Databases

Relational Algebra and Relational Calculus

Danilo Montesi

danilo.montesi@unibo.it

Database Languages

- Operations on the schema
  - DDL: data definition language
- Operations on data
  - DML: data manipulation language
    - **Query instructions** (to extract the data of interest)
    - **Update instructions** (to insert new data or modify current one)

Query Languages for Relational DBs

- **Declarative**

  They specify **what** are the results properties we want to obtain

- **Imperative/Procedural**

  They specify **how** the result is obtained

Query Languages

- **Relational Algebra**: procedural
- **Relational Calculus**: declarative (theoretical, not implemented)
- **SQL** (Structured Query Language): partially declarative (implemented)
- **QBE** (Query by Example): declarative (implemented)

Relational Algebra

- Defined by a set of **operators**
  - over the relations
  - … that produce relations as a result
  - … and can be compositional

Relational Algebra Operators

- union, intersection, difference
- rename
- select
- project
- join
  - natural join
  - cartesian product
  - theta-join (θ-join)

Set Operators

- Relations are sets
- The results must also be sets
- Thus, we can use **union**, **intersection**, **difference** if the involved relations have the same schema

Union

GRADUATED

Number	Name	Age
7274	Rossi	42
7432	Neri	54
9824	Verdi	45

TECHNICIANS

Number	Name	Age
9297	Neri	33
7432	Neri	54
9824	Verdi	45

GRADUATED ∪ TECHNICIANS

Number	Name	Age
7274	Rossi	42
7432	Neri	54
9824	Verdi	45
9297	Neri	33

Intersection

GRADUATED		
Number	Name	Age
7274	Rossi	42
7432	Neri	54
9824	Verdi	45

TECHNICIANS		
Number	Name	Age
9297	Neri	33
7432	Neri	54
9824	Verdi	45

GRADUATED ∩ TECHNICIANS

Number	Name	Age
7432	Neri	54
9824	Verdi	45

Difference

GRADUATED		
Number	Name	Age
7274	Rossi	42
7432	Neri	54
9824	Verdi	45

TECHNICIANS		
Number	Name	Age
~~9297~~	~~Neri~~	~~33~~
7432	Neri	54
9824	Verdi	45

GRADUATED - TECHNICIANS

Number	Name	Age
7274	Rossi	42

A Correct (but Impossible) Union

FATHERHOOD	
Father	Child
Adam	Abel
Adam	Cain
Abraham	Isaac

MOTHERHOOD	
Mother	Child
Eve	Abel
Eve	Seth
Sarah	Isaac

FATHERHOOD ∪ MOTHERHOOD
??

Renaming

- Unary operator (only one argument)
- It produces a result that “changes the schema” while keeping the contained data unaltered

Renaming: Syntax & Semantics

- Syntax:

$$\rho_{NewName \leftarrow OldName}\ (RELATION)$$

- Semantics:

  Changes the name of the attribute from "*OldName*" to "*NewName*"

Renaming: an Example (1)

FATHERHOOD	
Father	Child
Adam	Abel
Adam	Cain
Abraham	Isaac

$\rho_{\text{Parent} \leftarrow \text{Father}}$(FATHERHOOD)

Parent	Child
Adam	Abel
Adam	Cain
Abraham	Isaac

Renaming: an Example (2)

FATHERHOOD	
Father	Child
Adam	Abel
Adam	Cain
Abraham	Isaac

$\rho_{\text{Parent} \leftarrow \text{Father}}$ (FATHERHOOD)

Parent	Child
Adam	Abel
Adam	Cain
Abraham	Isaac

MOTHERHOOD	
Mother	Child
Eve	Abel
Eve	Seth
Sarah	Isaac

$\rho_{\text{Parent} \leftarrow \text{Mother}}$ (MOTHERHOOD)

Parent	Child
Eve	Abel
Eve	Seth
Sarah	Isaac

Renaming: an Example (3)

$\rho_{\text{Parent} \leftarrow \text{Father}}$ (FATHERHOOD)

Parent	Child
Adam	Abel
Adam	Cain
Abraham	Isaac

$\rho_{\text{Parent} \leftarrow \text{Mother}}$ (MOTHERHOOD)

Parent	Child
Eve	Abel
Eve	Seth
Sarah	Isaac

$\rho_{\text{Parent} \leftarrow \text{Father}}$ (FATHERHOOD) $\cup$

$\rho_{\text{Parent} \leftarrow \text{Mother}}$ (MOTHERHOOD)

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

Renaming: Another Example

EMPLOYEE		
Surname	Office	Salary
Rossi	Rome	55
Neri	Milan	64

WORKER		
Surname	Factory	Wage
Bruni	Paris	45
Verdi	Berlin	55

$\rho_{\text{Pay} \leftarrow \text{Salary}}$ (EMPLOYEE) $\cup$

$\rho_{\text{Office, Pay} \leftarrow \text{Factory, Wage}}$ (WORKER)

Surname	Office	Pay
Rossi	Rome	55
Neri	Milan	64
Bruni	Paris	45
Verdi	Berlin	55

Selection

- Unary operator
- As a result:
  - The output has the same schema of the input
  - The output tuples are a subset of the input tuples
  - The output tuples satisfy a predicate

Selection: Syntax & Semantics

- Syntax:

$$\sigma_{predicate}\ (RELATION)$$

Predicate: its interpretation is a boolean expression over the relations' tuples

- Semantics:
A subset of the relation that satisfy the given predicate

Selection: an Example (1)

EMPLOYEE			
Number	Surname	Office	Salary
7309	Rossi	Rome	55
5998	Neri	Milan	64
9553	Milan	Milan	44
5698	Neri	Naple	64

Return all the employees that:

- Earn more than 50
- Earn more than 50 and work in Milan
- Have the same surname as their city's office

Selection: an Example (2)

- Employees earning more than 50

$$\sigma_{\text{Salary} > 50}\ (\text{EMPLOYEE})$$

Number	Surname	Office	Salary
7309	Rossi	Rome	55
5998	Neri	Milan	64
5698	Neri	Naple	64

Selection: an Example (3)

- Employees earning more than 50 and working in Milan

$$\sigma_{\text{Salary > 50 AND Office = 'Milan'}} \text{ (EMPLOYEE)}$$

Number	Surname	Office	Salary
5998	Neri	Milan	64

Selection: an Example (4)

- Employees having the same surname as their city's office

$$\sigma_{\text{Surname } = \text{Office}} \text{(EMPLOYEE)}$$

Number	Surname	Office	Salary
9553	Milan	Milan	44

Projection

- Unary operator
- As a result:
  - the output's schema is a subset of the inputs' schema
  - the output is formed using all the input's tuples

Projection: Syntax & Semantics

- Syntax:

$$\pi_{AttributeList}\ (RELATION)$$

- Semantics:

  The results contains all the tuples in Relation, but only the attributes in AttributeList

Projection: an Example (1)

EMPLOYEE			
Number	Surname	Office	Salary
7309	Rossi	Rome	55
5998	Neri	Milan	64
9553	Milan	Milan	44
5698	Neri	Naple	64

For all the employees return

- Number and Surname
- Surname and Office

Projection: an Example (2)

- Return the Employees' number and surname

$$\pi_{\text{Number, Surname}} \text{ (EMPLOYEE)}$$

Number	Surname		
7309	Rossi		
5998	Neri		
9553	Milan		
5698	Neri		

Projection: an Example (3)

