



**Atzeni, Ceri, Fraternali,  
Paraboschi, Torlone**

**Basi di dati**

**Quarta edizione**

**McGraw-Hill, 2013**

**Capitolo 3:**

**ALGEBRA E CALCOLO  
RELAZIONALE**

# Linguaggi per basi di dati



- operazioni sullo schema
  - DDL: data definition language
- operazioni sui dati
  - DML: data manipulation language
    - interrogazione ("query")
    - aggiornamento

# Linguaggi di interrogazione per basi di dati relazionali



- **Dichiarativi**
  - specificano le proprietà del risultato ("che cosa")
- **Procedurali**
  - specificano le modalità di generazione del risultato ("come")

# Linguaggi di interrogazione

- **Algebra relazionale**: procedurale
- **Calcolo relazionale**:  
dichiarativo (teorico)
- **SQL** (Structured Query Language):  
parzialmente dichiarativo (reale)
- **QBE** (Query by Example):  
dichiarativo (reale)

# Algebra relazionale



- Insieme di operatori
  - su relazioni
  - che producono relazioni
  - e possono essere composti

# Operatori dell'algebra relazionale



- unione, intersezione, differenza
- ridenominazione
- selezione
- proiezione
- join (join naturale, prodotto cartesiano, theta-join)

# Operatori insiemistici

- le relazioni sono insiemi
- i risultati debbono essere relazioni
- è possibile applicare **unione**, **intersezione**, **differenza** solo a relazioni definite sugli stessi attributi

# Unione

## Laureati

Matricola	Nome	Età
7274	Rossi	42
7432	Neri	54
9824	Verdi	45

## Specialisti

Matricola	Nome	Età
9297	Neri	33
7432	Neri	54
9824	Verdi	45

## Laureati $\cup$ Specialisti

Matricola	Nome	Età
7274	Rossi	42
7432	Neri	54
9824	Verdi	45
9297	Neri	33



# Intersezione

## Laureati

Matricola	Nome	Età
7274	Rossi	42
7432	Neri	54
9824	Verdi	45

## Specialisti

Matricola	Nome	Età
9297	Neri	33
7432	Neri	54
9824	Verdi	45

## Laureati $\cap$ Specialisti

Matricola	Nome	Età
7432	Neri	54
9824	Verdi	45

# Differenza

## Laureati

Matricola	Nome	Età
7274	Rossi	42
7432	Neri	54
9824	Verdi	45

## Specialisti

Matricola	Nome	Età
9297	Neri	33
7432	Neri	54
9824	Verdi	45

## Laureati – Specialisti

Matricola	Nome	Età
7274	Rossi	42

# Un'unione sensata ma impossibile

## Paternità

Padre	Figlio
Adamo	Abele
Adamo	Caino
Abramo	Isacco

## Maternità

Madre	Figlio
Eva	Abele
Eva	Set
Sara	Isacco

Paternità  $\cup$  Maternità  
??

# Ridenominazione



- operatore monadico (con un argomento)
- "modifica lo schema" lasciando inalterata l'istanza dell'operando

## Paternità

Padre	Figlio
Adamo	Abele
Adamo	Caino
Abramo	Isacco

REN<sub>Genitore ← Padre</sub> (Paternità)

Genitore	Figlio
Adamo	Abele
Adamo	Caino
Abramo	Isacco

## Paternità

Padre	Figlio
Adamo	Abele
Adamo	Caino
Abramo	Isacco

## REN<sub>Genitore ← Padre</sub> (Paternità)

Genitore	Figlio
Adamo	Abele
Adamo	Caino
Abramo	Isacco

## Maternità

Madre	Figlio
Eva	Abele
Eva	Set
Sara	Isacco

## REN<sub>Genitore ← Madre</sub> (Maternità)

Genitore	Figlio
Eva	Abele
Eva	Set
Sara	Isacco

$REN_{\text{Genitore} \leftarrow \text{Padre}}$  (Paternità)

Genitore	Figlio
Adamo	Abele
Adamo	Caino
Abramo	Isacco

$REN_{\text{Genitore} \leftarrow \text{Padre}}$  (Paternità)



$REN_{\text{Genitore} \leftarrow \text{Madre}}$  (Maternità)

$REN_{\text{Genitore} \leftarrow \text{Madre}}$  (Maternità)

Genitore	Figlio
Eva	Abele
Eva	Set
Sara	Isacco

Genitore	Figlio
Adamo	Abele
Adamo	Caino
Abramo	Isacco
Eva	Abele
Eva	Set
Sara	Isacco

## Impiegati

Cognome	Ufficio	Stipendio
Rossi	Roma	55
Neri	Milano	64

## Operai

Cognome	Fabbrica	Salario
Bruni	Monza	45
Verdi	Latina	55

REN  $\text{Sede, Retribuzione} \leftarrow \text{Ufficio, Stipendio}$  (Impiegati)

REN  $\text{Sede, Retribuzione} \leftarrow \text{Fabbrica, Salario}$  (Operai)

Cognome	Sede	Retribuzione
Rossi	Roma	55
Neri	Milano	64
Bruni	Monza	45
Verdi	Latina	55



# Selezione



- operatore monadico
- produce un risultato che
  - ha lo stesso schema dell'operando
  - contiene un sottoinsieme delle ennuple dell'operando,
  - quelle che soddisfano una condizione

## Impiegati

Matricola	Cognome	Filiale	Stipendio
7309	Rossi	Roma	55
5998	Neri	Milano	64
9553	Milano	Milano	44
5698	Neri	Napoli	64

- impiegati che
  - guadagnano più di 50
  - guadagnano più di 50 e lavorano a Milano
  - hanno lo stesso nome della filiale presso cui lavorano

# Selezione, sintassi e semantica

- sintassi

**SEL** *Condizione* (*Operando*)

- *Condizione*: espressione booleana (come quelle dei vincoli di ennupla)
- semantica
  - il risultato contiene le ennuple dell'operando che soddisfano la condizione

- impiegati che guadagnano più di 50

**SEL<sub>Stipendio > 50</sub> (Impiegati)**

Matricola	Cognome	Filiale	Stipendio
7309	Rossi	Roma	55
5998	Neri	Milano	64
5698	Neri	Napoli	64

