>starName</u>)

FK constraint in RA notation:

$\pi_{starName}(\text{StarsIn}) \subseteq \pi_{name}(\text{MovieStar})$

Expressing primary key constraints

- If A is a primary key of relation T, and B is any other non-key attribute, then:

$$\sigma_{R.A=S.A \text{ AND } R.B \neq S.B}(\rho_R (T) \times \rho_S (T)) = \emptyset$$

- This expresses an idea that if we pair all tuples of relation T with itself, there could not be 2 tuples that agree on A but disagree on B.
- Value of A completely identifies all other attributes, A is a primary key for B

Expressing value constraints

Examples of domain constraints for Movies:

- The only permitted values of gender are 'F' or 'M'

$\sigma_{gender \neq 'F' \text{ AND } gender \neq 'M'}(\text{movieStar}) = \emptyset$

- The length of a movie cannot be less than 60 nor more than 250

$\sigma_{length<60 \text{ OR } length>250}(\text{movie}) = \emptyset$

Estimating size of resulting relations

Size estimation examples 1

Given relation R with N tuples and relation S with M tuples, what is the maximum and minimum size of the output to the following queries:

$\sigma_c (R)$

- Min: 0 (no tuples satisfy the condition)
- Max: N

$\pi_A (R)$

- Min: 1
- Max: N

What if A is a key?

- Min: N
- Max: N

Size estimation examples 2

Given relation R (A,B) with N tuples and relation S(B,C) with M tuples, tell what is the maximum and minimum size of the output to the following queries

R x S

- Min: NM
- Max: NM

R ⋈ S

- Min: 0 (no tuples to join)
- Max: NM (all tuples of S join with all tuples of R on their common attribute – equal values of B in both relations)

Sample test question

If I have a relation R with 100 tuples and a relation S with exactly 1 tuple, how many tuples will be in the result of

$R \ltimes S$?

A. At least 100, but could be more
B. Could be any number between 0 and 100 inclusive
C. 0
D. 1
E. 100

RA exercises

Tutorial

Running example: Movies database

Movie (title, year, length, inColor, studioName, producerC)

MovieStar (name, address, gender, birthdate)

StarsIn (movieTitle, movieYear, starName)

MovieExec (name, address, cert, netWorth)

Studio (studioname, presc);

Simple queries

1. Find producer of ‘Star wars’

2. Title and length of all Disney movies produced in year 1990

3. For each movie’s title produce the name of this movie’s producer

Movie (<u>title</u>, <u>year</u>, length, inColor, studioName, producerC)

MovieStar (<u>name</u>, address, gender, birthdate)

StarsIn (<u>movieTitle</u>, <u>movieYear</u>, <u>starName</u>)

MovieExec (<u>name</u>, address, cert, netWorth)

Studio (<u>studioname</u>, presc);

4. Find all name pairs in form (movie star, movie producer) that live at the same address.

$Star = \rho_{star,staraddress}(\pi_{name,\ address}(MovieStar))$

$Prod = \rho_{prod,\ prodaddress}(\pi_{name,\ address}(MovieExec))$

$\pi_{star,prod}((Star) \bowtie_{staraddress=prodaddress\ AND\ star\ !=prod}(Prod))$

MORE COMPLEX QUERIES

Movies

Movie (title, year, length, inColor, studioName, producerC)

MovieStar (name, address, gender, birthdate)

StarsIn (movieTitle, movieYear, starName)

MovieExec (name, address, cert, netWorth)

Studio (studioname, presc);

5. Find the names of all producers who did NOT produce 'Star wars'

- Simple:

$\pi_{name}(\text{MovieExec}) -$

$\pi_{name}((\text{Movie}) \bowtie_{title='Star\ wars'\ AND\ producerC=cert} (\text{MovieExec}))$

- More efficient (smaller Cartesian product)

$\pi_{name}((\sigma_{title='Star\ wars'}(\text{Movie})) \bowtie_{producerC!=cert} (\text{MovieExec}))$

6. Find all name pairs in form (movie star, movie producer) that live at the same address. The same person can be both a star and a producer. Now, try to eliminate palindrome pairs: leave (a,b) but not both (a,b) and (b,a).