- Surname and Office for all the Employees

$$\pi_{\text{Surname, Office}}\ (\text{EMPLOYEE})$$

	Surname	Office	
	Rossi	Rome	
	Neri	Milan	
	Milan	Milan	
	Neri	Naple	

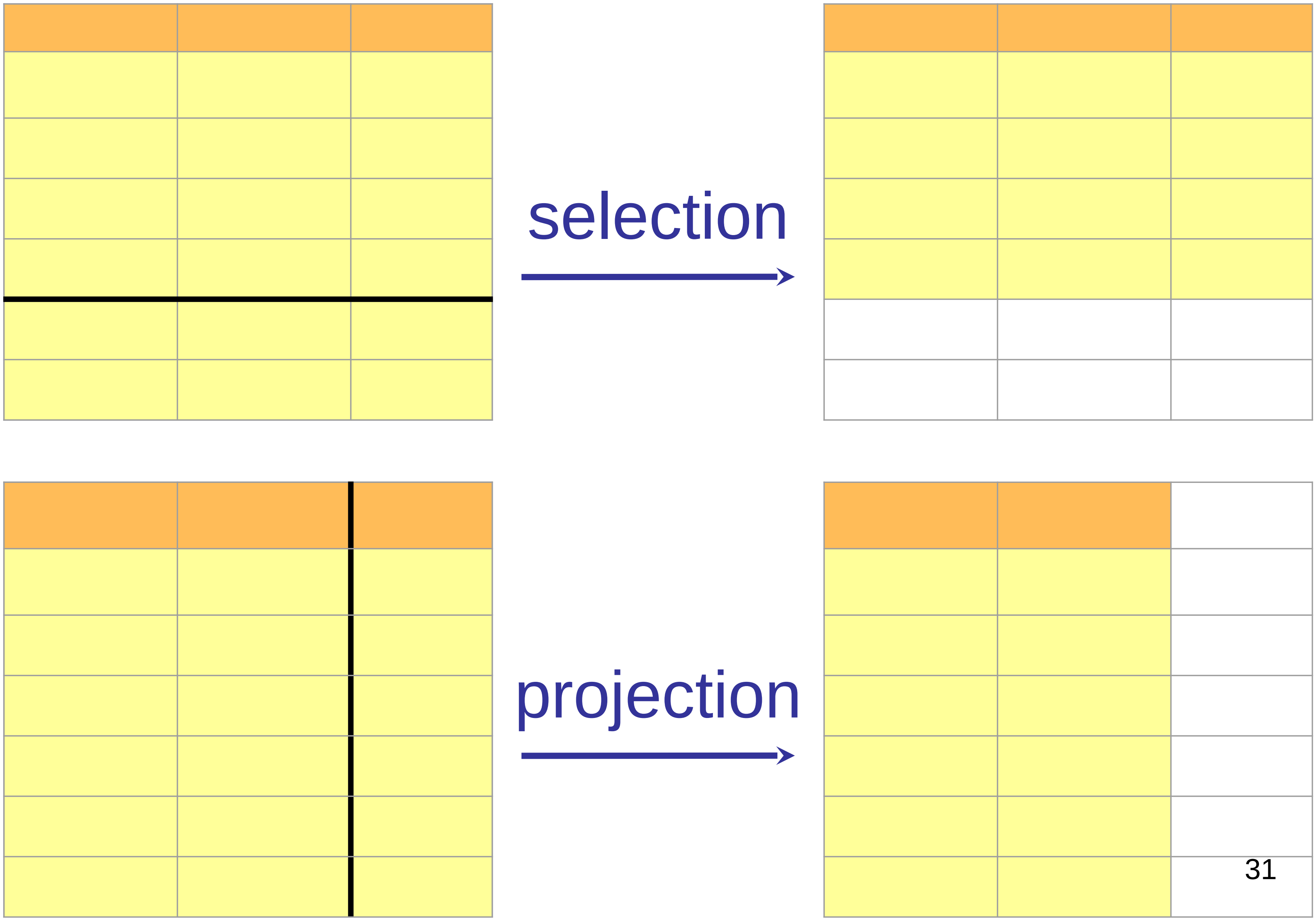
Projections' Cardinality

- A projection's output
  - Contains at most the same number of tuples as the input's
  - Could contain fewer tuples: restricting to a subset of the attributes, some tuples could be repeated! Repeated tuples are discarded in relations
- If X is a superkey for R, then π_X (R) contains the same number of tuples as R

Selection and Projection (1)

- “Orthogonal” operators
  - **Selections**: horizontal decomposition
  - **Projection**: vertical decomposition

Selection and Projection (2)



Selection and Projection (3)

- By combining selection and projection, it is possible to extract only the interesting information from a relation
- Let’s see an example

Selection and Projection (4)

- Return the Number and the Surname of the employees' having a salary greater than 50

$$\pi_{\text{Number, Surname}}\ (\ \sigma_{\text{Salary} > 50}\ (\text{EMPLOYEE})\ )$$

Number	Surname		
7309	Rossi		
5998	Neri		
5698	Neri		

Limits of the Combo: σ + π

- By using such operators we could only extract information from **one single** relation
- Hereby we can correlate neither tuples from different relations, nor different tuples from the same relation

Join

- The *join* is the most interesting operator of the relational algebra
- It allows the correlation between tuples in different relations

Exams During a Public Contest

- Each exam is anonymous and a closed envelope, containing the candidate’s surname, is associated to it
- Each exam and its related envelope have the same ID

Data

Each exam is anonymous

1	25
2	13
3	27
4	28

A closed envelope …

1	Jan Johansson
2	Jean B. Lully
3	Johann S. Bach
4	Giuseppe Verdi

	Jan Johansson	25
	Jean B. Lully	13
	Johann S. Bach	27
	Giuseppe Verdi	28

Data with Schema

Id	Grade
1	25
2	13
3	27
4	28

Id	Candidate
1	Jan Johansson
2	Jean B. Lully
3	Johann S. Bach
4	Giuseppe Verdi

Id	Candidate	Grade
1	Jan Johansson	25
2	Jean B. Lully	13
3	Johann S. Bach	27
4	Giuseppe Verdi	28

Natural Join

- Binary operator (generalizable through associativity)
- Provides a result such that:
  - joins two tables based on same attribute name
  - its schema is the union of the attributes of the two relations' schema
  - each tuple is produced combining two tuples: one from each of the two relations

Join: Syntax & Semantics

- Syntax: given $R_1(X_1)$, $R_2(X_2)$

$$R_1 \bowtie R_2$$

is a relation over X_1X_2

- Semantics:

$$R_1 \bowtie R_2 = \{\, t \text{ on } X_1X_2 \mid$$

$$\exists t_1 \in R_1 \text{ and } \exists t_2 \in R_2$$

$$\text{with } t[X_1] = t_1 \text{ and } t[X_2] = t_2 \,\}$$

Full Join

Employee	Dept
Rossi	A
Neri	B
Bianchi	B

$\bowtie$

Dept	Chief
A	Mori
B	Bruni

Employee	Dept	Chief
Rossi	A	Mori
Neri	B	Bruni
Bianchi	B	Bruni

- **Full join**: each tuple contributes to the final result

A “Not Full” Join

Employee	Dept
Rossi	A
Neri	B
Bianchi	B

$\bowtie$

Dept	Chief
B	Mori
C	Bruni

Employee	Dept	Chief
Neri	B	Mori
Bianchi	B	Mori

An “Empty” Join

Employee	Dept
Rossi	A
Neri	B
Bianchi	B

$\bowtie$

Dept	Chief
D	Mori
C	Bruni

Employee	Dept	Chief

A Full Join Having $n \times m$ Tuples

Employee	Dept
Rossi	B
Neri	B

$\bowtie$

Dept	Chief
B	Mori
B	Bruni

Employee	Dept	Chief
Rossi	B	Mori
Rossi	B	Bruni
Neri	B	Mori
Neri	B	Bruni

Size of a Join's Result

- The result of the join between R_1 and R_2 has a number of tuples between zero and $|R_1| \times |R_2|$

$$0 \leq |R_1 \bowtie R_2| \leq |R_1| \times |R_2|$$

- If the join involves a key from R_2, then the number of the resulting tuples is within 0 and $|R_1|$

$$0 \leq |R_1 \bowtie R_2| \leq |R_1|$$

- If the join involves a key from R_2 and a Referential Integrity Constraint, the number of the tuples is $|R_1|$

$$|R_1 \bowtie R_2| = |R_1|$$

Size of a Join's Result: Examples (1)

