Databases

Relational Algebra and Relational Calculus

ages

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

Query languages for relational DBs

- Declarative

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

- 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 j., cartesian product, θ-join)

Set operators

- Relations are sets
- The results must also be sets
- 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	Isacco

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 attribute “OldName” into “NewName”

Renaming, examples

Fatherhood

Father	Child
Adam	Abel
Adam	Cain
Abraham	Isaac

$\rho_{Parent \leftarrow Father}$ (Fatherhood)

Parent	Child
Adam	Abel
Adam	Cain
Abraham	Isaac

Renaming, examples

Fatherhood

Father	Child
Adam	Abel
Adam	Cain
Abraham	Isaac

$\rho_{Parent \leftarrow Father}$ (Fatherhood)

Parent	Child
Adam	Abel
Adam	Cain
Abraham	Isaac

Motherhood

Mother	Child
Eve	Abele
Eve	Seth
Sarah	Isaac

$\rho_{Parent \leftarrow Mother}$ (Motherhood)

Parent	Child
Eve	Abel
Eve	Seth
Sarah	Isaac

Renaming, examples

$\rho_{Parent \leftarrow Father}$ (Fatherhood)

Parent	Child
Adamo	Abele
Adamo	Caino
Abramo	Isacco

$\rho_{Parent \leftarrow Father}$ (Fatherhood) ∪
$\rho_{Parent \leftarrow Mother}$ (Motherhood)

$\rho_{Parent \leftarrow Mother}$ (Motherhood)

Parent	Child
Eva	Abele
Eva	Set
Sara	Isacco

Parent	Child
Adamo	Abele
Adamo	Caino
Abramo	Isacco
Eva	Abele
Eva	Set
Sara	Isacco

Employee

Surname	Office	Salary
Rossi	Rome	55
Neri	Milan	64

Worker

Surname	Factory	Wage
Bruni	Paris	45
Verdi	Berlin	55

$\rho_{\text{Office, Pay} \leftarrow \text{Office, Salary}}$ (Employee) ∪

$\rho_{\text{Office, Pay} \leftarrow \text{Factory, Wage}}$ (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 relations that satisfy the given predicate

Selection, examples

Employee

Number	Surname	Office	Salary
7309	Rossi	Rome	55
5998	Neri	Milan	64
9553	Milan	Milan	44
5698	Neri	Naples	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, examples (1)

- … 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	Naples	64

Selection, examples (2)

- …Employees earning more than 50 and working in Milan

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

Number	Surname	Office	Salary
5998	Neri	Milan	64

Selection, examples (3)

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

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

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:

$$\pi_{\text{AttributeList}}(\text{Relation})$$

- Semantics:

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

Projection, examples

Employee

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

- For all the employees return
  - Number and Surname
  - Surname and Office

Projection, examples (1)

- Return the Employees' number and surname

$$\Pi_{\text{Number, Surname}}(\text{Employee})$$

Number	Surname
7309	Neri
5998	Neri
9553	Rossi
5698	Rossi

Projection, examples (2)