**SEL<sub>Stipendio > 50</sub> (Impiegati)**

- impiegati che guadagnano più di 50 e lavorano a Milano

**SEL**<sub>Stipendio > 50 AND Filiale = 'Milano'</sub> (Impiegati)

Matricola	Cognome	Filiale	Stipendio
5998	Neri	Milano	64

**SEL**<sub>Stipendio > 50 AND Filiale = 'Milano'</sub> (Impiegati)

- impiegati che hanno lo stesso nome della filiale presso cui lavorano

**SEL**  $\text{Cognome} = \text{Filiale}$  (**Impiegati**)

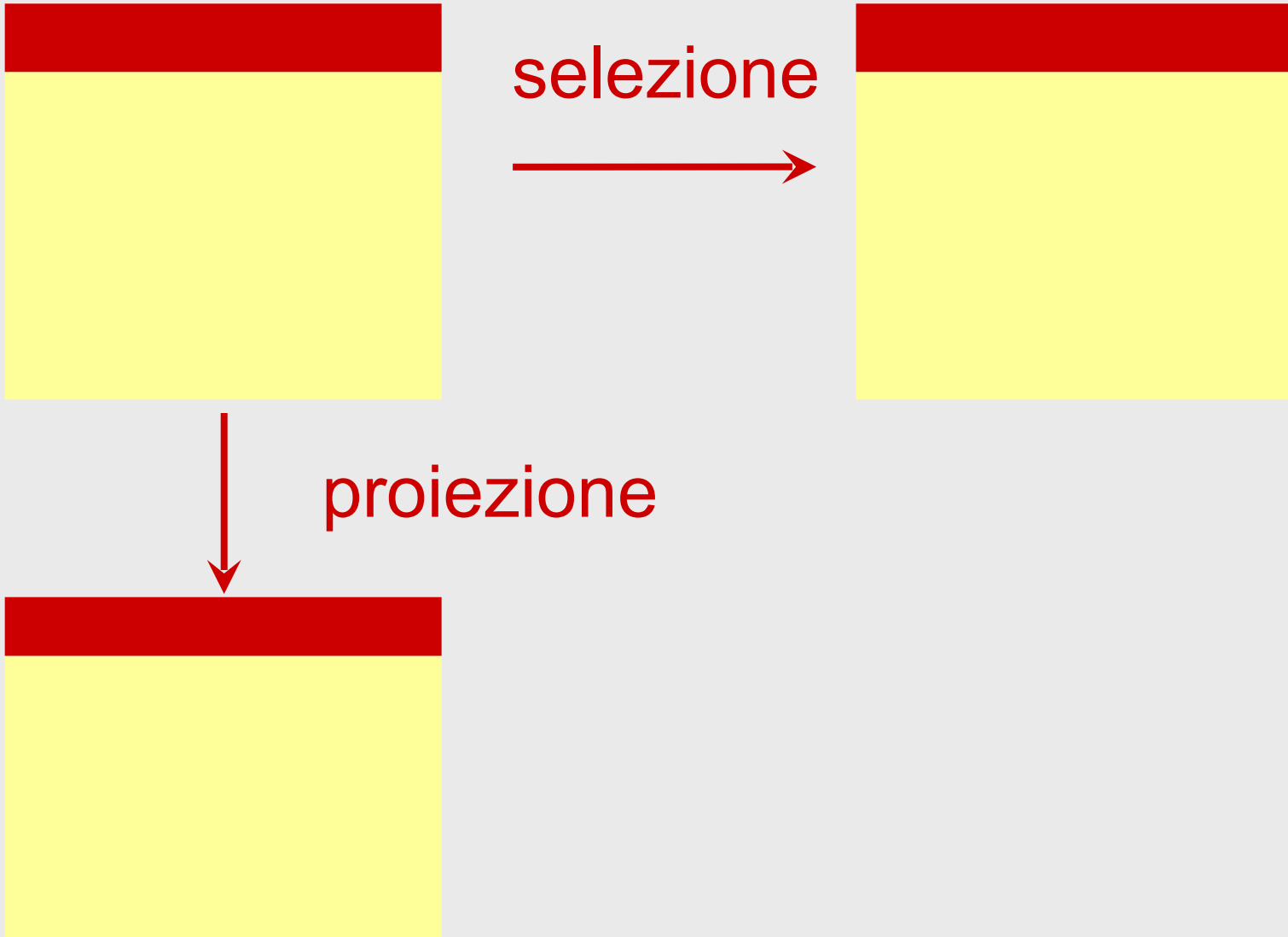
Matricola	Cognome	Filiale	Stipendio
9553	Milano	Milano	44

**SEL**  $\text{Cognome} = \text{Filiale}$  (**Impiegati**)

# Selezione e proiezione



- operatori "ortogonali"
- **selezione:**
  - decomposizione orizzontale
- **proiezione:**
  - decomposizione verticale





# Proiezione

- operatore monadico
- produce un risultato che
  - ha parte degli attributi dell'operando
  - contiene ennuple cui contribuiscono tutte le ennuple dell'operando

## Impiegati

Matricola	Cognome	Filiale	Stipendio
7309	Neri	Napoli	55
5998	Neri	Milano	64
9553	Rossi	Roma	44
5698	Rossi	Roma	64

- per tutti gli impiegati:
  - matricola e cognome
  - cognome e filiale

# Proiezione, sintassi e semantica

- sintassi

**PROJ**<sub>ListaAttributi</sub> (Operando)

- semantica
  - il risultato contiene le ennuple ottenute da tutte le ennuple dell'operando ristrette agli attributi nella lista

- matricola e cognome di tutti gli impiegati

Matricola	Cognome
7309	Neri
5998	Neri
9553	Rossi
5698	Rossi

**PROJ** *Matricola, Cognome* **(Impiegati)**

- cognome e filiale di tutti gli impiegati

Cognome	Filiale
Neri	Napoli
Neri	Milano
Rossi	Roma

**PROJ** Cognome, Filiale **(Impiegati)**

# Cardinalità delle proiezioni

- una proiezione
  - contiene al più tante ennuple quante l'operando
  - può contenerne di meno
- se  $X$  è una superchiave di  $R$ , allora  $PROJ_X(R)$  contiene esattamente tante ennuple quante  $R$

# Selezione e proiezione



- Combinando selezione e proiezione, possiamo estrarre interessanti informazioni da una relazione

- matricola e cognome degli impiegati che guadagnano più di 50

Matricola	Cognome
7309	Rossi
5998	Neri
5698	Neri

**PROJ<sub>Matricola,Cognome</sub> ( SEL<sub>Stipendio > 50</sub> (Impiegati))**



- combinando selezione e proiezione, possiamo estrarre informazioni da **una** relazione
- non possiamo però correlare informazioni presenti in relazioni diverse, né informazioni in ennuple diverse di una stessa relazione

# Join



- il join è l'operatore più interessante dell'algebra relazionale
- permette di correlare dati in relazioni diverse

# Prove scritte in un concorso pubblico



- I compiti sono anonimi e ad ognuno è associata una busta chiusa con il nome del candidato
- Ciascun compito e la relativa busta vengono contrassegnati con uno stesso numero