- $|R_1|$ and $|R_2|$ have a size of 3
- The size of the join between R_1 and R_2 is at most 9 (3×3)

R_1	Employee	Dept
	Rossi	A
	Neri	B
	Bianchi	B

R_2	Dept	Room
	B	1
	B	2
	C	3

$$|R_1 \bowtie R_2| = 4$$

Size of a Join's Result: Examples (2)

- The size of the join between R_1 and R_2 involves R_2 's key
- Hence its size is between 0 and $|R_1|$
- Since the key's value are unique, for each tuple of R_2 can match more tuples of R_1 and not vice versa

R_1	Employee	Dept
	Rossi	A
	Neri	B
	Bianchi	B

R_2	<u>Dept</u>	Chief
	B	Mori
	C	Bruni

$$|R_1 \bowtie R_2| \leq |R1| = 2$$

Size of a Join's Result: Examples (3)

- If there exists a referential integrity constraint between Dept (foreign key in R_1) and Dept (key in R_2):

$$|R_1 \bowtie R_2| = |R_1|$$

each tuple in R_1 is associated to at least one tuple in R_2

R_1	Employee	Dept
	Rossi	A
	Neri	B
	Bianchi	B

R_2	<u>Dept</u>	Chief
	A	Bruni
	B	Mori
	C	Neri

$$|R_1 \bowtie R_2| = |R1| = 3$$

Join: Critical Issues

Employee	Dept
Rossi	A
Neri	B
Bianchi	B

$\bowtie$

Dept	Chief
B	Mori
C	Bruni

Employee	Dept	Chief
Neri	B	Mori
Bianchi	B	Mori

Some tuples do not contribute to the final result: they are “left out”

Outer Join

- The **outer** join fills with NULL values the tuples that are normally discarded from an "**inner**" join
- There are three types of "outer joins":
  - **left outer join**
  - **right outer join**
  - **full outer join**

Outer Join: Syntax & Semantics

- **left outer join** ($⟕$)
  keeps all the tuples from the first operand, even if there are no right operand matching tuples. The latter are replaced by NULL values
- **right outer join** ($⟖$)
  A $⟖$ B is defined as B $⟕$ A
- **full outer join** ($⟗$)
  is defined as the union of the former operands

Left Outer Join

EMPLOYEE	
Employee	Dept
Rossi	A
Neri	B
Bianchi	B

⟕

DEPARTMENT	
Dept	Chief
B	Mori
C	Bruni

EMPLOYEE ⟕ DEPARTMENT		
Employee	Dept	Chief
Neri	B	Mori
Bianchi	B	Mori
Rossi	A	NULL

Right Outer Join

EMPLOYEE	
Employee	Dept
Rossi	A
Neri	B
Bianchi	B

⟖

DEPARTMENT	
Dept	Chief
B	Mori
C	Bruni

EMPLOYEE ⟖ DEPARTMENT		
Employee	Dept	Chief
Neri	B	Mori
Bianchi	B	Mori
NULL	C	Bruni

Full Outer Join

EMPLOYEE	
Employee	Dept
Rossi	A
Neri	B
Bianchi	B

⟗

DEPARTMENT	
Dept	Chief
B	Mori
C	Bruni

EMPLOYEE ⟗ DEPARTMENT		
Employee	Dept	Chief
Neri	B	Mori
Bianchi	B	Mori
Rossi	A	NULL
NULL	C	Bruni

Cartesian Product (1)

EMPLOYEE	
Employee	Dept
Rossi	A
Neri	B
Bianchi	B

DEPARTMENT	
Code	Chief
A	Mori
B	Bruni

EMPLOYEE ⋈ DEPARTMENT			
Employee	Dept	Code	Chief
Rossi	A	A	Mori
Rossi	A	B	Bruni
Neri	B	A	Mori
Neri	B	B	Bruni
Bianchi	B	A	Mori
Bianchi	B	B	Bruni

Cartesian Product (2)

- The result's size is the product of the operands' size
- In practice, cartesian product is useful only if followed by a selection:

$$\sigma_{condition}\ (R1 \bowtie R2)$$

Theta-join: Why?

- The cartesian product assumes a specific meaning when it is followed by a selection operation:

$$\sigma_{condition} (R_1 \bowtie R_2)$$

- Such operation is defined as **theta-join** (θ-join)

$$R_1 \bowtie_{condition} R_2$$

Theta-join: Condition

- The predicate condition is usually defined as a conjunction (AND) of atoms expressing binary relations

$$A_1 \mathscr{R}\, A_2$$

where $\mathscr{R}$ is a comparison operator (=, >, <, …)

Equi-join

- A theta-join is called "**equi-join**" iff all the $\mathscr{R}$ atoms in theta are equivalence relations

  Note that the equi-join is used much more often than the most general theta-join: it allows to specify on **which attribute to join without using rename operations**

Equi-join: an Example (1)

EMPLOYEE	
Employee	Dept
Rossi	A
Neri	B
Bianchi	B

DEPARTMENT	
Code	Chief
A	Mori
B	Bruni

EMPLOYEE $\bowtie_{Dept=Code}$ DEPARTMENT			
Employee	Dept	Code	Chief
Rossi	A	A	Mori
Neri	B	B	Bruni
Bianchi	B	B	Bruni

Equi-join: an Example (2)

- **Please Note**: The equi-join provides a similar result as the join between EMPLOYEE and DEPARTMENT where “Code” is renamed as “Dept”

$EMPLOYEE \bowtie \rho_{Dept \leftarrow Code}(DEPARTMENT)$		
Employee	**Dept**	**Chief**
Rossi	A	Mori
Neri	B	Bruni
Bianchi	B	Bruni

Natural Join and Equi-join

- After renaming we obtain the two following schemas, which can be the joined with natural join. We then proceed to selection and projection

EMPLOYEE	
Employee	Dept

DEPARTMENT	
Dept	Chief

$$\text{EMPLOYEE} \bowtie \text{DEPARTMENT}$$

$$\pi_{\text{Employee,Dept,Chief}}($$

$$\sigma_{\text{Dept=Code}}(\ \text{EMPLOYEE} \bowtie \ \rho_{\text{Code} \leftarrow \text{Dept}}(\text{DEPARTMENT}))\ )$$

Examples

EMPLOYEE	Number	Name	Age	Wage
	7309	Rossi	34	45
	5998	Bianchi	37	38
	9553	Neri	42	35
	5698	Bruni	43	42
	4076	Mori	45	50
	8123	Lupi	46	60

SUPERVISOR	Employee	Chief
	7309	5698
	5998	5698
	9553	4076
	5698	4076
	4076	8123

Relational Algebra: Examples (1)

- Return the employees' number, name, age and wage earning more than 40

$$\sigma_{Wage>40}\ (\text{EMPLOYEE})$$

Relational Algebra: Examples (2)

- Return the employees' number, name and age earning more than 40

$$\pi_{Number,Name,Age}\ (\sigma_{Wage>40}\ (EMPLOYEE))$$

Relational Algebra: Examples (3)

- Return the chiefs whose employees earn more than 40

$$\pi_{\text{Chief}}\,(\text{SUPERVISOR} \bowtie_{\text{Employee=Number}}$$

$$(\sigma_{\text{Wage>40}}\,(\text{EMPLOYEE})))$$

Relational Algebra: Examples (4)