- Surname and Office for all the Employees

$$\Pi_{\text{Surname, Office}}(\text{Employee})$$

Surname	Office
Neri	Naples
Neri	Milan
Rossi	Rome

Projections' cardinality

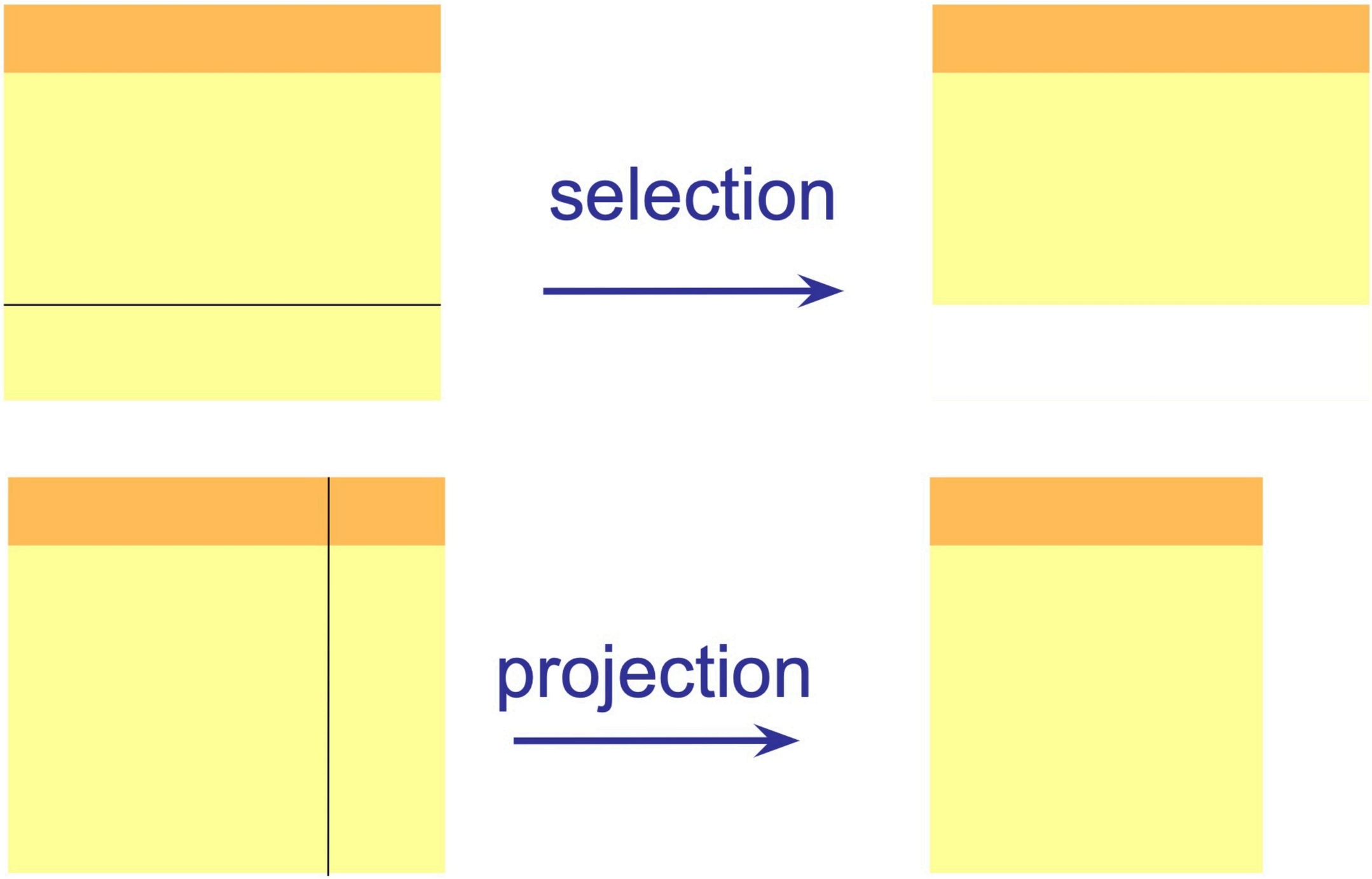
- A projection's output
  - contains at most the same number of tuples than the input's
  - could contain less tuples: restricting to a subset of the attributes, some tuples could be repeated! Repeated tuples are discarded in relations.

- if X is a superkey for R, then $\pi_X(R)$ 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

$$\Pi_{\text{Number,Surname}} \left(\sigma_{\text{Salary} > 50} (\text{Employees}) \right)$$

Number	Surname
7309	Rossi
5998	Neri
5698	Neri

Limits of the combo Selection+Projection

- By using such operators we could only extract informations from one single relation
- Hereby we cannot 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

1	25
2	13
3	27
4	28

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:
  - 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

$R_1(X_1)$, $R_2(X_2)$

$R_1 \bowtie R_2$ is a relation over X_1X_2

$$R1 \bowtie R2 = \{\, t \text{ on } X_1X_2 \mid$$

$$\exists t_1 \in R_1 \text{ and } t_2 \in R_2 \text{ with } t[X_1] = t_1 \text{ and } t[X_2] = t_2 \}$$

Full Join

Employee	Dept.
Rossi	A
Neri	B
Bianchi	B

⋈

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

⋈

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

⋈

Dept.	Chief
D	Mori
C	Bruni

Employee	Dept.	Chief

A full join, having n×m tuples

Employee	Dept.
Rossi	B
Neri	B

⋈

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, example (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

Room	Dept.
1	B
2	B
3	C

$|R_1 \bowtie R_2| = 4$

Size of a join's result, example (2)