1	25
2	13
3	27
4	28

1	Mario Rossi
2	Nicola Russo
3	Mario Bianchi
4	Remo Neri

Mario Rossi	25
Nicola Russo	13
Mario Bianchi	27
Remo Neri	28

Numero	Voto
1	25
2	13
3	27
4	28

Numero	Candidato
1	Mario Rossi
2	Nicola Russo
3	Mario Bianchi
4	Remo Neri

Numero	Candidato	Voto
1	Mario Rossi	25
2	Nicola Russo	13
3	Mario Bianchi	27
4	Remo Neri	28

# Join naturale

- operatore binario (generalizzabile)
- produce un risultato
  - sull'unione degli attributi degli operandi
  - con ennuple costruite ciascuna a partire da una ennupla di ognuno degli operandi

# Join, sintassi e semantica

- $R_1(X_1), R_2(X_2)$
- $R_1 \text{ JOIN } R_2$  è una relazione su  $X_1X_2$

$$\{ t \text{ su } X_1X_2 \mid \text{esistono } t_1 \in R_1 \text{ e } t_2 \in R_2 \\ \text{con } t[X_1] = t_1 \text{ e } t[X_2] = t_2 \}$$

Impiegato	Reparto	Reparto	Capo
Rossi	A	A	Mori
Neri	B	B	Bruni
Bianchi	B		

Impiegato	Reparto	Capo
Rossi	A	Mori
Neri	B	Bruni
Bianchi	B	Bruni

- ogni ennupla contribuisce al risultato:
  - join **completo**



# Un join non completo

Impiegato	Reparto
Rossi	A
Neri	B
Bianchi	B

Reparto	Capo
B	Mori
C	Bruni

Impiegato	Reparto	Capo
Neri	B	Mori
Bianchi	B	Mori

# Un join vuoto

Impiegato	Reparto
Rossi	A
Neri	B
Bianchi	B

Reparto	Capo
D	Mori
C	Bruni

Impiegato	Reparto	Capo
-----------	---------	------

# Un join completo, con $n \times m$ ennuple

Impiegato	Reparto
Rossi	B
Neri	B

Reparto	Capo
B	Mori
B	Bruni

Impiegato	Reparto	Capo
Rossi	B	Mori
Rossi	B	Bruni
Neri	B	Mori
Neri	B	Bruni

# Cardinalità del join

- Il join di  $R_1$  e  $R_2$  contiene un numero di ennuple ...
  - compreso fra zero e il prodotto di  $|R_1|$  e  $|R_2|$
- se il join coinvolge una chiave di  $R_2$ , allora il numero di ennuple è ...
  - compreso fra zero e  $|R_1|$
- se il join coinvolge una chiave di  $R_2$  e un vincolo di integrità referenziale, allora il numero di ennuple è
  - pari a  $|R_1|$

# Cardinalità del join, 2

- $R_1(A,B)$  ,  $R_2(B,C)$

- in generale

$$0 \leq |R_1 \text{ JOIN } R_2| \leq |R_1| \times |R_2|$$

- se  $B$  è chiave in  $R_2$

$$0 \leq |R_1 \text{ JOIN } R_2| \leq |R_1|$$

- se  $B$  è chiave in  $R_2$  ed esiste vincolo di integrità referenziale fra  $B$  (in  $R_1$ ) e  $R_2$ :

$$|R_1 \text{ JOIN } R_2| = |R_1|$$

# Join, una difficoltà

Impiegato	Reparto	Reparto	Capo
Rossi	A	B	Mori
Neri	B	C	Bruni
Bianchi	B		

Impiegato	Reparto	Capo
Neri	B	Mori
Bianchi	B	Mori

- alcune ennuple non contribuiscono al risultato: vengono "tagliate fuori"

# Join esterno

- Il join **esterno** estende, con valori nulli, le ennuple che verrebbero tagliate fuori da un join (**interno**)
- esiste in tre versioni:
  - sinistro, destro, completo

# Join esterno

- **sinistro**: mantiene tutte le ennuple del primo operando, estendendole con valori nulli, se necessario
- **destro**: ... del secondo operando ...
- **completo**: ... di entrambi gli operandi ...



## Impiegati

Impiegato	Reparto
Rossi	A
Neri	B
Bianchi	B

## Reparti

Reparto	Capo
B	Mori
C	Bruni

## Impiegati JOIN<sub>LEFT</sub> Reparti

Impiegato	Reparto	Capo
Neri	B	Mori
Bianchi	B	Mori
Rossi	A	NULL

## Impiegati

Impiegato	Reparto
Rossi	A
Neri	B
Bianchi	B

## Reparti

Reparto	Capo
B	Mori
C	Bruni

## Impiegati JOIN<sub>RIGHT</sub> Reparti

Impiegato	Reparto	Capo
Neri	B	Mori
Bianchi	B	Mori
NULL	C	Bruni

## Impiegati

Impiegato	Reparto
Rossi	A
Neri	B
Bianchi	B

## Reparti

Reparto	Capo
B	Mori
C	Bruni

## Impiegati JOIN<sub>FULL</sub> Reparti

Impiegato	Reparto	Capo
Neri	B	Mori
Bianchi	B	Mori
Rossi	A	NULL
NULL	C	Bruni

# Join e proiezioni



Impiegato	Reparto	Reparto	Capo
Rossi	A	B	Mori
Neri	B	C	Bruni
Bianchi	B		

Impiegato	Reparto	Capo
Neri	B	Mori
Bianchi	B	Mori

Impiegato	Reparto	Reparto	Capo
Neri	B	B	Mori
Bianchi	B		

# Join e proiezioni

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

$$\text{PROJ}_{X_1}(R_1 \text{ JOIN } R_2) \subseteq R_1$$

# Proiezioni e join



Impiegato	Reparto	Capo
Neri	B	Mori
Bianchi	B	Bruni
Verdi	A	Bini

Impiegato	Reparto
Neri	B
Bianchi	B
Verdi	A

Reparto	Capo
B	Mori
B	Bruni
A	Bini

Impiegato	Reparto	Capo
Neri	B	Mori
Neri	B	Bruni
Bianchi	B	Mori
Bianchi	B	Bruni
Verdi	A	Bini

# Join e proiezioni

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

$$\text{PROJ}_{X_1}(R_1 \text{ JOIN } R_2) \subseteq R_1$$

- $R(X), X = X_1 \cup X_2$

$$(\text{PROJ}_{X_1}(R)) \text{ JOIN } (\text{PROJ}_{X_2}(R)) \supseteq R$$

# Prodotto cartesiano



- un join naturale su relazioni senza attributi in comune
- contiene sempre un numero di ennuple pari al prodotto delle cardinalità degli operandi (le ennuple sono tutte combinabili)