- Return the chief's name and wage having employees earning more than 40

$$
\begin{aligned}
&\pi_{\text{Name,Wage}} (\text{EMPLOYEE} \bowtie_{\text{Number=Chief}} \\
&\qquad \pi_{\text{Chief}} (\text{SUPERVISOR} \bowtie_{\text{Employee=Number}} \\
&\qquad\qquad \sigma_{\text{Wage>40}} (\text{EMPLOYEE}) \\
&\qquad) \\
&)
\end{aligned}
$$

Relational Algebra: Examples (5)

- Return the employees having a wage greater than their chief's, along with number, name and wage of both the employee and his chief

$$\pi_{Number,Name,Wage,NumC,NameC,WageC}\Big($$

$$\sigma_{Wage>WageC}\Big($$

$$\rho_{NumC,NameC,WageC,AgeC \leftarrow Number,Name,Wage,Age}(EMPLOYEE) \bowtie_{NumC=Chief}$$

$$(SUPERVISOR \bowtie_{Employee=Number} EMPLOYEE)$$

$$\Big)$$

$$\Big)$$

Relational Algebra: Examples (6)

- Return the chiefs' number having all their employees earning more than 40

$$\pi_{Chief}(\text{SUPERVISOR}) -$$
$$\pi_{Chief}(\text{SUPERVISOR}$$
$$\bowtie_{\text{Employee=Number}}$$
$$\sigma_{\text{Wage} \leq 40}(\text{EMPLOYEE})$$
$$)$$

Equivalent Relational Algebra Expressions

- Two Relational Algebra expressions E_1 and E_2 are **equivalent** if they provide the same result independently from the database's state, but may depend on the schema
- Equivalence rules are very important because modern DBMSs try to rewrite the queries into **equivalent but more efficient** expressions (i.e., less time consuming)

Equivalent Expressions (1)

1. Combining a cascade of selections:

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

2. Projection is idempotent (*X* and *Y* belong to R's schema)

$$\pi_{X}(R) = \pi_{X}(\pi_{X\,Y}(R))$$

3. Projection vs selection order

$$\pi_{X\,Y}(\sigma_{X}(R)) = \sigma_{X}(\pi_{X\,Y}(R))$$

Equivalent Expressions (2)

4. Push selections down

$$\sigma_C (R_1 \bowtie R_2) = R_1 \bowtie (\sigma_C (R_2))$$

- *C* involves attributes that belong to R2’s schema
- It drastically reduces the intermediate size of the result (therefore also the operation cost)

Equivalent Expressions (3)

5. Pushing projections down

$$\pi_{X1\,Y2}(R_1 \bowtie R_2) \;=\; R_1 \bowtie \pi_{Y2}(R_2)$$

- R1 has schema *X1*, R2 has schema *X2*
- If ***Y2*** $\subseteq$ **(*X1*** $\cap$ ***X2*)**, the attributes in *X2* - *Y2* are not involved in the join, and hence the equivalence is valid

Equivalent Expressions (4)

6. Using the theta-join definition

$$\sigma_C (R_1 \bowtie R_2) = R_1 \bowtie_C R_2$$

example …

$$\pi_{Chief} (\sigma_{Age<30 \text{ AND } Number=Employee} (EMPLOYEE \bowtie SUPERVISOR)) =$$

$$\pi_{Chief} (\sigma_{Number=Employee} (\sigma_{Age<30} (EMPLOYEE \bowtie SUPERVISOR))) =$$

$$\pi_{Chief} (\sigma_{Age<30} (EMPLOYEE) \bowtie_{Number=Employee} SUPERVISOR) =$$

$$\pi_{Chief} (\pi_{Number} (\sigma_{Age<30} (EMPLOYEE)) \bowtie_{Number=Employee} SUPERVISOR)$$

Equivalent Expressions (5)

7. $\sigma_C (R_1 \cup R_2) = \sigma_C (R_1) \cup \sigma_C (R_2)$

8. $\sigma_C (R_1 - R_2) = \sigma_C (R_1) - \sigma_C (R_2)$

9. $\pi_X (R_1 \cup R_2) = \pi_X (R_1) \cup \pi_X (R_2)$

- Some distributive properties
- **Please note** projection is not distributive over difference

Equivalent Expressions (6)

10. $\sigma_{C1 \text{ OR } C2}(R) = \sigma_{C1}(R) \cup \sigma_{C2}(R)$

11. $\sigma_{C1 \text{ AND } C2}(R) = \sigma_{C1}(R) \cap \sigma_{C2}(R)$

12. $\sigma_{C1 \text{ AND } \neg C2}(R) = \sigma_{C1}(R) - \sigma_{C2}(R)$

- Some properties about sets and selection

Equivalent Expressions (7)

13. Distributive property of join with respect to union

$$R_1 \bowtie (R_2 \cup R_3) = (R_1 \bowtie R_2) \cup (R_1 \bowtie R_3)$$

- Moreover, the associative and commutative property applies to all binary operators except difference

Please Note

- You don’t have to learn to write efficient relational algebra queries in this course
  - it’s better having correct and clear answers
- That’s because modern DBMSs already do that work for you

Selection with NULL Values

$\sigma_{Age>40}$ (PEOPLE)

PEOPLE			
Number	Surname	Agency	Age
7309	Rossi	Rome	32
5998	Neri	Milan	45
9553	Bruni	Milan	NULL

- No NULL value satisfies this specific atomic condition

An Undesirable Result

$$\sigma_{Age>30}(PEOPLE) \cup \sigma_{Age\leq30}(PEOPLE) \neq PEOPLE$$

- Selections are evaluated separately

$$\sigma_{Age>30 \text{ OR } Age\leq30}(PEOPLE) \neq PEOPLE$$

- Atomic conditions are evaluated separately

NULL Valued Selections: a Solution

- The following atomic condition is satisfied only by non-NULL values $\sigma_{Age>40}$ (PEOPLE)
- Specific atomic statements are used for referring to NULL values

IS NULL

IS NOT NULL

- We could even define a three-valued logic (based upon true, false, unknown), but it is not necessary

NULL Valued Selections (1)

Therefore …

$$\sigma_{Age>30}(\text{PEOPLE}) \cup$$

$$\sigma_{Age\leq 30}(\text{PEOPLE}) \cup$$

$$\sigma_{Age\ IS\ NULL}(\text{PEOPLE})$$

$$=$$

$$\sigma_{Age>30\ OR\ Age\leq 30\ OR\ Age\ IS\ NULL}(\text{PEOPLE})$$

$$=$$

$$\text{PEOPLE}$$

NULL Valued Selections (2)

