



# COMP9311: Database Systems

## Relational Algebra

(textbook: chapter 8)

**Term 3 2022**

**Week 3 Relational Algebra and SQL**

**By Helen Paik, CSE UNSW**

**Disclaimer: the course materials are sourced from**

- previous offerings of COMP9311 and COMP3311
- Prof. Werner Nutt on Introduction to Database Systems (<http://www.inf.unibz.it/~nutt/Teaching/IDBs1011/>)

# Motivation

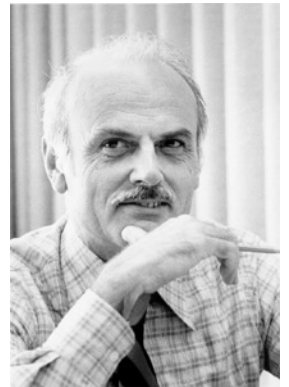
We know how to store data ... i.e., we modelled our data in relational data model -> then created tables to store them into a relational database.

How do we manipulate or retrieve (interesting) the data?

A data model must include a set of operations to manipulate the database, in addition to the concepts for defining database's structure and constraints.

The basic set of operations for the relational model is ***Relational Algebra***

- Edgar F. Codd (1970): Relational Algebra, mathematical foundation for relational data management
- supports basic retrieval requests (queries) -> the result of a query is also a ***relation***
- A sequence of relational algebra operations form a relational algebra expression -> results in a ***relation***



# Motivation

The relational algebra is very important for several reasons.

- First, it provides a formal foundation for relational model operations.
- Second, and perhaps more important, it is used as a basis for implementing and optimizing queries in the query processing and optimization modules that are integral parts of relational database management systems (RDBMSs),
- Third, some of its concepts are incorporated into the SQL, the standard query language for relational database management systems

# Characteristics of an Algebra

An algebra expression:

- is constructed with operators from atomic operands (constants, variables, ....)
- can be evaluated
- can be equivalent to another expression
  - ...if they return the same result for all values of the variables

This equivalence concept gives rise to an algebraic identity between expressions

An **algebraic identity** is an equality that holds for any values of its variables.

The value of an expression is independent of its context

- e.g.,  $5 + 3$  has the same value, no matter whether it occurs as  
 $10 - (5 + 3)$       or       $4 \cdot (5 + 3)$

*Atomic expressions:*

*numbers and variables*

*Operators:*       $+$ ,  $-$ ,  $\cdot$ ,  $:$

*Identities:*

$$x + y = y + x$$

$$x \cdot (y + z) = x \cdot y + x \cdot z$$

*... and so on*

*Consequence: subexpressions can be replaced by equivalent expressions without changing the meaning of the entire expression*

# Relational Algebra: Principles

Atoms are relations

Operators are defined for arbitrary instances of a relation

The following two results have to be defined for each operator:

- result schema
- result instance

Set theoretic operators

- union “ $\cup$ ”, intersection “ $\cap$ ”, difference “ $\setminus$ ”

Renaming operator  $\rho$

Removal operators

- projection  $\pi$ , selection  $\sigma$

Combination operators

- Cartesian product “ $\times$ ”, joins “ $\bowtie$ ”

Extended operators

- duplicate elimination, grouping, aggregation, sorting, outer joins, etc.

- *“Equivalent” to SQL query language ... Relational Algebra concepts reappear in SQL*
- *Used inside a DBMS, to express query plans*

# Set Operators

Observations:

Instances of relations are sets

→ we can form **unions**, **intersections**, and **differences**

Set algebra operators can only be applied to relations with identical attributes,

- same number of attributes
- same attribute names
- same domains
- (i.e., set operation compatibility)