## Impiegati

Impiegato	Reparto
Rossi	A
Neri	B
Bianchi	B

## Reparti

Codice	Capo
A	Mori
B	Bruni



## Impiegati JOIN Reparti

Impiegato	Reparto	Codice	Capo
Rossi	A	A	Mori
Rossi	A	B	Bruni
Neri	B	A	Mori
Neri	B	B	Bruni
Bianchi	B	A	Mori
Bianchi	B	B	Bruni

- Il prodotto cartesiano, in pratica, ha senso (quasi) solo se seguito da selezione:

$$SEL_{Condizione} (R_1 \text{ JOIN } R_2)$$

- L'operazione viene chiamata **theta-join** e indicata con

$$R_1 \text{ JOIN}_{Condizione} R_2$$

# Perché "theta-join"?

- La condizione **C** è spesso una congiunzione (**AND**) di atomi di confronto  $A_1 \vartheta A_2$  dove  $\vartheta$  è uno degli operatori di confronto ( $=, >, <, \dots$ )

# Equi-join

- Se l'operatore di confronto nel theta-join è sempre l'uguaglianza (=) allora si parla di **equi-join**

**Nota: ci interessa davvero l'equi-join, non il theta-join più generale**

## Impiegati

Impiegato	Reparto
Rossi	A
Neri	B
Bianchi	B

## Reparti

Codice	Capo
A	Mori
B	Bruni

## Impiegati JOIN<sub>Reparto=Codice</sub> Reparti

Impiegato	Reparto	Codice	Capo
Rossi	A	A	Mori
Neri	B	B	Bruni
Bianchi	B	B	Bruni

## Impiegati

Impiegato	Reparto
Rossi	A
Neri	B
Bianchi	B

## Reparti

Reparto	Capo
A	Mori
B	Bruni

## Impiegati JOIN Reparti

# Join naturale ed equi-join

Impiegati

Impiegato Reparto

Reparti

Reparto Capo

Impiegati JOIN Reparti

$$\text{PROJ}_{\text{Impiegato, Reparto, Capo}} \left( \text{SEL}_{\text{Reparto=Codice}} \left( \text{Impiegati JOIN}_{\text{Codice} \leftarrow \text{Reparto}} \text{Reparti} \right) \right)$$

# Esempi

## Impiegati

Matricola	Nome	Età	Stipendio
7309	Rossi	34	45
5998	Bianchi	37	38
9553	Neri	42	35
5698	Bruni	43	42
4076	Mori	45	50
8123	Lupi	46	60

## Supervisione

Impiegato	Capo
7309	5698
5998	5698
9553	4076
5698	4076
4076	8123



- Trovare matricola, nome, età e stipendio degli impiegati che guadagnano più di 40

$SEL_{\text{Stipendio} > 40}(\text{Impiegati})$

- Trovare matricola, nome ed età degli impiegati che guadagnano più di 40

**PROJ<sub>Matricola, Nome, Età</sub> (SEL<sub>Stipendio>40</sub>(Impiegati))**

- Trovare i capi degli impiegati che guadagnano più di 40

$\text{PROJ}_{\text{Capo}} (\text{Supervisione}$   
 $\text{JOIN}_{\text{Impiegato=Matricola}} (\text{SEL}_{\text{Stipendio}>40} (\text{Impiegati})))$

- Trovare nome e stipendio dei capi degli impiegati che guadagnano più di 40

```

      PROJNome,Stipendio (
Impiegati JOINMatricola=Capo
      PROJCapo(Supervisione
JOINImpiegato=Matricola (SELStipendio>40(Impiegati))))
    
```

- Trovare gli impiegati che guadagnano più del proprio capo, mostrando matricola, nome e stipendio dell'impiegato e del capo

```

PROJMatr,Nome,Stip,MatrC,NomeC,StipC
  (SELStipendio>StipC
    RENMatrC,NomeC,StipC,EtàC ← Matr,Nome,Stip,Età (Impiegati)
      JOINMatrC=Capo
        (Supervisione JOINImpiegato=Matricola Impiegati)))
  
```

- Trovare le matricole dei capi i cui impiegati guadagnano **tutti** più di 40

$$\text{PROJ}_{\text{Capo}}(\text{Supervisione}) -$$

$$\text{PROJ}_{\text{Capo}}(\text{Supervisione}$$

$$\text{JOIN}_{\text{Impiegato}=\text{Matricola}}$$

$$(\text{SEL}_{\text{Stipendio} \leq 40}(\text{Impiegati})))$$

# Equivalenza di espressioni

- Due espressioni sono **equivalenti** se producono lo stesso risultato qualunque sia l'istanza attuale della base di dati
- L'equivalenza è importante in pratica perché i DBMS cercano di eseguire espressioni equivalenti a quelle date, ma meno "costose"

# Un'equivalenza importante

- Push selections (se  $A$  è attributo di  $R_2$ )

$$SEL_{A=10}(R_1 \text{ JOIN } R_2) = R_1 \text{ JOIN } SEL_{A=10}(R_2)$$

- Riduce in modo significativo la dimensione del risultato intermedio (e quindi il costo dell'operazione)



# Nota



- In questo corso, ci preoccupiamo poco dell'efficienza:
  - l'obiettivo è di scrivere interrogazioni corrette e leggibili
- Motivazione:
  - I DBMS si preoccupano di scegliere le strategie realizzative efficienti

# Selezione con valori nulli

## Impiegati

Matricola	Cognome	Filiale	Età
7309	Rossi	Roma	32
5998	Neri	Milano	45
9553	Bruni	Milano	NULL

**SEL**  $Età > 40$  (Impiegati)

- la condizione atomica è vera solo per valori non nulli

# Un risultato non desiderabile

$$SEL_{Età>30} (Persone) \cup SEL_{Età\leq 30} (Persone) \neq Persone$$

- Perché? Perché le selezioni vengono valutate separatamente!
- Ma anche

$$SEL_{Età>30 \vee Età\leq 30} (Persone) \neq Persone$$

- Perché? Perché anche le condizioni atomiche vengono valutate separatamente!