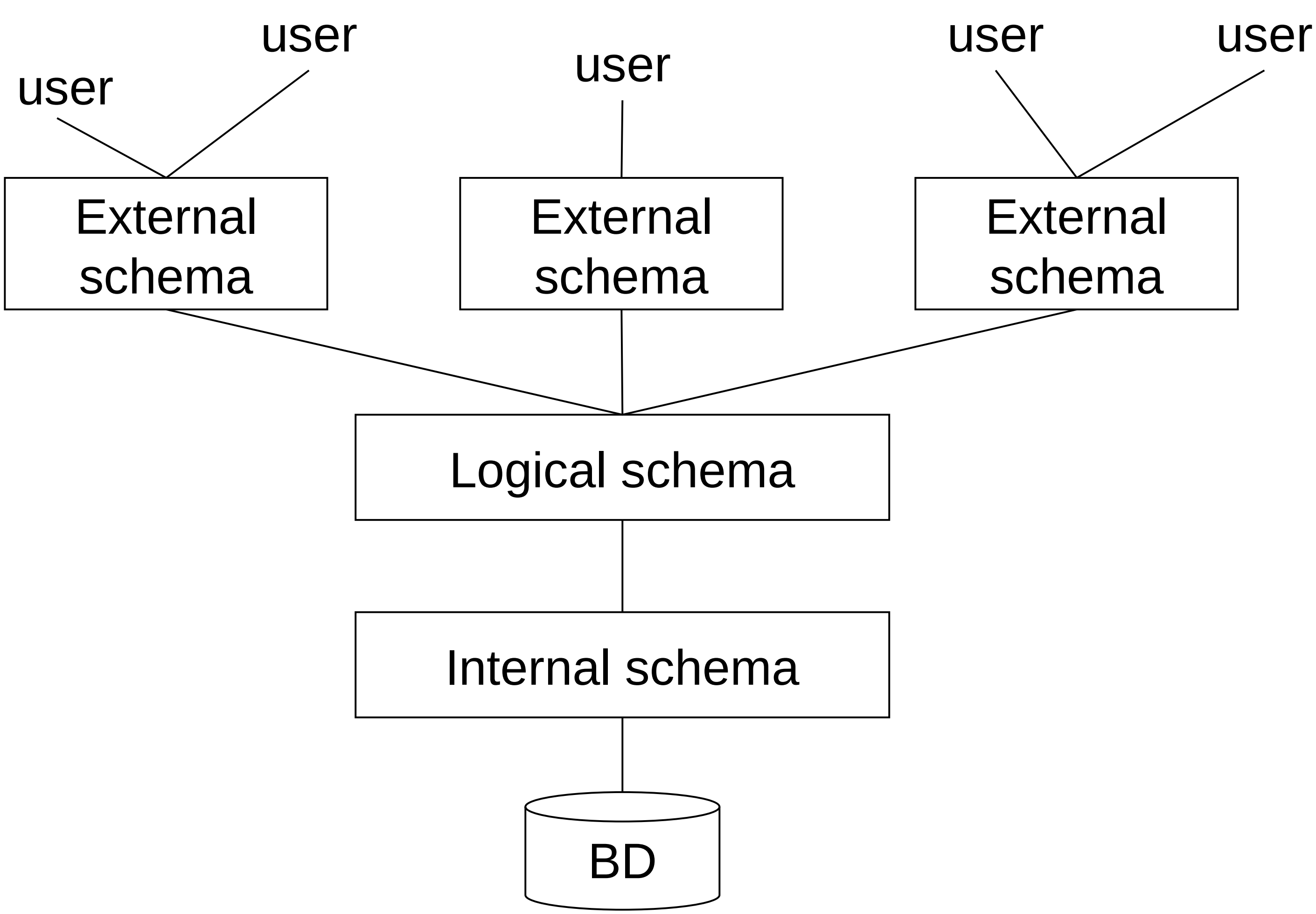
$\sigma_{\text{(Age > 40) OR (Age IS NULL)}}$ (PEOPLE)

PEOPLE			
Number	Surname	Agency	Age
7309	Rossi	Rome	32
5998	Neri	Milan	45
9553	Bruni	Milan	NULL

Views

- Different representations for the same data
  - **Derived tables**: such relations are created from queries
  - **Base tables**: original content
- Derived relations could be formed from other derived relations, but …

Standard Three-tiered ANSI/SPARC Architecture



Views: an Example

AFFILIATION	
Employee	Dept
Rossi	A
Neri	B
Bianchi	B

MANAGEMENT	
Dept	Chief
A	Mori
B	Bruni

- A view:

SUPERVISOR := $\pi_{\text{Employee,Chief}}$ (AFFILIATION $\bowtie$ MANAGEMENT)

Using Views

- Views could be used in two fashions:
  - **Materialized views**
  - **Virtual relations** (or **views**)

View Materialization

- A temporary or permanent view table is **stored physically** in the database
  - Pros:
    - Data is promptly available for further queries
  - Cons:
    - Data redundancy
    - Updates are slowed down
    - DBMS rarely support them

Virtual Relations (1)

- **Virtual relations** (or **views**)
  - All the DBMS support them
  - The view query is transformed into a query on the underlying database

Virtual Relations (2)

- The view's name is replaced by its associated query

$$\sigma_{\text{Chief='Mori'}}\ (\text{SUPERVISOR})$$

Is run as:

$$\sigma_{\text{Chief='Mori'}}\ (\pi_{\text{Employee, Chief}}\ (\text{AFFILIATION} \bowtie \text{MANAGEMENT}))$$

Views: Reasons (1)

- *External schema*: each user cannot only see
  - both what (s)he is interested on and in the way (s)he likes it, with no further distractions
  - what (s)he is allowed to see

Views: Reasons (2)

- *Programming tool*:
  - We could simplify the writing of complex queries when sub-expressions are repeated
- Using already-existent software over refactored schemas
- But …
  - Views do not affect queries' efficiency!

Views as a Programming Tool

- Return the employees having Jones's Chief
  - *Without views*:

$$\pi_{Employee}\ ((AFFILIATION \bowtie MANAGEMENT) \bowtie$$
$$\rho_{EmpR,RepR \leftarrow Employee,Dept}\ ($$
$$\sigma_{Employee='Jones'}\ (AFFILIATION \bowtie MANAGEMENT))$$
$$)$$

  - *Using views*:

$$\pi_{Employee}\ (SUPERVISOR \bowtie$$
$$\rho_{EmpR \leftarrow Employee}\ (\sigma_{Employee='Jones'}\ (SUPERVISOR))$$
$$)$$

Updating Views

AFFILIATION	
Employee	Dept
Rossi	A
Neri	B
Verdi	A

MANAGEMENT	
Dept	Chief
A	Mori
B	Bruni
C	Bruni

SUPERVISOR	
Employee	Dept
Rossi	Mori
Neri	Bruni
Verdi	Mori

- How could we update the data such that Bruni is Lupi's chief and that Falchi is Belli's chief?

Incremental Updates

- To “update a view” means to **change the base tables** such that the updated view reflects the update
- To each update over the view must correspond to the one over the tables
  - Such update could be not unambiguous!
  - Only a few possible updates are allowed on views

An Alternative Notation for Join

- Please note: such approach is usually adopted for SQL implementations
- We ignore the *Natural Join*: we do not implicitly assume conditions over attributes with the same name
- We disambiguate the attributes with the same name over different relations using the *RELATION.Attribute* syntax
- We give relations a new name by creating views, and we rename attributes only when it's needed for the union operation

An Example

EMPLOYEE	<u>Number</u>	Name	Age	Wage
	7309	Rossi	34	45
	5998	Bianchi	37	38
	9553	Neri	42	35
	5698	Bruni	43	42
	4076	Mori	45	50
	8123	Lupi	46	60

SUPERVISOR	Employee	Chief
	7309	5698
	5998	5698
	9553	4076
	5698	4076
	4076	8123

Conventions and Notations (1)

- Find the employees earning more than their chiefs, showing the number, name and wage of both employee and chief

$$
\begin{array}{l}
\pi_{\text{Number,Name,Wage,NumC,NameC,WageC}}\,(\\
\quad \sigma_{\text{Wage>WageC}}(\\
\qquad\qquad\qquad \rho_{\text{NumC,NameC,AgeC,WageC} \leftarrow \text{Number,Name,Age,Wage}}\,(\text{EMPLOYEE}) \\
\qquad\qquad\qquad\qquad\qquad \bowtie_{\text{NumC=Chief}} \\
\qquad (\text{SUPERVISOR} \bowtie_{\text{Employee=Number}} \text{EMPLOYEE}) \\
\quad) \\
)
\end{array}
$$

Conventions and Notations (2)

- Assign Employee to Chief (view)

$$CHIEF := EMPLOYEE$$

- We use relations' name as prefix to differentiate between shared attributes, for example

$$EMPLOYEE.Wage \text{ and } CHIEF.Wage$$

$$\pi_{EMPLOYEE.Number,EMPLOYEE.Name,EMPLOYEE.Wage,\ CHIEF.Number,CHIEF.Name,CHIEF.Wage} ($$

$$\sigma_{EMPLOYEE.Wage>CHIEF.Wage} (CHIEF \bowtie_{CHIEF.Number=Chief}$$

$$(SUPERVISOR \bowtie_{Employee=EMPLOYEE.Number} EMPLOYEE)$$

$$)$$

$$)$$

Relational Calculi

- A family of declarative languages based on First Order Logic
- Two different definitions:
  - **Domain Relational Calculus**
  - **Tuple Relational Calculus with Range Declarations**