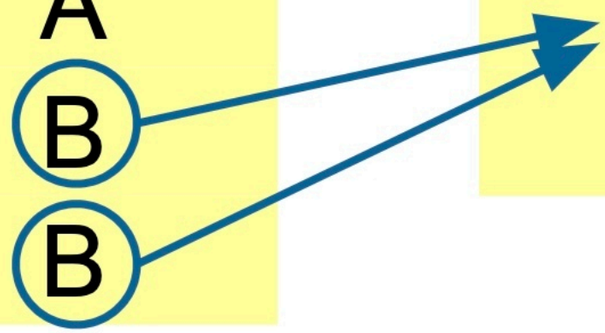
- The size of the join between R_1 and R_2 involves R_2 's key.
- Hence its size is between 0 and $|R_1|$
- Since the key's value are unique, for each tuple of R_2 can match more tuples of R_1 and not vice versa

R_1

Employee	Dept.
Rossi	A
Neri	B
Bianchi	B

R_2

Dept.	Chief
B	Mori
C	Bruni



Size of a join's result, example (3)

- If there exists a referential integrity constraint between Dept. (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

Dept.	Chief
A	Bruni
B	Mori
C	Russo

Join, critical issues

Employee	Dept.
Rossi	A
Neri	B
Bianchi	B

⋈

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, right, and full.

Outer Join

- left (⟕): 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 (⟖): A⟖ B is defined as B⟕ A
- full (⟗): 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

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

- It's a Natural Join over two relations with no common attributes.
- 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 natural join 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

$$R_1 \bowtie_{condition} R_2$$

Theta-join (2)

- 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.

Employee

Employee	Dept.
Rossi	A
Neri	B
Bianchi	B

Department

Code	Chief
A	Mori
B	Bruni

Employee $\bowtie_{\text{Dept.=Code}}$ Department

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

Please Note

- The equi-join provides a similar result as the join between Employee and Department where "Code" is renamed as "Dept."

$$\text{Employee} \bowtie \rho_{\text{Dept.} \leftarrow \text{Code}} (\text{Department})$$

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

Natural Join and EquiJoin

- 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

$$\text{Employee} \bowtie \text{Department}$$

$$\pi_{\text{Employee,Dept.,Chief}} (\sigma_{\text{Department=Code}} (\text{Employee} \bowtie \rho_{\text{Code} \leftarrow \text{Dept.}} (\text{Department})))$$

Examples

Employee

Code	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

$$\sigma_{Wage>40}(Employee)$$

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 employees' chiefs earning more than 40

$$\pi_{Chief}(Supervisor \bowtie_{Employee=Number} (\sigma_{Salary>40}(Employee)))$$

Relational Algebra: Examples (4)

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

$$\pi_{Name,Wage}\Big(\text{Employee} \bowtie_{Number=Chief} \pi_{Chief}\big(\text{Supervisor} \bowtie_{Employee=Number} \sigma_{Wage>40}(\text{Employee})\big)\Big)$$

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

$$\pi_{\text{Code,Name,Wage,NoC,NameC,WageC}}\Big(\sigma_{\text{Salary>WageC}}\Big(\rho_{\text{NoC,NameC,WageC,AgeC} \leftarrow \text{Number,Name,Wage,Age}}(\text{Employee}) \bowtie_{\text{NoC=Chief}} (\text{Supervisor} \bowtie_{\text{Employee=Number}} \text{Employee})\Big)\Big)$$

Relational Algebra: Examples (6)

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

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

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.
- Equivalence rules are very important because modern DBMSs try to rewrite the queries into equivalent (but more efficient, i.e. less time consuming) expressions

Equivalent Relational Algebra Expressions (2)

1. Combining a cascade of selections:

$$\sigma_{A>20}(\sigma_{B='Jones'}(R)) = \sigma_{A>20 \text{ AND } B='Jones'}(R)$$

2. Projection's idempotency (X and Y belong to R's schema)

$$\pi_{X}(R) = \pi_{X}(\pi_{XY}(R))$$

Equivalent Relational Algebra Expressions (3)

3. Push selections (A belongs to R_2's schema)

$$\sigma_{A=10}(R_1 \bowtie R_2) = R_1 \bowtie (\sigma_{A=10}(R_2))$$

It drastically reduces the intermediate result's size (therefore also the operation cost).