# Union ( $\cup$ )

CS-Student

Studno	Name	Year
s1	Egger	5
s3	Rossi	4
s4	Maurer	2

Master-Student

Studno	Name	Year
s1	Egger	5
s2	Neri	5
s3	Rossi	4

CS-Student  $\cup$  Master-Student

Studno	Name	Year
s1	Egger	5
s2	Neri	5
s3	Rossi	4
s4	Maurer	2

# Intersection ( $\cap$ )

CS-Student

Studno	Name	Year
s1	Egger	5
s3	Rossi	4
s4	Maurer	2

Master-Student

Studno	Name	Year
s1	Egger	5
s2	Neri	5
s3	Rossi	4

CS-Student  $\cap$  Master-Student

Studno	Name	Year
s1	Egger	5
s3	Rossi	4



# Difference (\)

## CS-Student

Studno	Name	Year
s1	Egger	5
s3	Rossi	4
s4	Maurer	2

## Master-Student

Studno	Name	Year
s1	Egger	5
s2	Neri	5
s3	Rossi	4

## CS-Student \ Master-Student

Studno	Name	Year
s4	Maurer	2

Set difference, formally:

$$B \setminus A = \{x \in B \mid x \notin A\}.$$

# Renaming $\rho$

- The renaming operator  $\rho$  (reads 'rho') **changes the name of** relation schema (both for relation name and relation attributes)
- It changes the **schema**, but **only within a query**
- $\rho_x(E)$  where  $E$  is the relation name and  $x$  is the new name for  $E$ , usually a shorter name
  - $\rho_{FC}(\text{Father-Child})$
- $\rho_{a/b}(E)$  where  $E$  is the relation name,  $a$ ,  $b$  are attribute names,  $b$  is an attribute of  $E$ 
  - $\rho_{\text{parent/father}}(\text{Father-Child})$

Father-Child

Father	Child
Adam	Abel
Adam	Cain
Abraham	Isaac

$\rho_{FC}(\text{Father-Child})$

FC

Father	Child
Adam	Abel
Adam	Cain
Abraham	Isaac

$\rho_{\text{parent/father}}(\text{Father-Child})$

Parent	Child
Adam	Abel
Adam	Cain
Abraham	Isaac

# Union example after renaming

$\rho$  Parent / Father (Father-Child)

Parent	Child
Adam	Abel
Adam	Cain
Abraham	Isaac

$\rho$  Parent / Father (Father-Child)

$\cup$

$\rho$  Parent / Mother (Mother-Child)

$\rho$  Parent / Mother (Mother-Child)

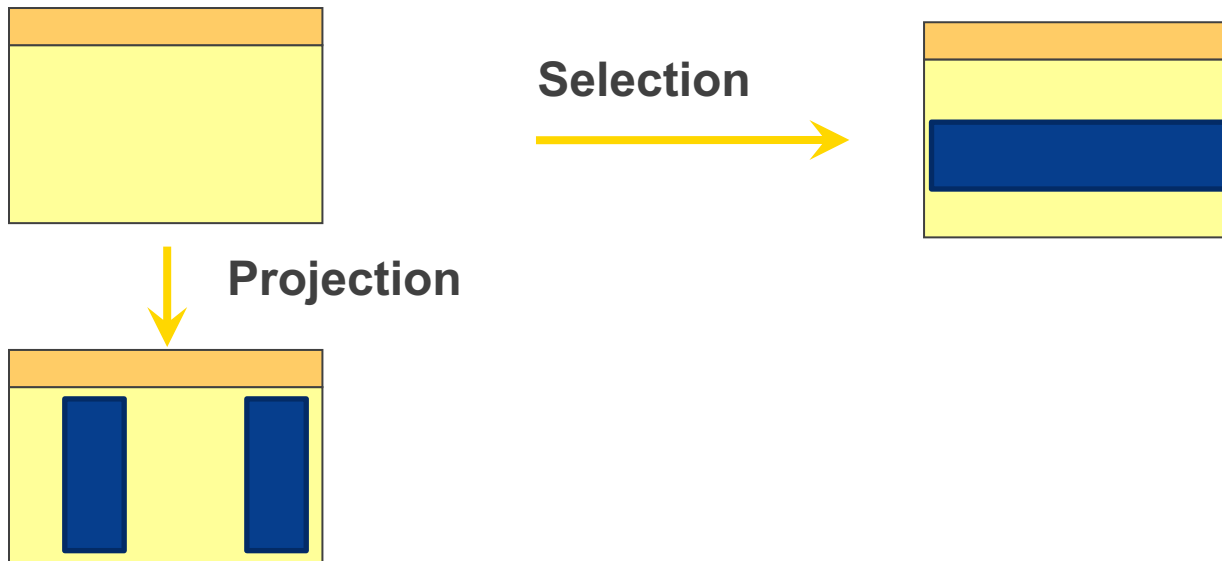
Parent	Child
Eve	Abel
Eve	Seth
Sara	Isaac