Domain Relational Calculus

- Syntax: an expression is in the form

$$\{ A_1: x_1, \ldots, A_k: x_k \mid f \}$$

  - $A_1: x_1, \ldots, A_k: x_k$ target list:
    - $A_1, \ldots, A_k$ different attributes (can be in different databases)
    - $x_1, \ldots, x_k$ different variables
  - f is a formula (with boolean operators and quantifiers)
- Semantics: the result is a relation over $A_1, \ldots, A_k$ containing tuples of values for $x_1, \ldots, x_k$ satisfying the formula f

Comments

- Differences with Predicate Logic (for those who know it):
  - “predicate” symbols:
    - represent relations within the database
    - “standard” built-in predicates (=, >, ...)
  - There are no function symbols
  - The most interesting are “unbounded” predicates
  - Non-positional notation

Example

EMPLOYEE(<u>Number</u>, Name, Age, Wage)

SUPERVISOR(Chief, <u>Employee</u>)

Example (0a)

- Return the employees' number, name, age and salary earning more than 40

$$\sigma_{Wage>40}\ (\text{EMPLOYEE})$$

$$\{\ \text{Number: } m,\ \text{Name: } n,\ \text{Age: } a,\ \text{Wage: } w \mid \text{EMPLOYEE(Number: } m,\ \text{Name: } n,\ \text{Age: } a,\ \text{Wage: } w) \ \wedge\ w > 40\ \}$$

Example (0b)

- Return the employees' number, name and age for all

$$\pi_{\text{Number,Name,Age}} (\text{EMPLOYEE})$$

$$\{ \text{Number: } m, \text{ Name: } n, \text{ Age: } a \mid \exists w \, (\text{EMPLOYEE}(\text{Number: } m, \text{ Name: } n, \text{ Age: } a, \text{ Wage: } w) \}$$

$$\{ \text{Number: } m, \text{ Name: } n, \text{ Age: } a \mid \text{EMPLOYEE}(\text{Number: } m, \text{ Name: } n, \text{ Age: } a, \text{ Wage: } w) \}$$

Example (1)

- Return the employees' number, name and age earning more than 40

$$\pi_{\text{Number,Name,Age}}\,(\sigma_{\text{Wage}>40}\,(\text{EMPLOYEE}))$$

$$\{\ \text{Number: } m, \text{ Name: } n, \text{ Age: } a \mid \text{EMPLOYEE(Number: } m, \text{ Name: } n, \text{ Age: } a, \text{ Wage: } w) \ \wedge\ w > 40\ \}$$

Example (2)

- Return the number identifying the employees' chiefs earning more than 40

$$\pi_{\text{Chief}}\ (\text{SUPERVISOR} \bowtie_{\text{Employee=Number}} \sigma_{\text{Wage>40}}\ (\text{EMPLOYEE}))$$

$$\{\ \text{Chief}: c \mid \text{SUPERVISOR}(\text{Chief}: c, \text{Employee}: e) \wedge \text{EMPLOYEE}(\text{Number}: e, \text{Name}: n, \text{Age}: a, \text{Wage}: w) \wedge w > 40\ \}$$

Example (3)

- Return the chiefs' name and salary having employees earning more than 40

$$\pi_{\text{NameC,WageC}}\ (\rho_{\text{NumC,NameC,WageC,AgeC} \leftarrow \text{Number,Name,Wage,Age}}\ (\text{EMPLOYEE})$$

$$\bowtie_{\text{NumC=Chief}}$$

$$(\text{SUPERVISOR} \bowtie_{\text{Employee=Number}} (\sigma_{\text{Wage>40}}\ (\text{EMPLOYEE}))))$$

$$\{\ \text{NameC: } nc,\ \text{WageC: } wc \mid$$
$$\text{EMPLOYEE(Number: } m, \text{ Name: } n, \text{ Age: } a, \text{ Wage: } w) \ \wedge$$
$$w > 40 \ \wedge\ \text{SUPERVISOR(Chief: } c, \text{ Employee: } m) \ \wedge$$
$$\text{EMPLOYEE(Number: } c, \text{ Name: } nc, \text{ Age: } ac, \text{ Wage: } wc)\ \}$$

Example (4)

- Return the employees earning more money than their boss; for both such employees and chiefs return the number, name and salary

$$\pi_{\text{Number,Name,Wage,NumC,NameC,WageC}} (\sigma_{\text{Wage>WageC}}($$

$$\rho_{\text{NumC,NameC,WageC,AgeC} \leftarrow \text{Number,Name,Wage,Age}}(\text{EMPLOYEE})$$

$$\bowtie_{\text{NumC=Chief}}$$

$$(\text{SUPERVISOR} \bowtie_{\text{Employee=Number}} \text{EMPLOYEE})))$$

{ Number: *m*, Name: *n*, Wage: *w*, NumC: *c*, NameC: *nc*, WageC: *wc* |
EMPLOYEE(Number: *m*, Name: *n*, Age: *a*, Wage: *w*) ∧
SUPERVISOR(Chief: *c*, Employee: *m*) ∧
EMPLOYEE(Number: *c*, Name: *nc*, Age: *ac*, Wage: *wc*) ∧ *w* > *sc* }

Example (5)

- Return the chiefs' number and name having all employees earning more than 40

$$\pi_{\text{Number,Name}}\ (\text{EMPLOYEE} \bowtie_{\text{Number=Chief}}$$
$$(\pi_{\text{Chief}}\ (\text{SUPERVISOR}) -$$
$$\pi_{\text{Chief}}\ (\text{SUPERVISOR} \bowtie_{\text{Employee=Number}} \sigma_{\text{Wage}\leq 40}\ (\text{EMPLOYEE}))))$$

$$\{\ \text{Number}: c,\ \text{Name}: n\ |$$
$$\text{EMPLOYEE}(\text{Number}: c,\ \text{Name}: n,\ \text{Age}: a,\ \text{Wage}: w)\ \wedge$$
$$\text{SUPERVISOR}(\text{Chief}: c,\ \text{Employee}: m)\ \wedge$$
$$\neg\exists m'(\exists n'(\exists a'(\exists w'(\text{EMPLOYEE}(\text{Number}: m',\ \text{Name}: n',\ \text{Age}: a',\ \text{Wage}: w')\ \wedge$$
$$w' \leq 40\ \wedge\ \text{SUPERVISOR}(\text{Chief}: c,\ \text{Employee}: m')))))\ \}$$

Recap

- De Morgan rules:
  - $\neg(A \wedge B) = (\neg A) \vee (\neg B)$
  - $\neg(A \vee B) = (\neg A) \wedge (\neg B)$
- Moreover:
  - $\neg\forall x\, A = \exists x\, \neg A$
  - $\neg\exists x\, A = \forall x\, \neg A$
  - $\forall x\, A = \neg\exists x\, \neg A$
  - $\exists x\, A = \neg\forall x\, \neg A$
- Bonus:
  - $\neg A \vee B \rightarrow$ if A then B

Which Quantifier Shall We Use?