Equivalent Relational Algebra Expressions (4)

4. Pushing projections

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

- 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 Relational Algebra Expressions (5)

5. Using the theta-join definition

a. $\sigma_{A=10}(R_1 \bowtie R_2) = R_1 \bowtie_{A=10} R_2$

b. $\sigma_{A=10 \text{ AND } B='Jones'}(R_1 \bowtie R_2) =$

$\sigma_{A=10}(R_1 \bowtie_{B='Jones'} R_2) =$

$\sigma_{B='Jones'}(R_1 \bowtie_{A=10} R_2)$

Equivalent Relational Algebra Expressions (6)

- Some distributive properties

6. $\sigma_{A=10}(R_1 \cup R_2) = \sigma_{A=10}(R_1) \cup \sigma_{A=10}(R_2)$

7. $\sigma_{A=10}(R_1 - R_2) = \sigma_{A=10}(R_1) - \sigma_{A=10}(R_2)$

8. $\pi_{X}(R_1 \cup R_2) = \pi_{X}(R_1) \cup \pi_{X}(R_2)$

- Please note that projection is not distributive over difference

Equivalent Relational Algebra Expressions (7)

- Alcune proprietà degli insiemi e della selezione

9. $\sigma_{A=10 \text{ OR } B='Jones'}(R) = \sigma_{A=10}(R) \cup \sigma_{B='Jones'}(R)$
10. $\sigma_{A=10 \text{ AND } B='Jones'}(R) = \sigma_{A=10}(R) \cap \sigma_{B='Jones'}(R)$
11. $\sigma_{A=10 \text{ AND } \neg B='Jones')}(R) = \sigma_{A=10}(R) - \sigma_{='Jones'}(R)$

12. Commutativity for the union operator

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

Example

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

Example (1)

- Provide all the Chief's numbers being less than 30 years old.

$$\pi_{\text{Chief}}(\sigma_{\text{No=Emp AND Age<30}}(\text{Employee} \bowtie \text{Supervisor}))$$

Example (2)

- Rule №1, split the selection

  $\pi_{Chief}(\sigma_{No=Emp}(\sigma_{Age<30}(Employee \bowtie Supervisor)))$

- Rule №5 for the *No=Emp* selection and №3 for the *age<30* selection

$\pi_{Chief}(\sigma_{Age<30}(Employee) \bowtie_{No=Emp} Supervisor)$

Example (3)

- Rule №4, delete all the unused attributes

$$\pi_{Chief}(\pi_{No}(\sigma_{Age<30}(Employee)) \bowtie_{No=Emp} Supervisor)$$

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

$$\sigma_{\text{Age} > 40}(\text{Employee})$$

Employee

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

- No NULL value satisfies this specific atomic condition

Non Desiderata

- Selections are evaluated separately:

$$\sigma_{Age>30}(People) \cup \sigma_{Age\leq 30}(People) \neq People$$

- Atomic conditions are evaluated separately

$$\sigma_{Age>30\ OR\ Age\leq 30}(People) \neq People$$

NULL Valued selections (1)

- The following atomic condition is satisfied only by non-NULL values: $\sigma_{\text{Age} > 40}(\text{People})$
- 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 (2)

Hereby:

$$\sigma_{Age>30}(People) \cup \sigma_{Age\leq 30}(People) \cup \sigma_{Age\ IS\ NULL}(People)$$

$$=$$

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

$$=$$

$$People$$

NULL Valued selections (3)