Parent	Child
Adam	Abel
Adam	Cain
Abraham	Isaac
Eve	Abel
Eve	Seth
Sara	Isaac

# Projection and Selection

Two “orthogonal” operators

- Selection:
  - horizontal decomposition
- Projection:
  - vertical decomposition



# Projection ( $\pi$ )

General form:  $\pi_{A_1, \dots, A_k}(R)$

where  $R$  is a relation and  $A_1, \dots, A_k$  are attributes of  $R$ .

Result:

- Schema:  $(A_1, \dots, A_k)$
- Instance: the set of all subtuples  $t[A_1, \dots, A_k]$  where  $t \in R$

Intuition: “removes” all attributes that are not in projection list

# Projection: Example

STUDENT

studno	name	hons	tutor	year
s1	jones	ca	bush	2
s2	brown	cis	kahn	2
s3	smith	cs	goble	2
s4	bloggs	ca	goble	1
s5	jones	cs	zobel	1
s6	peters	ca	kahn	3

$$\pi_{\text{tutor}}(\text{STUDENT}) = \begin{array}{|c|} \hline \text{tutor} \\ \hline \text{bush} \\ \text{kahn} \\ \text{goble} \\ \text{zobel} \\ \hline \end{array}$$

*Note:*

- *result relations don't have a name*
- *If duplicates?*

# Selection ( $\sigma$ )

General form:  $\sigma_C(R)$

with a relation  $R$  and a condition  $C$  on the attributes of  $R$ .

Result:

- Schema: the schema of  $R$
- Instance: the set of all  $t \in R$  that satisfy  $C$

Intuition: Filters out all tuples that do not satisfy  $C$

# Selection: Example

STUDENT

<u>studno</u>	name	hons	tutor	year
s1	jones	ca	bush	2
s2	brown	cis	kahn	2
s3	smith	cs	goble	2
s4	bloggs	ca	goble	1
s5	jones	cs	zobel	1
s6	peters	ca	kahn	3

STUDENT

$\sigma_{\text{name='bloggs'}}(\text{STUDENT})$  =

studno	name	hons	tut or	year
s4	bloggs	ca	goble	1

*Note:*

- *result relation has a name*



# Selection Conditions

## Elementary conditions:

`<attr> op <val> or <attr> op <attr> or <expr> op <expr>`

where op is "=", "<", "≤", (on numbers and strings)  
"LIKE" (for string comparisons),...

## Example:

- `age ≤ 24`
- `phone LIKE '0039%'`
- `salary + commission ≥ 24000`

## Combined conditions (using Boolean connectives):

`C1 and C2 or C1 or C2 or not C`

# Selection conditions

STUDENT

<u>studno</u>	name	hons	tutor	year
s1	jones	ca	bush	2
s2	brown	cis	kahn	2
s3	smith	cs	goble	2
s4	bloggs	ca	goble	1
s5	jones	cs	zobel	1
s6	peters	ca	kahn	3

$\sigma_{((\text{hons}='cs') \text{ or } (\text{hons} = 'ca')) \text{ and } (\text{tutor}='goble') \text{ (STUDENT)}}$  =

STUDENT

studno	name	hons	tut or	year
s3	smiths	cs	goble	2
s4	bloggs	ca	goble	1

# Operators Can Be Nested

Who is the tutor of the student named “Bloggs”?