# Selezione con valori nulli: soluzione

$SEL_{Età > 40}$  (Impiegati)

- la condizione atomica è vera solo per valori non nulli
- per riferirsi ai valori nulli esistono forme apposite di condizioni:

IS NULL

IS NOT NULL

- si potrebbe usare (ma non serve) una "logica a tre valori" (vero, falso, sconosciuto)

- Quindi:

$$\begin{aligned}
 & \text{SEL}_{\text{Età} > 30}(\text{Persone}) \cup \text{SEL}_{\text{Età} \leq 30}(\text{Persone}) \cup \\
 & \quad \text{SEL}_{\text{Età IS NULL}}(\text{Persone}) \\
 & \quad = \\
 & \text{SEL}_{\text{Età} > 30 \vee \text{Età} \leq 30 \vee \text{Età IS NULL}}(\text{Persone}) \\
 & \quad = \\
 & \text{Persone}
 \end{aligned}$$

## Impiegati

Matricola	Cognome	Filiale	Età
5998	Neri	Milano	45
9553	Bruni	Milano	NULL

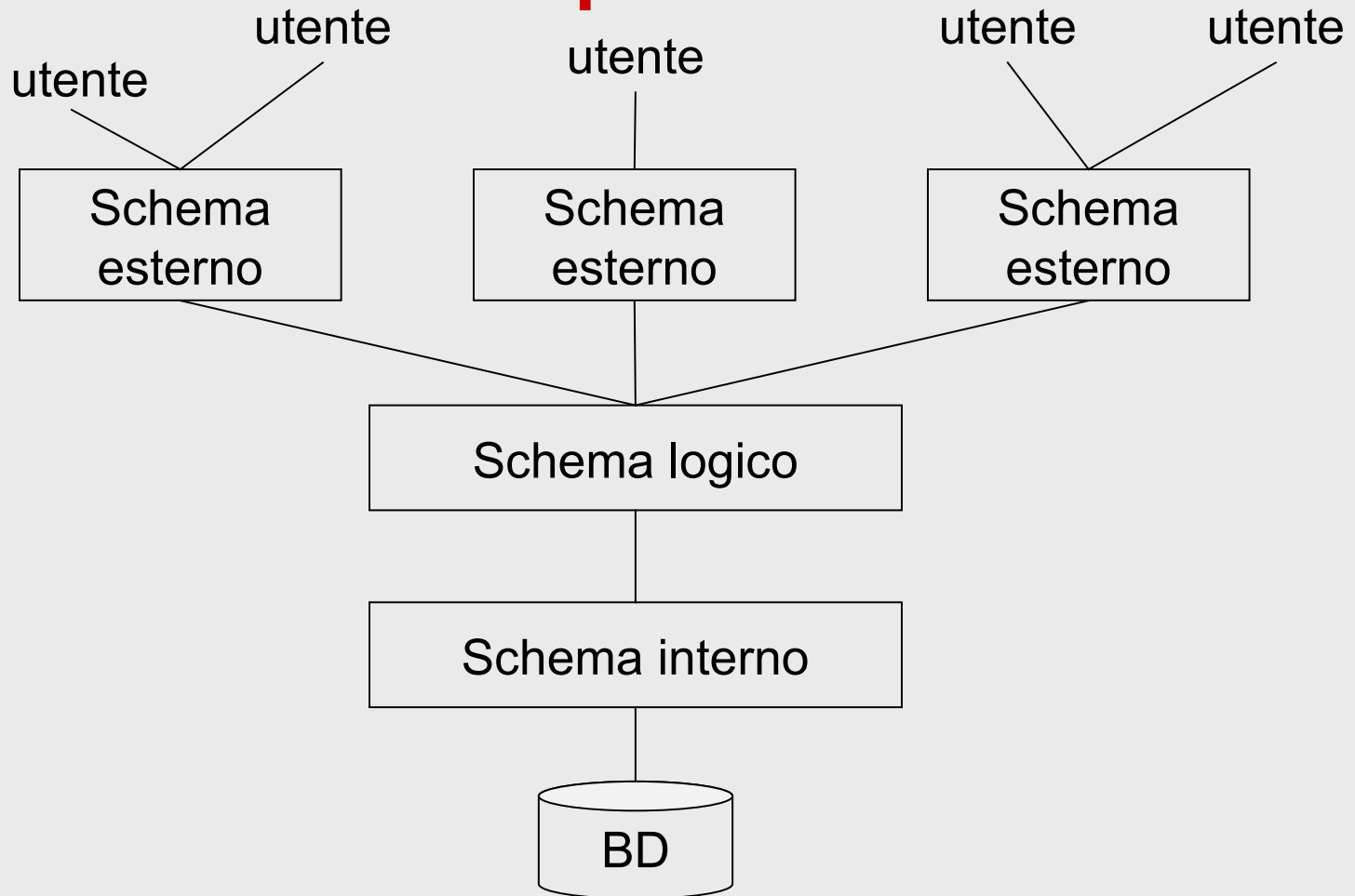
**SEL** (Età > 40) OR (Età IS NULL) (Impiegati)

# Viste (relazioni derivate)

- Rappresentazioni diverse per gli stessi dati (**schema esterno**)
- **Relazioni derivate**:
  - relazioni il cui contenuto è funzione del contenuto di altre relazioni (definito per mezzo di interrogazioni)
- **Relazioni di base**: contenuto autonomo
- Le relazioni derivate possono essere definite su altre derivate, ma ...



# Architettura standard (ANSI/SPARC) a tre livelli per DBMS





# Viste, esempio

Afferenza

Impiegato	Reparto
Rossi	A
Neri	B
Bianchi	B

Direzione

Reparto	Capo
A	Mori
B	Bruni

- una vista:

Supervisione =

$\text{PROJ}_{\text{Impiegato, Capo}} (\text{Afferenza JOIN Direzione})$

# Viste virtuali e materializzate

- Due tipi di relazioni derivate:
  - viste materializzate
  - relazioni virtuali (o viste)

# Viste materializzate

- relazioni **derivate memorizzate** nella base di dati
  - vantaggi:
    - immediatamente disponibili per le interrogazioni
  - svantaggi:
    - ridondanti
    - appesantiscono gli aggiornamenti
    - sono raramente supportate dai DBMS



# Viste virtuali

- **relazioni virtuali (o viste):**
  - sono supportate dai DBMS (tutti)
  - una interrogazione su una vista viene eseguita "ricalcolando" la vista (o quasi)

# Interrogazioni sulle viste

- Sono eseguite sostituendo alla vista la sua definizione:

$SEL_{\text{Capo}='Leoni'}$  (Supervisione)

viene eseguita come

$SEL_{\text{Capo}='Leoni'}$   
 $PROJ_{\text{Impiegato, Capo}}$  (Afferenza JOIN Direzione))

# Viste, motivazioni



- Schema esterno: ogni utente vede solo
  - ciò che gli interessa e nel modo in cui gli interessa, senza essere distratto dal resto
  - ciò che e' autorizzato a vedere (autorizzazioni)
- **Strumento di programmazione:**
  - si può semplificare la scrittura di interrogazioni: espressioni complesse e sottoespressioni ripetute
- Utilizzo di programmi esistenti su schemi ristrutturati

Invece:

- L'utilizzo di viste non influisce sull'efficienza delle interrogazioni



# Viste come strumento di programmazione

- Trovare gli impiegati che hanno lo stesso capo di Rossi
- Senza vista:

```

PROJImpiegato ((Afferenza JOIN Direzione) JOIN
                RENImpR,RepR ← Imp,Reparto (
                SELImpiegato='Rossi' (Afferenza JOIN Direzione)))
    
```

- Con la vista:

```

PROJImpiegato (Supervisione JOIN
                RENImpR← Imp (
                SELImpiegato='Rossi' (Supervisione)))
    
```



# Viste e aggiornamenti, attenzione



## Afferenza

Impiegato	Reparto
Rossi	A
Neri	B
Verdi	A

## Direzione

Reparto	Capo
A	Mori
B	Bruni
C	Bruni

## Supervisione

Impiegato	Capo
Rossi	Mori
Neri	Bruni
Verdi	Mori

- Vogliamo inserire, nella vista, il fatto che Lupi ha come capo Bruni; oppure che Belli ha come capo Falchi; come facciamo?