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

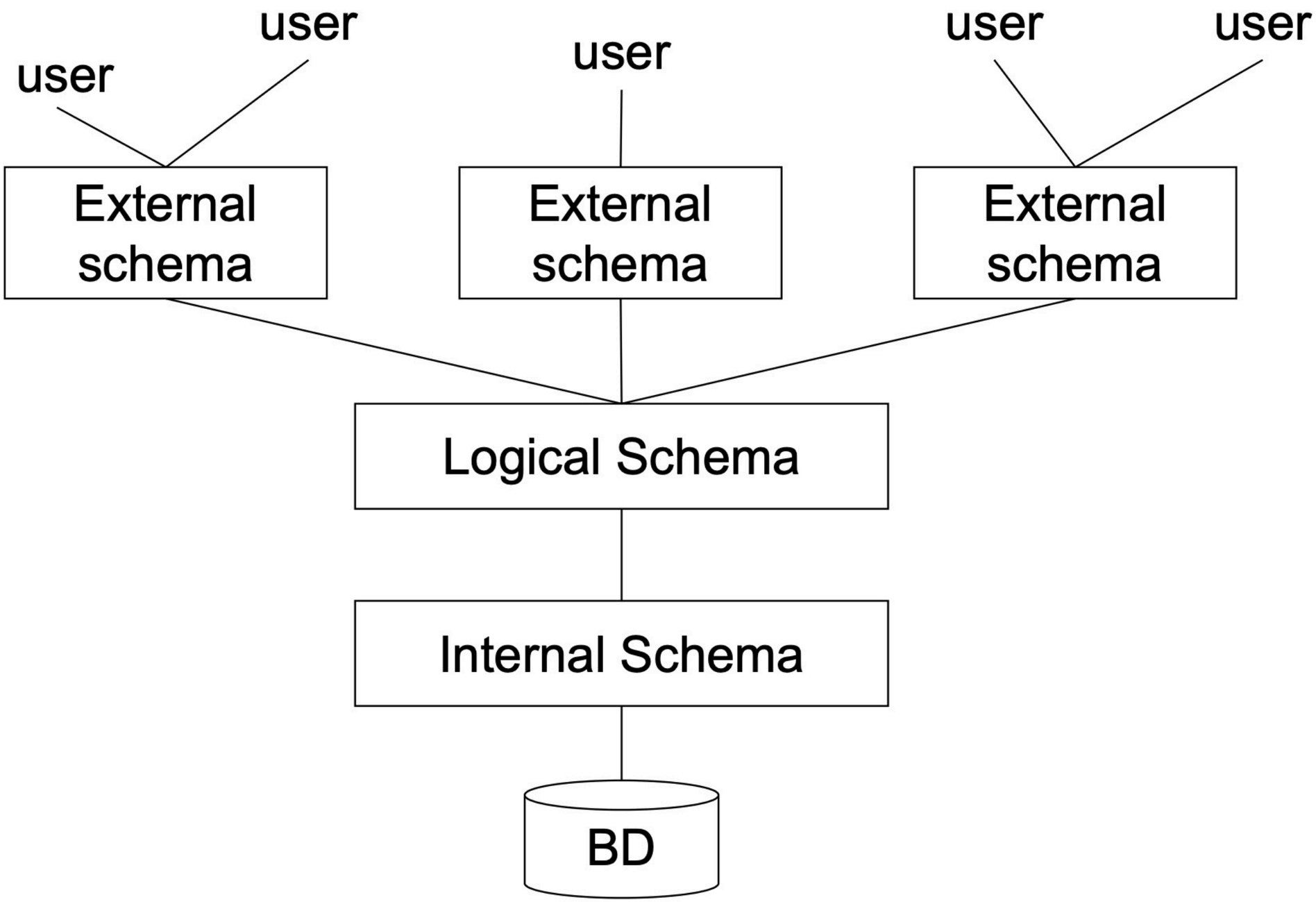
Employee

Number	Surname	Agency	Age
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, Example

Concern

Employee	Dept.
Rossi	A
Neri	B
Bianchi	B

Manage

Dept.	Chief
A	Mori
B	Bruni

- Create a view:

$$\text{Supervisor} = \pi_{\text{Employee, Chief}} (\text{Concern} \bowtie \text{Manage})$$

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
  - 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:

$$\sigma_{\text{Chief='Leoni'}}(\text{Supervisor})$$

Is run as:

$$\sigma_{\text{Chief='Leoni'}}(\pi_{\text{Employee, Chief}}(\text{Concern} \bowtie \text{Manage}))$$

Views, reasons

- *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
- *Tool in programming*:
  - we could simplify the writing of complex queries when sub-expressions are repeated
- Using already-existent software over pre-defined schemas

But:

- Views do not affect queries' efficiency

Views as a programming tool

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

$$\pi_{\text{Employee}}((\text{Concern} \bowtie \text{Manage}) \bowtie \rho_{\text{ImpR,RepR} \leftarrow \text{Emp,Dept}}(\sigma_{\text{Employee='Jones'}}(\text{Concern} \bowtie \text{Manage})))$$

  - Using views:

$$\pi_{\text{Employee}}(\text{Supervisor} \bowtie \rho_{\text{ImpR} \leftarrow \text{Imp}}(\sigma_{\text{Employee='Jones'}}(\text{Supervisor})))$$

Updating views

Concern

Employee	Dept
Rossi	A
Neri	B
Verdi	A

Manage

Dept	Chief
A	Mori
B	Bruni
C	Bruni

Supervisor

Employee	Chief
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 joins (1)

- 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).