STUDENT

<u>studno</u>	name	hons	tutor	year
s1	jones	ca	bush	2
s2	brown	cis	kahn	2
s3	smith	cs	goble	2
s4	bloggs	ca	goble	1
s5	jones	cs	zobel	1
s6	peters	ca	kahn	3

$\pi_{\text{tutor}} ( \sigma_{\text{name='bloggs'}} (\text{STUDENT}) )$

=

tutor
goble

# Identities for Selection and Projection

For all conditions C1, C2, more generally predicates p,q and relation R we have:

Selection splitting:

$$\sigma_{C1 \text{ and } C2}(R) = \sigma_{C1}(\sigma_{C2}(R))$$

Also, selection is commutative:

$$\sigma_{C1}(\sigma_{C2}(R)) = \sigma_{C2}(\sigma_{C1}(R))$$

What about these – commutativity of selection and projection

$$\pi_{A1, \dots, Am}(\sigma_C(R)) = \sigma_C(\pi_{A1, \dots, Am}(R))$$

# Selection Conditions and “NULL”

Does the following identity hold?

$$\text{Student} = \sigma_{\text{year} \leq 3}(\text{Student}) \cup \sigma_{\text{year} > 3}(\text{Student}) ?$$

What if Student contains a tuple  $t$  with  $t[\text{year}] = \text{null}$  ?

Convention: Only comparisons with non-null values are TRUE or FALSE. Comparisons involving null yield a value UNKNOWN. To test, whether a value is null or not null, there are two conditions:

$\langle \text{attr} \rangle \text{ IS NULL}$  or  $\langle \text{attr} \rangle \text{ IS NOT NULL}$

Thus, the following identities hold:

$$\begin{aligned} \text{Student} &= \sigma_{\text{year} \leq 3}(\text{Student}) \cup \sigma_{\text{year} > 3}(\text{Student}) \cup \sigma_{\text{year IS NULL}}(\text{Student}) \\ &= \sigma_{\text{year} \leq 3 \text{ OR } \text{year} > 3 \text{ OR } \text{year IS NULL}}(\text{Student}) \end{aligned}$$

# Cartesian Product (X)

General form:

where R and S are arbitrary relations  $R \times S$

Result:

- Schema:  $(A_1, \dots, A_m, B_1, \dots, B_n)$ , where  $(A_1, \dots, A_m)$  is the schema of R and  $(B_1, \dots, B_n)$  is the schema of S.

*(If A is an attribute of both, R and S, then  $R \times S$  contains the disambiguated attributes R.A and S.A.)*

- Instance: the set of all concatenated tuples  $(t, s)$  where  $t \in R$  and  $s \in S$

# Cartesian Product: Student × Staff

## STUDENT

studno	name	hons	tutor	year
s1	jones	ca	bush	2
s2	brown	cis	kahn	2
s3	smith	cs	goble	2
s4	bloggs	ca	goble	1
s5	jones	cs	zobel	1
s6	peters	ca	kahn	3

## STAFF

lecturer	roomno
kahn	IT206
bush	2.26
goble	2.82
zobel	2.34
watson	IT212
woods	IT204
capon	A14
lindsey	2.10
barringer	2.125

*Brings all information  
from relations into one  
without applying any  
conditions*