- Existential or universal? They are interchangeable by De Morgan

$$\{ \text{Number}: c, \text{Name}: n \mid$$
$$\text{EMPLOYEE}(\text{Number}: c, \text{Name}: n, \text{Age}: a, \text{Wage}: w) \wedge$$
$$\text{SUPERVISOR}(\text{Chief}: c, \text{Employee}: m) \wedge$$
$$\neg\exists m'(\exists n'(\exists a'(\exists w'(\text{EMPLOYEE}(\text{Number}: m', \text{Name}: n', \text{Age}: a', \text{Wage}: w') \wedge$$
$$\text{SUPERVISOR}(\text{Chief}: c, \text{Employee}: m') \wedge w' \leq 40)))) \}$$

$$\{ \text{Number}: c, \text{Name}: n \mid$$
$$\text{EMPLOYEE}(\text{Number}: c, \text{Name}: n, \text{Age}: a, \text{Wage}: w) \wedge$$
$$\text{SUPERVISOR}(\text{Chief}: c, \text{Employee}: m) \wedge$$
$$\forall m'(\forall n'(\forall a'(\forall w'(\neg\text{EMPLOYEE}(\text{Number}: m', \text{Name}: n', \text{Age}: a', \text{Wage}: w') \wedge$$
$$\text{SUPERVISOR}(\text{Chief}: c, \text{Employee}: m') \vee w' > 40)))) \}$$

On Domain Relational Calculi

- Pros:
  - Declarative
- Cons:
  - “Verbose” so many variables!
  - Meaningless expressions:

$$\{\, A{:}\ x \mid \neg\, R(A{:}\ x) \,\}$$

$$\{\, A{:}\ x,\ B{:}\ y \mid R(A{:}\ x) \,\}$$

$$\{\, A{:}\ x,\ B{:}\ y \mid R(A{:}\ x) \ \wedge\ y{=}y \,\}$$

    - such expressions are **domain dependant** and we shall avoid them
    - we cannot state such statements in relational algebra because it is domain independent

Domain Calculus vs. Algebra

- Domain Relational Calculus (DRC) and Relational Algebra (RA) are "**equivalent**"
  - For each domain independent DRC expression, there exists an equivalent RA expression.
  - For each RA expression there exists an equivalent domain independent DRC expression

Tuples Calculus with Range Declarations

- To overcome the domain calculus limitations:
  - We must reduce the number of variables. A good way to do so: we restrict the variables to the tuples, one variable for each tuple
  - All the data values must come from the database
- The **tuples calculus with range declarations** satisfies both needs

Tuples Calculus with Range Declarations Syntax

- The expressions have the following syntax:

{ *TargetList* | *RangeList* | *Formula* }

  - *TargetList* has elements like Y: *x*.Z (or *x*.Z or even *x*.*)
  - *RangeList* shows the free variables in *Formula* specifying from which relation they come from
  - *Formula* has
    - Comparison atoms $x.A \mathscr{R} c$, $x.A \mathscr{R} y.B$
    - Logical connectives
    - Quantifiers associating a range over the variables

$\exists x(R)(f)$ $\forall x(R)(f)$

Example (0a)

- Return the employees' number, name, age and salary earning more than 40

$$\sigma_{\text{Wage}>40} \text{ (EMPLOYEE)}$$

$$\{ \text{ Number: } m, \text{ Name: } n, \text{ Age: } a, \text{ Wage: } w \mid \text{EMPLOYEE(Number: } m, \text{ Name: } n, \text{ Age: } a, \text{ Wage: } w) \wedge w > 40 \}$$

DRC

$$\{ e.^* \mid e(\text{EMPLOYEE}) \mid e.\text{Wage} > 40 \}$$

TRC-RD

Example (0b)

- Return the employees' number, name and age for all

$$\pi_{\text{Number,Name,Age}}\ (\text{EMPLOYEE})$$

$$\{\ \text{Number: } m,\ \text{Name: } n,\ \text{Age: } a \mid \text{EMPLOYEE}(\text{Number: } m,\ \text{Name: } n,\ \text{Age: } a,\ \text{Wage: } w)\ \}$$

$$\{\ e.(\text{Number,Name,Age}) \mid e(\text{EMPLOYEE}) \mid\ \}$$

Example (1)

- Return the employees' number, name and age earning more than 40

$$\pi_{\text{Number,Name,Age}} (\sigma_{\text{Wage>40}} (\text{EMPLOYEE}))$$

$$\{ \text{Number: } m, \text{ Name: } n, \text{ Age: } a \mid \text{EMPLOYEE(Number: } m, \text{ Name: } n, \text{ Age: } a, \text{ Wage: } w) \wedge w > 40 \}$$

$$\{ e.\text{(Number,Name,Age)} \mid e\text{(EMPLOYEE)} \mid e.\text{Wage} > 40 \}$$

Example (2)

- Return the number identifying the employees' chiefs earning more than 40

{ Chief: *c* | SUPERVISOR(Chief: *c*, Employee: *e*) $\wedge$ EMPLOYEE(Number: *e*, Name: *n*, Age: *a*, Wage: *w*) $\wedge$ *w* > 40 }

{ *s*.Chief | *e*(EMPLOYEE), *s*(SUPERVISOR) | *e*.Number=*s*.Employee $\wedge$ *e*.Wage > 40 }

Example (3)

- Return the chiefs' name and salary having employees earning more than 40

$$\{\ \text{NameC}: nc,\ \text{WageC}: wc\ |$$
$$\text{EMPLOYEE}(\text{Number}: m,\ \text{Name}: n,\ \text{Age}: a,\ \text{Wage}: w)\ \wedge$$
$$w > 40\ \wedge\ \text{SUPERVISOR}(\text{Chief}: c,\ \text{Employee}: m)\ \wedge$$
$$\text{EMPLOYEE}(\text{Number}: c,\ \text{Name}: nc,\ \text{Age}: ac,\ \text{Wage}: wc)\ \}$$

$$\{\ \text{NameC},\text{WageC}: e'.(\text{Name},\text{Wage})\ |$$
$$e'(\text{EMPLOYEE}),\ s(\text{SUPERVISOR}),\ e(\text{EMPLOYEE})\ |$$
$$e'.\text{Number}{=}s.\text{Chief}\ \wedge\ s.\text{Employee}{=}e.\text{Number}\ \wedge\ e.\text{Wage} > 40\ \}$$

Example (4)

- Return the employees earning more money than their boss; for both such employees and chiefs return the number, name and salary

$$\{\ \text{Number: } m,\ \text{Name: } n,\ \text{Wage: } w,\ \text{NumC: } c,\ \text{NameC: } nc,\ \text{WageC: } wc \mid$$
$$\text{EMPLOYEE(Number: } m,\ \text{Name: } n,\ \text{Age: } a,\ \text{Wage: } w) \wedge$$
$$\text{SUPERVISOR(Chief: } c,\ \text{Employee: } m) \wedge$$
$$\text{EMPLOYEE(Number: } c,\ \text{Name: } nc,\ \text{Age: } ac,\ \text{Wage: } wc) \wedge w > wc\}$$

$$\{\ e.\text{(Name,Number,Wage), NameC,NumC,WageC: } e'.\text{(Name,Number,Wage)} \mid$$
$$e'\text{(EMPLOYEE)},\ s\text{(SUPERVISOR)},\ e\text{(EMPLOYEE)} \mid$$
$$e'.\text{Number}=s.\text{Chief} \wedge s.\text{Employee}=e.\text{Number} \wedge e.\text{Wage} > e'.\text{Wage}\ \}$$

Example (5)