# Viste e aggiornamenti



- "Aggiornare una vista":
  - modificare le relazioni di base in modo che la vista, "ricalcolata" rispecchi l'aggiornamento
- L'aggiornamento sulle relazioni di base corrispondente a quello specificato sulla vista deve essere univoco
- In generale però non è univoco!
- Ben pochi aggiornamenti sono ammissibili sulle viste

# Una convenzione e notazione alternativa per i join



- Nota: è sostanzialmente l'approccio usato in SQL
- Ignoriamo il join naturale (cioè non consideriamo implicitamente condizioni su attributi con nomi uguali)
- Per "riconoscere" attributi con lo stesso nome gli premettiamo il nome della relazione
- Usiamo "assegnazioni" (viste) per ridenominare le relazioni (e gli attributi solo quando serve per l'unione)

- Trovare gli impiegati che guadagnano più del proprio capo, mostrando matricola, nome e stipendio dell'impiegato e del capo

```

PROJMatr,Nome,Stip,MatrC,NomeC,StipC
  (SELStipendio>StipC
    RENMatrC,NomeC,StipC,EtàC ← Matr,Nome,Stip,Età (Impiegati)
      JOINMatrC=Capo
        (Supervisione JOINImpiegato=Matricola Impiegati)))
  
```

PROJ<sub>Matr,Nome,Stip,MatrC,NomeC,StipC</sub>  
(SEL<sub>Stip>StipC</sub>(

REN<sub>MatrC,NomeC,StipC,EtàC</sub> ← Matr,Nome,Stip,Età (Imp)

JOIN<sub>MatrC=Capo</sub>  
(Sup JOIN<sub>Imp=Matr</sub> Imp)))

Capi := Imp

PROJ<sub>Imp.Matr, Imp.Nome, Imp.Stip,Capi.Matr,Capi.Nome, Capi.Stip</sub>  
(SEL<sub>Imp.Stip>Capi.Stip</sub>(

Capi JOIN<sub>Capi.Matr=Capo</sub> (Sup JOIN<sub>Imp=Imp.Matr</sub> Imp)))

# Calcolo relazionale

- Una famiglia di linguaggi **dichiarativi**, basati sul calcolo dei predicati del primo ordine
- Diverse versioni:
  - calcolo relazionale su domini
  - calcolo su ennuple con dichiarazioni di range

# Calcolo su domini, sintassi e semantica



- Le espressioni hanno la forma:
 
$$\{ A_1: x_1, \dots, A_k: x_k \mid f \}$$
  - $f$  e' una formula (con connettivi booleani e quantificatori)
  - $A_1: x_1, \dots, A_k: x_k$  "target list":
    - $A_1, \dots, A_k$  attributi distinti (anche non nella base di dati)
    - $x_1, \dots, x_k$  variabili distinte
- Semantica: il risultato e' una relazione su  $A_1, \dots, A_k$  che contiene ennuple di valori per  $x_1, \dots, x_k$  che rendono vera la formula  $f$

# Commenti

- Differenze rispetto al calcolo dei predicati (per chi lo conosce):
  - simboli di predicato
    - relazioni nella base di dati
    - predicati "standard" predefiniti ( $=$ ,  $>$ , ...)
  - non ci sono "simboli di funzione"
  - interessano (quasi) solo "formule aperte"
  - utilizziamo notazione non posizionale

# Base di dati per gli esempi

Impiegati(Matricola, Nome, Età, Stipendio)  
Supervisione(Capo, Impiegato)



# Esempio 0a

- Trovare matricola, nome, età e stipendio degli impiegati che guadagnano più di 40

$SEL_{\text{Stipendio} > 40}(\text{Impiegati})$

{ Matricola: m, Nome: n, Età: e, Stipendio: s |  
Impiegati(Matricola: m, Nome: n, Età: e,  
Stipendio: s)  $\wedge$  s > 40 }

# Esempio 0b

- Trovare matricola, nome ed età di tutti gli impiegati

$\text{PROJ}_{\text{Matricola, Nome, Et\`a}}(\text{Impiegati})$

$\{ \text{Matricola: } m, \text{ Nome: } n, \text{ Et\`a: } e \mid$   
 $\exists s (\text{Impiegati}(\text{Matricola: } m, \text{ Nome: } n, \text{ Et\`a: } e, \text{ Stipendio: } s)) \}$

$\{ \text{Matricola: } m, \text{ Nome: } n, \text{ Et\`a: } e \mid$   
 $\text{Impiegati}(\text{Matricola: } m, \text{ Nome: } n, \text{ Et\`a: } e, \text{ Stipendio: } s) \}$

# Esempio 1

- Trovare matricola, nome ed età degli impiegati che guadagnano più di 40

$\text{PROJ}_{\text{Matricola, Nome, Et\`a}}(\text{SEL}_{\text{Stipendio} > 40}(\text{Impiegati}))$

$\{ \text{Matricola: m, Nome: n, Et\`a: e} \mid$   
 $\text{Impiegati}(\text{Matricola: m, Nome: n, Et\`a: e,}$   
 $\text{Stipendio: s}) \wedge s > 40 \}$

## Esempio 2

- Trovare le matricole dei capi degli impiegati che guadagnano più di 40

$\text{PROJ}_{\text{Capo}} (\text{Supervisione JOIN}_{\text{Impiegato=Matricola}} (\text{SEL}_{\text{Stipendio}>40} (\text{Impiegati})))$

$\{ \text{Capo: } c \mid \text{Supervisione}(\text{Capo:}c, \text{Impiegato:}m) \wedge \text{Impiegati}(\text{Matricola: } m, \text{Nome: } n, \text{Età: } e, \text{Stipendio: } s) \wedge s > 40 \}$