*What's the point  
of this?*

studno	name	hons	tutor	year	lecturer	roomno
s1	jones	ca	bush	2	kahn	IT206
s1	jones	ca	bush	2	bush	2.26
s1	jones	ca	bush	2	goble	2.82
s1	jones	ca	bush	2	zobel	2.34
s1	jones	ca	bush	2	watson	IT212
s1	jones	ca	bush	2	woods	IT204
s1	jones	ca	bush	2	capon	A14
s1	jones	ca	bush	2	lindsey	2.10
s1	jones	ca	bush	2	barringer	2.125
s2	brown	cis	kahn	2	kahn	IT206
s2	brown	cis	kahn	2	bush	2.26
s2	brown	cis	kahn	2	goble	2.82
s2	brown	cis	kahn	2	zobel	2.34
s2	brown	cis	kahn	2	watson	IT212
s2	brown	cis	kahn	2	woods	IT204
s2	brown	cis	kahn	2	capon	A14
s2	brown	cis	kahn	2	lindsey	2.10
s2	brown	cis	kahn	2	barringer	2.125
s3	smith	cs	goble	2	kahn	IT206
s3	smith	cs	goble	2	bush	2.26
s3	smith	cs	goble	2	goble	2.82
s3	smith	cs	goble	2	zobel	2.34
s3	smith	cs	goble	2	watson	IT212
s3	smith	cs	goble	2	woods	IT204
s3	smith	cs	goble	2	capon	A14
s3	smith	cs	goble	2	lindsey	2.10
s3	smith	cs	goble	2	barringer	2.125
s4	bloggs	ca	goble	1	kahn	IT206

...

# “Where are the Tutors of Students?”

To answer the query

“For each student, identified by name and student number, return the name of the tutor and their office number”

we have to

- combine tuples from Student and Staff
- that satisfy “Student.tutor=Staff.lecturer”
- and keep the attributes studno, name, (tutor or lecturer), and roomno.

In relational algebra:

STAFF

lecturer	roomno
kahn	IT206
bush	2.26
goble	2.82
zobel	2.34
watson	IT212
woods	IT204
capon	A14
lindsey	2.10
barringer	2.125

STUDENT

studno	name	hons	tutor	year
s1	jones	ca	bush	2
s2	brown	cis	kahn	2
s3	smith	cs	goble	2
s4	bloggs	ca	goble	1
s5	jones	cs	zobel	1
s6	peters	ca	kahn	3

$\pi_{\text{studno,name,lecturer,roomno}}(\sigma_{\text{tutor=lecturer}}(\text{Student} \times \text{Staff}))$

The part  $\sigma_{\text{tutor=lecturer}}(\text{Student} \times \text{Staff})$  is a “join”.



# Example: Student Marks in Courses

STUDENT

<u>studno</u>	name	hons	tutor	year
s1	jones	ca	bush	2
s2	brown	cis	kahn	2
s3	smith	cs	goble	2
s4	bloggs	ca	goble	1
s5	jones	cs	zobel	1
s6	peters	ca	kahn	3

*“For each student,  
show the courses in which they are  
enrolled and their marks”*

ENROL

<u>stud no</u>	<u>course no</u>	lab mark	exam mark
s1	cs250	65	52
s1	cs260	80	75
s1	cs270	47	34
s2	cs250	67	55
s2	cs270	65	71
s3	cs270	49	50
s4	cs280	50	51
s5	cs250	0	3
s6	cs250	2	7

First,

$R \leftarrow \sigma_{\text{Student.studno} = \text{Enrol.studno}}(\text{Student} \times \text{Enrol}),$

then

$\text{Result} \leftarrow \pi_{\text{studno}, \text{name}, \dots, \text{exam\_mark}}(R)$

# Join ( $\bowtie$ )

- The most used operator in the relational algebra.

Allows us to establish connections among data in different relations, taking advantage of the "data-based" nature of the relational model.

- Three main versions of the join:
  - "**natural**" join: takes attribute names into account;
  - "**theta**" join.
  - "**equi**" join (a special form of theta join)
  - all denoted by the symbol  $\bowtie$

# Natural Join

STUDENT

<u>studno</u>	name	hons	tutor	year
s1	jones	ca	bush	2
s2	brown	cis	kahn	2
s3	smith	cs	goble	2
s4	bloggs	ca	goble	1
s5	jones	cs	zobel	1
s6	peters	ca	kahn	3

ENROL

<u>stud no</u>	<u>course no</u>	lab mark	exam mark
s1	cs250	65	52
s1	cs260	80	75
s1	cs270	47	34
s2	cs250	67	55
s2	cs270	65	71
s3	cs270	49	50
s4	cs280	50	51
s5	cs250	0	3
s6	cs250	2	7