Conventions and notations (1)

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

$$\pi_{No,Name,Wage,NoC,NameC,WageC}\Big(\sigma_{Wage>WageC}\Big(\rho_{NoC,NameC,WageC,AgeC \leftarrow No,Name,Wage,Age}(Emp) \bowtie_{NoC=Chief} (Sup \bowtie_{Emp=No} Emp)\Big)\Big)$$

Conventions and notations (2)

- Assign Emp to Chief (View)

Chief:= Emp

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

e.g. Emp.Wage and Chief.Wage

$$\pi_{\text{Emp.No, Emp.Name, Emp.Wage,Chief.No,Chief.Name, Chief.Wage}}\Big(\sigma_{\text{Emp.Wage>Chief.Wage}}\big(\text{Chief} \bowtie_{\text{Chief.No=Chief}} (\text{Sup} \bowtie_{\text{Emp=Emp.No}} \text{Emp})\big)\Big)$$

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

- An expression is in the form:

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

  - f is a formula (with boolean operators and quantifiers)
  - $A_1: x_1, \ldots, A_k: x_k$ target list:
    - $A_1, \ldots, A_k$ different attributes (can be in different databases)
    - $x_1, \ldots, x_k$ different variables

- 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(Number,Name, Age, Salary)

Supervisor(Chief, Employee)

Example (0a)

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

$$\sigma_{Salary>40}(Employee)$$

$$\{ \text{Number: } m, \text{ Name: } n, \text{ Age: } a, \text{ Salary: } s \mid \text{Employee(Number: } m, \text{ Name: } n, \text{Age: } a, \text{ Salary: } s) \wedge s > 40 \}$$

Example (0b)

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

$$\Pi_{\text{Number, Name, Age}}(\text{Employee})$$

$$\{ \text{Number: } m, \text{ Name: } n, \text{ Age: } a \mid$$
$$\exists s\ (\text{Employee(Number: } m, \text{ Name: } n, \text{ Age: } a, \text{ Salary: } s)\}$$

$$\{ \text{Number: } m, \text{ Name: } n, \text{ Age: } a \mid$$
$$\text{Employee(Number: } m, \text{ Name: } n, \text{ Age: } a, \text{ Salary: s)}\}$$

Example (1)

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

$$\pi_{\text{Number, Name, Age}}(\sigma_{\text{Salary>40}}(\text{Employee}))$$

$$\{ \text{Number: } m, \text{Name: } n, \text{Age: } a \mid \text{Employee}(\text{Number: } m, \text{Name: } n, \text{Age: } a, \text{Salary: } s) \wedge s > 40 \}$$

Example (2)

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

$$\pi_{Chief}(Supervisor \bowtie_{Employee=Number} \sigma_{Salary>40}(Employee))$$

$$\{ Chief: c \mid Supervisor(Chief:c, Employee:e) \wedge Employee(Number: e, Name: n, Age: a, Salary: s) \wedge s > 40 \}$$

Example (3)

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

$$\pi_{NameC,WageC}(\rho_{NoC,NameC,WageC,AgeC \leftarrow No,Name,Wage,Age}(Employee) \bowtie_{NoC=Chief} (Supervisor \bowtie_{Employee=Number} (\sigma_{Salary>40}(Employee))))$$

$$\{ NameC: nc, WageC: sc \mid Employee(Number: m, Name: n, Age: a, Salary: s) \wedge s > 40 \wedge Supervisor(Chief:c,Employee:m) \wedge Employee(Number:c, Name:nc, Age:ac, Salary: sc) \}$$

Example (4)