# Esempio 3

- Trovare nome e stipendio dei capi degli impiegati che guadagnano più di 40

PROJ<sub>NomeC,StipC</sub> (REN<sub>MatrC,NomeC,StipC,EtàC</sub> ← Matr, Nome, Stip, Età (Impiegati)  
 JOIN<sub>MatrC=Capo</sub>  
 (Supervisione JOIN<sub>Impiegato=Matricola</sub> (SEL<sub>Stipendio>40</sub> (Impiegati))))

{ NomeC: nc, StipC: sc |  
 Impiegati(Matricola: m, Nome: n, Età: e, Stipendio: s) ∧ s >  
 40 ∧ Supervisione(Capo:c, Impiegato:m) ∧  
 Impiegati(Matricola:c, Nome:nc, Età:ec, Stipendio: sc) }

# Esempio 4

- Trovare gli impiegati che guadagnano più del rispettivo capo, mostrando matricola, nome e stipendio di ciascuno di essi e del capo

$$\begin{aligned}
 & \text{PROJ}_{\text{Matr, Nome, Stip, MatrC, NomeC, StipC}} \\
 & (\text{SEL}_{\text{Stipendio} > \text{StipC}} (\text{REN}_{\text{MatrC, NomeC, StipC, Et\`aC} \leftarrow \\
 & \quad \text{Matr, Nome, Stip, Et\`a} (\text{Impiegati}) \\
 & \quad \text{JOIN}_{\text{MatrC} = \text{Capo}} \\
 & (\text{Supervisione JOIN}_{\text{Impiegato} = \text{Matricola}} ( (\text{Impiegati}))))
 \end{aligned}$$

$$\begin{aligned}
 & \{ \text{Matr: m, Nome: n, Stip: s, MatrC: c, NomeC: nc, StipC: sc} \mid \\
 & \quad \text{Impiegati}(\text{Matricola: m, Nome: n, Et\`a: e, Stipendio: s}) \wedge \\
 & \quad \text{Supervisione}(\text{Capo: c, Impiegato: m}) \wedge \\
 & \quad \text{Impiegati}(\text{Matricola: c, Nome: nc, Et\`a: ec, Stipendio: sc}) \wedge s > sc \}
 \end{aligned}$$

# Esempio 5

- Trovare matricola e nome dei capi i cui impiegati guadagnano tutti più di 40

$\text{PROJ}_{\text{Matricola, Nome}} (\text{Impiegati JOIN}_{\text{Matricola=Capo}} (\text{PROJ}_{\text{Capo}} (\text{Supervisione}) - \text{PROJ}_{\text{Capo}} (\text{Supervisione JOIN}_{\text{Impiegato=Matricola}} (\text{SEL}_{\text{Stipendio} \leq 40} (\text{Impiegati}))))))$

$\{ \text{Matricola: } c, \text{ Nome: } n \mid$   
 $\text{Impiegati}(\text{Matricola: } c, \text{ Nome: } n, \text{ Et\`a: } e, \text{ Stipendio: } s) \wedge$   
 $\text{Supervisione}(\text{Capo: } c, \text{ Impiegato: } m) \wedge$   
 $\neg \exists m' (\exists n' (\exists e' (\exists s' (\text{Impiegati}(\text{Matr: } m', \text{ Nome: } n', \text{ Et\`a: } e', \text{ Stip: } s') \wedge$   
 $\text{Supervisione}(\text{Capo: } c, \text{ Impiegato: } m') \wedge s' \leq 40)))))) \}$

# Quantificatori esistenziali o universali?

- Sono intercambiabili, per le leggi di De Morgan:

$$\{ \text{Matricola: } c, \text{ Nome: } n \mid \\ \text{Impiegati}(\text{Matricola: } c, \text{ Nome: } n, \text{ Et\`a: } e, \text{ Stipendio: } s) \wedge \\ \text{Supervisione}(\text{Capo: } c, \text{ Impiegato: } m) \wedge \\ \neg \exists m'(\exists n'(\exists e'(\exists s'(\text{Impiegati}(\text{Matr: } m', \text{ Nome: } n', \text{ Et\`a: } e', \text{ Stip: } s') \wedge \\ \text{Supervisione}(\text{Capo: } c, \text{ Impiegato: } m') \wedge s' \leq 40)))))) \}$$

$$\{ \text{Matricola: } c, \text{ Nome: } n \mid \\ \text{Impiegati}(\text{Matricola: } c, \text{ Nome: } n, \text{ Et\`a: } e, \text{ Stipendio: } s) \wedge \\ \text{Supervisione}(\text{Capo: } c, \text{ Impiegato: } m) \wedge \\ \forall m'(\forall n'(\forall e'(\forall s'(\neg(\text{Impiegati}(\text{Matr: } m', \text{ Nome: } n', \text{ Et\`a: } e', \text{ Stip: } s') \wedge \\ \text{Supervisione}(\text{Capo: } c, \text{ Impiegato: } m')) \vee s' > 40)))))) \}$$



# Calcolo su domini, discussione

- Pregi:
  - dichiaratività
- Difetti:
  - "verbosità": tante variabili!
  - espressioni senza senso:

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

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

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

queste espressioni sono "dipendenti dal dominio" e vorremmo evitarle;

nell'algebra espressioni come queste non sono formulabili: l'algebra è indipendente dal dominio

# Calcolo e algebra

- Calcolo e algebra sono "**equivalenti**"
  - per ogni espressione del calcolo relazionale che sia indipendente dal dominio esiste un'espressione dell'algebra relazionale equivalente a essa
  - per ogni espressione dell'algebra relazionale esiste un'espressione del calcolo relazionale equivalente a essa (e di conseguenza indipendente dal dominio)

# Calcolo su ennuple con dichiarazioni di range



- Per superare le limitazioni del calcolo su domini:
  - dobbiamo "ridurre" le variabili; un buon modo: una variabile per ciascuna ennupla
  - far sì che i valori provengano dalla base di dati
- Il **calcolo su ennuple con dichiarazioni di range** risponde ad entrambe le esigenze

# Calcolo su ennuple con dichiarazioni di range, sintassi

- Le espressioni hanno la forma:  

$$\{ \textit{TargetList} \mid \textit{RangeList} \mid \textit{Formula} \}$$
- RangeList* elenca le variabili libere della *Formula* ognuna con il relativo campo di variabilità (una relazione)
- TargetList* ha elementi del tipo  $Y: x.Z$  (oppure  $x.Z$  o anche  $x.*$ )
- Formula* ha:
  - atomi di confronto  $x.A \vartheta c$ ,  $x.A \vartheta y.B$
  - connettivi
  - quantificatori che associano un range alle variabili  

$$\exists x(R)(\dots) \quad \forall x(R)(\dots)$$

# Esempio 0a

- Trovare matricola, nome, età e stipendio degli impiegati che guadagnano più di 40