Student ⋈ Enrol

- **Implicit** join based on **common** attributes
- The tuples in the resulting relation are obtained by combining tuples in the operands with equal values on the common attributes
- Common attributes appear once in the results

# Student ⋈ Enrol

STUDENT

<u>studno</u>	name	hons	tutor	year
s1	jones	ca	bush	2
s2	brown	cis	kahn	2
s3	smith	cs	goble	2
s4	bloggs	ca	goble	1
s5	jones	cs	zobel	1
s6	peters	ca	kahn	3

ENROL

<u>stud</u> <u>no</u>	<u>course</u> <u>no</u>	lab mark	exam mark
s1	cs250	65	52
s1	cs260	80	75
s1	cs270	47	34
s2	cs250	67	55
s2	cs270	65	71
s3	cs270	49	50
s4	cs280	50	51
s5	cs250	0	3
s6	cs250	2	7

stuno	name	hons	tutor	year	courseno	labmark	exammark
s1	jones	ac	bush	2	cs250	65	52
s1	jones	ac	bush	2	cs260	80	75
s1	jones	ac	bush	2	cs270	47	34
s2	brown	is	kahn	2	cs250	67	55
s2	brown	is	kahn	2	cs270	65	71
s3	smith	cs	goble	2	cs270	49	50
s4	bloggs	ac	goble	1	cs250	50	51
s5	jones	cs	zobel	1	cs250	0	3
s6	peters	ac	kahn	3	cs250	2	7

(9 rows)

# Natural Join (another example)

**Offences**

<u>Code</u>	Date	Officer	Dept	Registration
143256	25/10/1992	567	75	5694 FR
987554	26/10/1992	456	75	5694 FR
987557	26/10/1992	456	75	6544 XY
630876	15/10/1992	456	47	6544 XY
539856	12/10/1992	567	47	6544 XY

**Cars**

<u>Registration</u>	<u>Dept</u>	Owner	...
6544 XY	75	Cordon Edouard	...
7122 HT	75	Cordon Edouard	...
5694 FR	75	Latour Hortense	...
6544 XY	47	Mimault Bernard	...

**Offences ⋈ Cars**

<u>Code</u>	Date	Officer	Dept	Registration	Owner	...
143256	25/10/1992	567	75	5694 FR	Latour Hortense	...
987554	26/10/1992	456	75	5694 FR	Latour Hortense	...
987557	26/10/1992	456	75	6544 XY	Cordon Edouard	...
630876	15/10/1992	456	47	6544 XY	Mimault Bernard	...
539856	12/10/1992	567	47	6544 XY	Mimault Bernard	...

# $\theta$ -Joins (read “Theta”-Joins), Equi-Joins

Theta-Join:

- *The most general form of JOIN ...*
- Theta join combines tuples from different relations provided they satisfy the theta condition. The join condition is denoted by the symbol  $\theta$ .
- Theta join can use comparison operators and common attributes are not required.

Student  $\bowtie_{\text{student.year} < \text{enrol.labmark}}$  Enrol

- The results include the ‘joined’ attributes from both relations
- The attribute names do not have to match (but their domains have to be compatible)

Equi-Join:

A special form of Theta-join, and *the most common form of JOIN ...*

with a **join** condition containing an equality operator (i.e., explicitly stating the joining attributes)

Student  $\bowtie_{\text{stuno}=\text{stuno}}$  Enrol

## STUDENT

<u>studno</u>	name	hons	tutor	year
s1	jones	ca	bush	2
s2	brown	cis	kahn	2
s3	smith	cs	goble	2
s4	bloggs	ca	goble	1
s5	jones	cs	zobel	1
s6	peters	ca	kahn	3

## STAFF

<u>lecturer</u>	roomno
kahn	IT206
bush	2.26
goble	2.82
zobel	2.34
watson	IT212
woods	IT204
capon	A14
lindsey	2.10
barringer	2.125