6 – solution 1. Find all name pairs in form (movie star, movie producer) that live at the same address. The same person can be both a star and the producer. Now, try to eliminate palindrome pairs: leave (a,b) but not both (a,b) and (b,a).

1. $Star = \rho_{name \rightarrow star}(MovieStar)$

   $Prod = \rho_{name \rightarrow prod}(MovieExec)$

2. $Pairs = \pi_{star,prod}((Star) \bowtie_{Star.address=Prod.address \text{ AND } star!=prod} (Prod))$

3. $PA = \sigma_{star<prod}(Pairs)$ // Pairs in Ascending order

   $PD = \sigma_{star>prod}(Pairs)$ //Pairs in Descending order

4. $Palindrome = (PA) \bowtie_{PA.star=PD.prod \text{ AND } PA.prod=PD.star} (PD)$

5. $Pairs - \pi_{PD.star,PD.prod} (Palindrome)$

Example on the next page

Step 1. Renaming

Star	
star	addr
A	1
B	1
C	2
F	3

Prod	
prod	addr
A	1
B	1
D	2
E	3

1
Star=$\rho_{name \rightarrow star}$(MovieStar)
Prod=$\rho_{name \rightarrow prod}$(MovieExec)

Step 2. Cartesian product: Star x Prod

Star	Addr	Prod	Addr
A	1	A	1
A	1	B	1
A	1	D	2
A	1	E	3
B	1	A	1
B	1	B	1
B	1	D	2
B	1	E	3
C	2	A	1
C	2	B	1
C	2	D	2
C	2	E	3
F	3	A	1
F	3	B	1
F	3	D	2
F	3	E	3

2. Pairs = $\pi_{star,prod}$ ((Star) $\bowtie_{Star.address=Prod.address\ AND\ star!=prod}$ (Prod))

Pairs	
Star	Prod
A	B
B	A
C	D
F	E

Step 3. Sorted pairs

Pairs	
Star	Prod
A	B
B	A
C	D
F	E

3. PA = $\sigma_{star<prod}$(Pairs) // Pairs where Star < Prod
   PD = $\sigma_{star>prod}$(Pairs) //Pairs where Star > Prod

PA	
Star	Prod
A	B
C	D

PD	
Star	Prod
B	A
F	E

Step 4. Cartesian product PA x PD

PA	
Star	Prod
A	B
C	D

x

PD	
Star	Prod
B	A
F	E

Palyndrome (only colored tuple qualify)			
PA.Star	PA.Prod	PD.Star	PD.Prod
A	B	B	A
A	B	F	E
C	D	B	A
C	D	F	E

4. Palindrome = (PA) $\bowtie_{PA.star=PD.prod \text{ AND } PA.prod=PD.star}$ (PD)

Step 5. Remove palindrome tuples from pairs

5. Pairs – $\pi_{PD.star,PD.prod}$ (Palindrome)

Pairs	
Star	Prod
A	B
B	A
C	D
F	E

-

Palyndrome			
PA.Star	PA.Prod	PD.Star	PD.Prod
A	B	B	A

result	
Star	Prod
A	B
C	D
F	E

6. Another solution proposed in class

$Star = \rho_{name \rightarrow star}(MovieStar)$
$Prod = \rho_{name \rightarrow prod}(MovieExec)$

$SP = Star \bowtie_{Star.address=Prod.address \text{ AND } star!=prod} Prod$
$PS = Prod \bowtie_{Star.address=Prod.address \text{ AND } star!=prod} Star$

$$PAIRS = \rho_{star \rightarrow name1,\ prod \rightarrow name2}(SP) \cup \rho_{prod \rightarrow name1,\ star \rightarrow name2}(PS)$$

$Result = \sigma_{name1 < name2}(Pairs)$

Example on the next page

Step 1. Renaming

The renaming is done for readability - to distinguish names:

MovieStar.name → Star.star

MovieExec.name → Prod.prod

Star	
star	addr
A	1
B	1
C	2
F	3

Prod	
prod	addr
A	1
B	1
D	2
E	3

$$Star=\rho_{name \rightarrow star}(MovieStar)$$

$$Prod=\rho_{name \rightarrow prod}(MovieExec)$$

Step 2. Join Star ⋈ Prod and Prod ⋈ Star on address

$SP = Star \bowtie_{Star.address=Prod.address \text{ AND } star!=prod} Prod$

$PS = Prod \bowtie_{Star.address=Prod.address \text{ AND } star!=prod} Star$

Star	
star	addr
A	1
B	1
C	2
F	3

Prod	
prod	addr
A	1
B	1
D	2
E	3

→

SP	
star	prod
A	B
B	A
C	D
F	E

PS	
prod	star
A	B
B	A
D	C
E	F

Step 3. Union (set union) of SP and PS

PAIRS = $\rho_{\text{star} \rightarrow \text{name1, prod} \rightarrow \text{name2}}$ (SP) U $\rho_{\text{prod} \rightarrow \text{name1, star} \rightarrow \text{name2}}$ (PS)

SP	
star	prod
A	B
B	A
C	D
F	E

PS	
prod	star
A	B
B	A
D	C
E	F

PAIRS	
name1	name 2
A	B
B	A
C	D
F	E
D	C
E	F

Step 4. Select only one instance of palindrome pair – where name1<name2

Result = $\sigma_{\text{name1} < \text{name2}}$ (Pairs)

PAIRS	
name1	name 2
A	B
B	A
C	D
F	E
D	C
E	F

Result	
name1	name 2
A	B
C	D
E	F

We don't know at this point who is a star and who is a producer, but we can later do the selection for each name in both tables to figure out if this is important for our query

Movie (<u>title</u>, <u>year</u>, length, inColor, studioName, producerC)

MovieStar (<u>name</u>, address, gender, birthdate)

StarsIn (<u>movieTitle</u>, <u>movieYear</u>, <u>starName</u>)

MovieExec (<u>name</u>, address, cert, netWorth)

Studio (<u>studioname</u>, presc);

7. Find names of producers that produced at least one movie for each of different studios: Disney and MGM

$\pi_{name}[(\sigma_{studioName='Disney'}(Movie)) \bowtie_{producerC=cert}(MovieExec)]$

$\wedge$

$\pi_{name}[(\sigma_{studioName='MGM'}(Movie)) \bowtie_{producerC=cert}(MovieExec)]$

Movie (title, year, length, inColor, studioName, producerC)

MovieStar (name, address, gender, birthdate)

StarsIn (movieTitle, movieYear, starName)

MovieExec (name, address, cert, netWorth)

Studio (studioname, presc);

8. Find all movie titles for which there is no producer entry in MovieExec table

$\pi_{title}(\text{Movie}) - \pi_{title}((\text{Movie}) \bowtie_{producerC=cert} (\text{MovieExec}))$

Movie (title, year, length, inColor, studioName, producerC)
MovieStar (name, address, gender, birthdate)
StarsIn (movieTitle, movieYear, starName)
MovieExec (name, address, cert, netWorth)
Studio (studioname, presc);

9. Find the names of all stars which starred in at least 2 movies (according to our database)

1. $S1=\rho_{title1,year1,name1}(StarsIn)$

   $S2=\rho_{title2,year2,name2}(StarsIn)$

2. $(S1) \bowtie_{name1=name2 \text{ AND } (title1 \text{ != } title2 \text{ or } year1\text{!=}year2)} (S2)$

Relational algebra for bags – basis for SQL

$\dot{\bowtie}$ $\dot{\bowtie}_R$ $\dot{\bowtie}$

Relational Algebra on Bags

- A **bag** is like a set, but an element may appear more than once.
  - *Multiset* is another name for "bag."
- Example:
  - {1,2,1,3} is a bag.
  - {1,2,3} is also a bag that happens to be a set.
- Bags also resemble lists, but order in a bag is unimportant.
  - Example:
    - {1,2,1} = {1,1,2} as bags, but
    - [1,2,1] != [1,1,2] as lists.

Why bags?

- SQL is actually a bag language.
- SQL will eliminate duplicates, but usually only if you ask it to do so explicitly.

- Some operations, like **projection** or **union**, are much more efficient on bags than sets.
  - Why?

Operations on Bags

- **Selection** applies to each tuple, so its effect on bags is like its effect on sets.

- **Projection** also applies to each tuple, but as a bag operator, we do not eliminate duplicates.