- Return the chiefs' number and name having all employees earning more than 40

$$\{\ \text{Number: } c,\ \text{Name: } n \mid$$
$$\text{EMPLOYEE}(\text{Number: } c,\ \text{Name: } n,\ \text{Age: } a,\ \text{Wage: } w)\ \wedge$$
$$\text{SUPERVISOR}(\text{Chief: } c,\ \text{Employee: } m)\ \wedge$$
$$\neg\exists m'(\exists n'(\exists a'(\exists w'(\text{EMPLOYEE}(\text{Number: } m',\ \text{Name: } n',\ \text{Age: } a',\ \text{Wage: } w')\ \wedge$$
$$\text{SUPERVISOR}(\text{Chief: } c,\ \text{Employee: } m')\ \wedge\ w' \leq 40))))\ \}$$

$$\{\ e.(\text{Number, Name}) \mid s(\text{SUPERVISOR}),\ e(\text{EMPLOYEE}) \mid$$
$$s.\text{Chief}=e.\text{Number}\ \wedge\ \neg(\exists e'(\text{EMPLOYEE})(\exists s'(\text{SUPERVISOR})$$
$$(s.\text{Chief}=s'.\text{Chief}\ \wedge\ s'.\text{Employee}=e'.\text{Number}\ \wedge\ e'.\text{Wage} \leq 40)))\ \}$$

Remark

- Such calculus cannot express some important queries, such as unions:

$$R_1(AB) \cup R_2(AB)$$

- How could I express it through ranges? Is it possible with either one variable or two?
- On the other hand, we can express intersection and difference
- For such reasons SQL (using this calculus) has an explicit union operator, while the intersection and difference operators do not appear in all SQL instances

Calculus and Relational Algebra: Limits

- Both languages are basically equivalent: we can express a meaningful set of tuple operations with them
- Some queries, potentially useful, cannot be expressed:
  - We can **only extract values**, we cannot compute new values from them
  - Some interesting computations:
    - over each tuple (e.g., conversions, sums, differences, etc.)
    - over a set of tuples (e.g., summation, average, etc.)

    ➢ Such extensions are adopted in SQL, we will see them
  - Recursive queries, such as the **transitive closure**

Transitive Closure (1)

- The **transitive closure** R^+ of a binary relation R on a set X is the smallest relation on X that contains R and is transitive
- Given R a relation on A × A, the transitive closure is the relation R^+ such that

$$R^+ = \{ \langle x, y\rangle \mid \exists y_1, \ldots, y_n \in A,$$

$$n \geq 2,\ y_1 = x,\ y_n = y,$$

$$\langle y_i, y_{i+1}\rangle \in R,\ i=1, \ldots, n-1 \}$$

- If X is a set of airports and x R y means "*there is a direct flight from airport x to airport y*", the transitive closure x R^+ y means "*it is possible to fly from x to y in one or more flights*"

Transitive Closure (2)

SUPERVISOR(<u>Employee</u>, Chief)

- For each employee, return all its superiors (e.g., its chief, the chief of its chief, and so on)

Employee	Chief
Rossi	Lupi
Neri	Bruni
Lupi	Falchi

Employee	Superior
Rossi	Lupi
Neri	Bruni
Lupi	Falchi
Rossi	Falchi

Transitive Closure: How?

- We could use both joins with renaming in order to express such relations
- But:

Employee	Chief
Rossi	Lupi
Neri	Bruni
Lupi	Falchi
Falchi	Leoni

Employee	Superior
Rossi	Lupi
Neri	Bruni
Lupi	Falchi
Falchi	Leoni
Rossi	Falchi
Lupi	Leoni
Rossi	Leoni

Transitive Closures are Impossible

- In standard relational algebra we cannot express the transitive closure for each binary relation
- In such languages, in order to express the transitive closure, we have each time to re-create a different expression:
  - How many join we would need?
  - There is no limit on the number of joins that are required!

Datalog

- A database oriented logical programming language, which ancestor is Prolog
- It uses two different types of predicates:
  - **extensional**: database's relations
  - **intensional**: correspond to "views"
- Intensional predicates are defined through **rules**

Datalog: Syntax

- Rules:

  head $\leftarrow$ *body*

  - ***head*** is an atomic predicate (intensional)
  - ***body*** is a list (conjunction) of atomic predicates
- Queries are specified by atomic predicates with a **?** prefix

Preliminary Example

- Return the employees' number, name, age and salary being 30 years old

$$\{\ \text{Number: } m,\ \text{Name: } n,\ \text{Age: } a,\ \text{Wage: } w \mid \text{EMPLOYEE(Number: } m,\ \text{Name: } n,\ \text{Age: } a,\ \text{Wage: } w) \land a = 30\ \}$$

DRC

? EMPLOYEE(Number: *m*, Name: *n*, Age: 30, Wage: *w*)

DATALOG

Example (0a)

- Return the employees' number, name, age and salary earning more than 40

$$\{\ \text{Number: } m, \text{ Name: } n, \text{ Age: } a, \text{ Wage: } w \mid \text{EMPLOYEE(Number: } m, \text{ Name: } n, \text{ Age: } a, \text{ Wage: } w) \wedge w > 40\ \}$$

- We need an intensional predicate

$$\text{RICHER(Number: } m, \text{ Name: } n, \text{ Age: } a, \text{ Wage: } w) \leftarrow \text{EMPLOYEE(Number: } m, \text{ Name: } n, \text{ Age: } a, \text{ Wage: } w), w > 40$$

$$?\ \text{RICHER(Number: } m, \text{ Name: } n, \text{ Age: } a, \text{ Wage: } w)$$

Example (0b)

- Return the employees' number, name and age for all

$$\pi_{\text{Number,Name,Age}}\ (\text{EMPLOYEE})$$

{ Number: *m*, Name: *n*, Age: *a* |
EMPLOYEE(Number: *m*, Name: *n*, Age: *a*, Wage: *w*) }

PUBINFO(Number: *m*, Name: *n*, Age: *a*) ←
EMPLOYEE(Number: *m*, Name: *n*, Age: *a*, Wage: *w*)

? PUBINFO(Number: *m*, Name: *n*, Age: *a*)

Example (2)

- Return the number identifying the employees' chiefs earning more than 40

$$\{ \text{Chief: } c \mid \text{SUPERVISOR(Chief: } c\text{, Employee: } e) \wedge \text{EMPLOYEE(Number: } e\text{, Name: } n\text{, Age: } a\text{, Wage: } w) \wedge w > 40 \}$$

$$\text{CHIEFSOFRICHERS(Chief: } c) \leftarrow \text{EMPLOYEE(Number: } m\text{, Name: } n\text{, Age: } a\text{, Wage: } w), w > 40, \text{SUPERVISOR(Chief: } c\text{, Employee: } m)$$

$$?\ \text{CHIEFSOFRICHERS(Chief: } c)$$

Example (5)

- Return the chiefs' number and name having all employees earning more than 40
- We need negation

CHIEFSOFNORICHERS(Chief: *c*) ←
EMPLOYEE(Number: *m*, Name: *n*, Age: *a*, Wage: *w*),
w ≤ 40, SUPERVISOR(Chief: *c*, Employee: *m*)

CHIEFSOFONLYRICHER(Number: *c*, Name: *n*) ←
EMPLOYEE(Number: *c*, Name: *n*, Age: *a*, Wage: *w*),
SUPERVISOR(Chief: c, Employee: *m*),
NOT CHIEFSOFNORICHERS(Chief: *c*)

? CHIEFSOFONLYRICHER(Number: *c*, Name: *n*)

Example (6)

- For each employee, get all his/her chiefs
- We need recursion

HIGHGRADE(Employee: *e*, SuperChief: *c*) ←
SUPERVISOR(Employee: *e*, Chief: *c*)

HIGHGRADE(Employee: *e*, SuperChief: *c*) ←
SUPERVISOR(Employee: *e*, Chief: *c'*),
HIGHGRADE(Employee: *c'*, SuperChief: *c*)

Datalog: Semantics

- The definition of the recursive queries is quite tricky (in particular, the “negation” case)
- Expressive Power:
  - Non-recursive Datalog without negation is as expressive to the Calculus without negation and without universal quantifier
  - Non-recursive Datalog with negation is as expressive as calculus and algebra
  - We cannot compare Recursive Datalog without negation and Calculus
  - Recursive Datalog with negation is more expressive than calculus and algebra