Student      tutor=lecturer      Staff



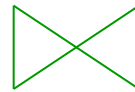
(equivalent to:  $\sigma_{\text{tutor}=\text{lecturer}}(\text{Student} \times \text{Staff})$ )

stud no	name	hons	tutor	year	lecturer	roomno
s1	jones	ca	bush	2	bush	2.26
s2	brown	cis	kahn	2	kahn	IT206
s3	smith	cs	goble	2	goble	2.82
s4	bloggs	ca	goble	1	goble	2.82
s5	jones	cs	zobel	1	zobel	2.34
s6	peters	ca	kahn	3	kahn	IT206

# Join: An Observation

Some tuples don't contribute to the result, they get lost.

Employee	Department
Brown	A
Jones	B
Smith	B



Department	Head
B	Black
C	White

Employee	Department	Head
Jones	B	Black
Smith	B	Black



# Outer Join

An outer join extends those tuples with null values that would get lost by a join like natural join or equi join (a.k.a. inner joins)

The outer join comes in three versions

- left: keeps the tuples of the left argument, extending them with nulls if necessary
- right: ... of the right argument ...
- full: ... of both arguments ...

# (Natural) Left Outer Join

Employee

Employee	Department
Brown	A
Jones	B
Smith	B

Department

Department	Head
B	Black
C	White

Employee ⋈<sup>Left</sup> Department

Employee	Department	Head
Brown	A	null
Jones	B	Black
Smith	B	Black

# (Natural) Right Outer Join

Employee

Employee	Department
Brown	A
Jones	B
Smith	B

Department

Department	Head
B	Black
C	White

Employee ⋈<sup>Right</sup> Department

Employee	Department	Head
Jones	B	Black
Smith	B	Black
null	C	White

# (Natural) Full Outer Join

Employee

Employee	Department
Brown	A
Jones	B
Smith	B

Department

Department	Head
B	Black
C	White

Employee ⋈<sup>Full</sup> Department

Employee	Department	Head
Brown	A	null
Jones	B	Black
Smith	B	Black
null	C	White

# Duplicate Elimination

Real DBMSs implement a version of relational algebra that operates on multisets (“bags”) instead of sets.

(Which of these operators may return bags,  
even if the input consists of sets?)

For the bag version of relational algebra, there exists a duplicate elimination operator  $\delta$ .

$$\text{If } R = \begin{array}{|c|c|} \hline A & B \\ \hline 1 & 2 \\ \hline 3 & 4 \\ \hline 3 & 4 \\ \hline 1 & 2 \\ \hline \end{array}, \text{ then } \delta(R) = \begin{array}{|c|c|} \hline A & B \\ \hline 1 & 2 \\ \hline 3 & 4 \\ \hline \end{array}$$

# Aggregation

Often, we want to retrieve aggregate values, like the “sum of salaries” of employees, or the “average age” of students.

This is achieved using aggregation functions, such as SUM, AVG, MIN, MAX, or COUNT.

Such functions are applied by the grouping and aggregation operator  $\gamma$ .

If $R =$	A	B	, then $\gamma_{\text{SUM(A)}}(R) =$	SUM(A)
	1	2		8
	3	4		
	3	5		
	1	1		
			and $\gamma_{\text{AVG(B)}}(R) =$	AVG(B)
				3

# Grouping and Aggregation

More often, we want to retrieve aggregate values for groups, like the “sum of employee salaries” per department, or the “average student age” per faculty.

As additional parameters, we give  $\gamma$  attributes that specify the criteria according to which the tuples of the argument are grouped.

E.g., the operator  $\gamma_{A, \text{SUM}(B)}(R)$

- partitions the tuples of  $R$  in groups that agree on  $A$ ,
- returns the sum of all  $B$  values for each group.

If  $R =$

A	B
1	2
3	4
3	5
1	3

, then  $\gamma_{A, \text{SUM}(B)}(R) =$

A	SUM(B)
1	5
3	9