- **Products** and **joins** are done on each pair of tuples, so duplicates in bags have no effect on how we operate.

Example: Bag Selection

R(A B)

A	B
1	2
5	6
1	2

S(B C)

B	C
3	4
7	8

$\sigma_{A+B<5}$ (R) =

A	B
1	2
1	2

Example: Bag Projection

R(A	B)
1	2
5	6
1	2

S(B	C)
3	4
7	8

π_A (R) =

A
1
5
1

Bag projection yields always the same number of tuples as the original relation.

Example: Bag Product

R(A B)

A	B
1	2
5	6
1	2

S(B C)

B	C
3	4
7	8

$R \times S =$

A	R.B	S.B	C
1	2	3	4
1	2	7	8
5	6	3	4
5	6	7	8
1	2	3	4
1	2	7	8

- *Each copy* of the tuple **(1,2)** of **R** is being paired with each tuple of **S**.
- So, the duplicates do not have an effect on the way we compute the product.

Bag Union

- **Union**, **intersection**, and **difference** need new definitions for bags.

- An element appears in the **union** of two bags the **sum** of the number of times it appears in each bag.

- Example:

  $\{1,2,1\} \cup \{1,1,2,3,1\}$

  $= \{1,1,1,1,1,2,2,3\}$

Bag Intersection

- An element appears in the **intersection** of two bags the **minimum** of the number of times it appears in either.

- Example:

  $\{1,2,1\} \cap \{1,2,3\}$

  $= \{1,2\}$.

Bag Difference

- An element appears in **difference** $A - B$ of bags as many times as it appears in A, **minus** the number of times it appears in B.
  - But never less than 0 times.

- Example: {1,2,1} – {1,2,3}
  = {1}.

Beware: Bag Laws != Set Laws

Not all algebraic laws that hold for sets also hold for bags.

Example

- Set union is *idempotent*, meaning that

  $\boldsymbol{S \cup S = S}$.

- However, for bags, if $\boldsymbol{x}$ appears $\boldsymbol{n}$ times in $\boldsymbol{S}$, then it appears $2n$ times in $S \cup S$.

- Thus $\boldsymbol{S \cup S}$ **!=** $\boldsymbol{S}$ in general.

The Extended Algebra (for bags)

1. δ: eliminate duplicates from bags.
2. τ: sort tuples.
3. γ: grouping and aggregation.

Example: Duplicate Elimination

$R_1 := \delta(R_2)$

R_1 consists of one copy of each tuple that appears in R_2 one or more times.

R =

A	B
1	2
3	4
1	2

δ(R) =

A	B
1	2
3	4

Sorting

$R_1 := \tau_L (R_2)$

- *L* is a list of some of the attributes of R_2.

R_1 is the list of tuples of R_2 sorted first on the value of the first attribute on *L*, then on the second attribute of *L*, and so on.

Aggregation Operators AGG

- They apply to entire columns of a table and produce a single result.

- The most important examples:
  - SUM
  - AVG
  - COUNT
  - MIN
  - MAX

Example: Aggregation

R =

A	B
1	3
3	4
3	2

SUM(A) = 7
COUNT(A) = 3
MAX(B) = 4
MIN(B) = 2
AVG(B) = 3

Grouping Operator

$R_1 := \gamma_L (R_2)$

L is a list of elements that are either:

1. Individual (*grouping*) attributes.
2. AGG(*A*), where AGG is one of the aggregation operators and *A* is an attribute.

Example: Grouping/Aggregation

R =

A	B	C
1	2	3
4	5	6
1	2	5

$\gamma_{A,B,\text{AVG}(C)}$ (R) = ??

First, group *R* :

A	B	C
1	**2**	3
1	**2**	5
4	**5**	6

Then, average *C* within groups:

A	B	AVG(C)
1	2	4
4	5	6

γ_L(R) - Formally

- Group *R* according to all the grouping attributes on list *L*.
  - That is, form one group **for each distinct list** of values for those attributes in ***R***.

- Within each group, compute AGG(*A*) for each aggregation on list *L*.

- Result has grouping attributes and aggregations as attributes: One tuple for each list of values for the grouping attributes **and** their group's aggregations.