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

$$\pi_{No,Name,Wage,NoC,NameC,WageC}(\sigma_{Salary>WageC}(\rho_{NoC,NameC,WageC,AgeC \leftarrow No,Name,Wage,Age}(Employee) \bowtie_{NoC=Chief} (Supervisor \bowtie_{Employee=Number} ((Employee)))))$$

$$\{ No: m, Name: n, Wage: s, NoC: c, NameC: nc, WageC: sc \mid Employee(Number: m, Name: n, Age: a, Salary: s) \wedge Supervisor(Chief:c,Employee:m) \wedge Employee(Number: c, Name: nc, Age: ac, Salary: sc) \wedge s > sc\}$$

Example (5)

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

$$\pi_{Number,Name}(Employee \bowtie_{Number=Chief} (\pi_{Chief}(Supervisor) - \pi_{Chief}(Supervisor \bowtie_{Employee=Number} (\sigma_{Salary \leq 40}(Employee)))))$$

$$\{Number: c, Name: n \mid Employee(Number: c, Name: n, Age: e, Salary: s) \wedge Supervisor(Chief:c, Employee:m) \wedge \neg \exists m'(\exists n'(\exists e'(\exists s'(Employee(No: m', Name: n', Age: e', Wage: s') \wedge Supervisor(Chief:c, Employee:m') \wedge s' \leq 40))))\}$$

Recap

- De Morgan rules:
  - $\neg(A \wedge B) = (\neg A) \vee (\neg B)$
  - $\neg(A \vee B) = (\neg A) \wedge (\neg B)$
- Moreover:
  - $\neg\ \exists x\ A = \forall x\ A$
  - $\neg\ \forall x\ A = \exists x\ A$
  - $\exists x\ A = \neg \forall x\ \neg A$

Which quantifier shall we use?

- Existential or universal? They are interchangeable by De Morgan:

$$\{\text{Number: c, Name: n} \mid \text{Employee(Number: c, Name: n, Age: e, Salary: s)} \wedge \text{Supervisor(Chief:c, Employee:m)} \wedge \neg\, \exists m'(\exists n'(\exists e'(\exists s'(\text{Employee(No: m', Name: n', Age: e', Wage: s')} \wedge \text{Supervisor(Chief:c, Employee:m')} \wedge s' \leq 40))))\}$$

$$\{\text{Number: c, Name: n} \mid \text{Employee(Number: c, Name: n, Age: e, Salary: s)} \wedge \text{Supervisor(Chief:c, Employee:m)} \wedge \forall m'(\forall n'(\forall e'(\forall s'(\neg(\text{Employee(No:m', Name:n', Age:e', Wage:s')} \wedge \text{Supervisor(Chief:c, Employee:m')}) \vee s' > 40))))\}$$

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

- We want 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 proposed calculus satisfies both needs

Tuple calculus with range declarations syntax

- The expressions have the following syntax:

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

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

      ∃x(R)(...) ∀x(R)(...)

Example (0a)

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

$$\sigma_{Salary>40}(Employee)$$

$$\{ \text{Number: m, Name: n, Age: e, Salary: s} \mid \text{Employee(Number: m, Name: n, Age: e, Salary: s)} \wedge s > 40 \}$$

$$\{ i.* \mid i(Employee) \mid i.Salary > 40 \}$$

Example (0b)

- Return the employees' number, name and age

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

{ Number: m, Name: n, Age: e |
Employee(Number: m, Name: n, Age: e, Salary: s)}

{ i.(Number,Name,Age) | i(Employee) | }

Example (1)

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

$$\pi_{\text{Number, Name, Age}}(\sigma_{\text{Salary>40}}(\text{Employee}))$$

$$\{ \text{Number: m, Name: n, Age: e} \mid \text{Employee(Number: m, Name: n, Age: e, Salary: s)} \wedge \text{s} > 40 \}$$

$$\{ \text{i.(Number,Name,Age)} \mid \text{i(Employee)} \mid \text{i.Salary} > 40 \}$$

Example (2)

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

{ Chief: c | Supervisor(Chief:c,Employee:m) ∧ Employee(Number: m, Name: n, Age: e, Salary: s) ∧ s > 40 }

{ s.Chief | i(Employee) , s(Supervisor) | i.Number=s.Employee ∧ i.Salary > 40 }

Example (3)

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

{ NameC: nc, WageC: sc |
Employee(Number: m, Name: n, Age: e, Salary: s) ∧ s > 40 ∧
Supervisor(Chief:c,Employee:m) ∧
Employee(Number:c, Name:nc, Age:ec, Salary:sc) }

{ NameC,WageC: i'.(Name,Wage) |
i'(Employee), s(Supervisor), i(Employee) |
i'.Number=s.Chief ∧ i.Number=s.Employee ∧ i.Salary > 40 }

Example (4)

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

$$\{ \text{No: m, Name: n, Wage: s, NameC: nc, WageC: sc} \mid$$
$$\text{Employee(Number: m, Name: n, Age: e, Salary: s)} \wedge$$
$$\text{Supervisor(Chief:c,Employee:m)} \wedge$$
$$\text{Employee(Number: c, Name: nc, Age: ec, Salary: sc)} \wedge$$
$$\text{s} > \text{sc}\}$$

$$\{ \text{i.(Name,No,Wage), NameC,NoC,WageC: i'.(Name,No,Wage)} \mid$$
$$\text{i'(Employee), s(Supervisor), i(Employee)} \mid$$
$$\text{i'.Number=s.Chief} \wedge \text{i.Number=s.Employee} \wedge \text{i.Salary} > \text{i'.Salary} \}$$

Example (5)

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

$$\{\text{Number: c, Name: n} \mid$$
$$\text{Employee(Number: c, Name: n, Age: e, Salary: s)} \wedge$$
$$\text{Supervisor(Chief:c, Employee:m)} \wedge$$
$$\neg\, \exists m'(\exists n'(\exists e'(\exists s'(\text{Employee(No: m', Name: n', Age: e', Wage: s')} \wedge$$
$$\text{Supervisor(Chief:c, Employee:m')} \wedge s' \leq 40\}$$

$$\{\ i.(\text{Number, Name}) \mid s(\text{Supervisor}), i(\text{Employee}) \mid$$
$$i.\text{Number}=s.\text{Chief} \wedge \neg(\exists i'(\text{Employee})(\exists s'(\text{Supervisor})$$
$$(s.\text{Chief}=s'.\text{Chief} \wedge s'.\text{Employee}=i'.\text{Number} \wedge i'.\text{Salary} \leq 40)))\}$$

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.

Tuple Calculus and Relational Algebra: Cons

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

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

Transitive Closure

Supervisor(Employee, 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	Superiors
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

$$\{ \text{Number: m, Name: n, Age: e, Salary: s} \mid \text{Employee(Number: m, Name: n, Age: e, Salary: s)} \wedge e = 30 \}$$

? Employee(Number: m, Name: n, Age: 30, Salary: s)

Example (0a)

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

{ Number: m, Name: n, Age: e, Salary: s |
Employee(Number: m, Name: n, Age: e, Salary: s) ∧ s > 40 }

- We need an intensional predicate

RichE(Number: m, Name: n, Age: e, Salary: s) ← Employee(Number: m, Name: n, Age: e, Salary: s) , s >40

? RichE(Number: m, Name: n, Age: e, Salary: s)

Example (0b)

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

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

{ Number: m, Name: n, Age: e |
Employee(Number: m, Name: n, Age: e, Salary: s)}

PublicInfos(Number: m, Name: n, Age: e)
← Employee(Number: m, Name: n, Age: e, Salary: s)

? PublicInfos(Number: m, Name: n, Age: e)

Example (2)

- Return the chief's number having employees earning more than 40

{ Chief: c | Supervisor(Chief:c,Employee:m) ∧
Employee(Number: m, Name: n, Age: e, Salary: s) ∧ s > 40 }

ChiefsOfRichE (Chief:c) ←
Employee(Number: m, Name: n, Age: e, Salary: s), s > 40,
Supervisor (Chief:c,Employee:m)

? ChiefsOfRichE (Chief:c)

Example (5)

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

- We need the negation

ChiefsOfNoRichE (Chief:c) ←
Supervisor (Chief:c,Employee:m),
Employee (Number: m, Name: n, Age: e, Salary: s),
s ≤ 40

ChiefsOfOnlyRichE(Number: c, Name: n) ←
Employee (Number: c, Name: n, Age: e, Salary: s) ,
Supervisor (Chief:c,Employee:m),
not ChiefsOfNoRichE (Chief:c)

? ChiefsOfOnlyRichE (Number: c, Name: n)

Example (6)

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

HighGrade (Employee: i, SuperChief: c) ←
Supervisor (Employee: i, Chief: c)

HighGrade (Employee: i, SuperChief: c) ←
Supervisor (Employee: i, 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