$SEL_{\text{Stipendio} > 40}(\text{Impiegati})$

$\{ \text{Matricola: } m, \text{ Nome: } n, \text{ Et\`a: } e, \text{ Stipendio: } s \mid$   
 $\text{Impiegati}(\text{Matricola: } m, \text{ Nome: } n, \text{ Et\`a: } e, \text{ Stipendio: } s) \wedge s$   
 $> 40 \}$

$\{ i.* \mid i(\text{Impiegati}) \mid i.\text{Stipendio} > 40 \}$

# Esempio 0b

- Trovare matricola, nome ed età di tutti gli impiegati

$\text{PROJ}_{\text{Matricola, Nome, Età}}(\text{Impiegati})$

$\{ \text{Matricola: m, Nome: n, Età: e} \mid$   
 $\text{Impiegati}(\text{Matricola: m, Nome: n, Età: e, Stipendio: s}) \}$

$\{ i.(\text{Matricola, Nome, Età}) \mid i(\text{Impiegati}) \mid \}$

# Esempio 1

- Trovare matricola, nome ed età degli impiegati che guadagnano più di 40

$\text{PROJ}_{\text{Matricola, Nome, Et\`a}}(\text{SEL}_{\text{Stipendio} > 40}(\text{Impiegati}))$

$\{ \text{Matricola: m, Nome: n, Et\`a: e} \mid$   
 $\text{Impiegati}(\text{Matricola: m, Nome: n, Et\`a: e, Stipendio: s}) \wedge s$   
 $> 40 \}$

$\{ i.(\text{Matricola, Nome, Et\`a}) \mid i(\text{Impiegati}) \mid i.\text{Stipendio} > 40 \}$

## Esempio 2

- Trovare le matricole dei capi degli impiegati che guadagnano più di 40

$$\{ \text{Capo: } c \mid \text{Supervisione}(\text{Capo:}c, \text{Impiegato:}m) \wedge \\ \text{Impiegati}(\text{Matricola: } m, \text{Nome: } n, \text{Età: } e, \\ \text{Stipendio: } s) \wedge s > 40 \}$$

$$\{ s.\text{Capo} \mid i(\text{Impiegati}), s(\text{Supervisione}) \mid \\ i.\text{Matricola} = s.\text{Impiegato} \wedge i.\text{Stipendio} > 40 \}$$



## Esempio 3

- Trovare nome e stipendio dei capi degli impiegati che guadagnano più di 40

$$\{ \text{NomeC: nc, StipC: sc} \mid \\ \text{Impiegati}(\text{Matricola: m, Nome: n, Et\`a: e, Stipendio: s}) \wedge s > 40 \wedge$$

$$\text{Supervisione}(\text{Capo: c, Impiegato: m}) \wedge \\ \text{Impiegati}(\text{Matricola: c, Nome: nc, Et\`a: ec, Stipendio: sc}) \}$$

$$\{ \text{NomeC, StipC: i'.(Nome, Stip)} \mid \\ i'(\text{Impiegati}), s(\text{Supervisione}), i(\text{Impiegati}) \mid \\ i'.\text{Matricola} = s.\text{Capo} \wedge i.\text{Matricola} = s.\text{Impiegato} \wedge i.\text{Stipendio} > 40 \}$$

# Esempio 4

- Trovare gli impiegati che guadagnano più del rispettivo capo, mostrando matricola, nome e stipendio di ciascuno di essi e del capo

$\{ \text{Matr: } m, \text{ Nome: } n, \text{ Stip: } s, \text{ NomeC: } nc, \text{ StipC: } sc \mid$   
 $\text{Impiegati}(\text{Matricola: } m, \text{ Nome: } n, \text{ Et\`a: } e, \text{ Stipendio: } s) \wedge$   
 $\text{Supervisione}(\text{Capo: } c, \text{ Impiegato: } m) \wedge$   
 $\text{Impiegati}(\text{Matricola: } c, \text{ Nome: } nc, \text{ Et\`a: } ec, \text{ Stipendio: } sc) \wedge$   
 $s > sc \}$

$\{ i.(\text{Nome, Matr, Stip}), \text{ NomeC, MatrC, StipC: } i'.(\text{Nome, Matr, Stip}) \mid$   
 $i'(\text{Impiegati}), s(\text{Supervisione}), i(\text{Impiegati}) \mid$   
 $i'.\text{Matricola} = s.\text{Capo} \wedge i.\text{Matricola} = s.\text{Impiegato} \wedge i.\text{Stipendio} >$   
 $i'.\text{Stipendio} \}$

# Esempio 5

- Trovare matricola e nome dei capi i cui impiegati guadagnano tutti più di 40

$$\begin{aligned} & \{ \text{Matricola: } c, \text{ Nome: } n \mid \\ & \text{Impiegati}(\text{Matricola: } c, \text{ Nome: } n, \text{ Et\`a: } e, \text{ Stipendio: } s) \wedge \\ & \text{Supervisione}(\text{Capo: } c, \text{ Impiegato: } m) \wedge \\ & \neg \exists m' (\exists n' (\exists e' (\exists s' (\text{Impiegati}(\text{Matr: } m', \text{ Nome: } n', \text{ Et\`a: } e', \text{ Stip: } s') \wedge \\ & \text{Supervisione}(\text{Capo: } c, \text{ Impiegato: } m') \wedge s' \leq 40) \end{aligned}$$

$$\begin{aligned} & \{ i.(\text{Matricola}, \text{Nome}) \mid s(\text{Supervisione}), i(\text{Impiegati}) \mid \\ & i.\text{Matricola} = s.\text{Capo} \wedge \neg (\exists i' (\text{Impiegati}) (\exists s' (\text{Supervisione}) \\ & (s.\text{Capo} = s'.\text{Capo} \wedge s'.\text{Impiegato} = i'.\text{Matricola} \wedge i'.\text{Stipendio} \leq 40))) \} \end{aligned}$$

# Attenzione!

- Il calcolo su ennuple con dichiarazioni di range non permette di esprimere alcune interrogazioni importanti, in particolare le unioni:  

$$R_1(AB) \cup R_2(AB)$$
- Quale potrebbe essere il range per una variabile? Oppure due variabili?
- Nota: intersezione e differenza sono esprimibili
- Per questa ragione SQL (che è basato su questo calcolo) prevede un operatore esplicito di unione, ma non tutte le versioni prevedono intersezione e differenza

# Calcolo e algebra relazionale: limiti

- Calcolo e algebra sono sostanzialmente equivalenti: l'insieme di interrogazioni con essi esprimibili è quindi significativo; il concetto è **robusto**
- Ci sono però interrogazioni interessanti non esprimibili:
  - calcolo di valori derivati: possiamo solo **estrarre** valori, non calcolarne di nuovi; calcoli di interesse:
    - a livello di ennupla o di singolo valore (conversioni somme, differenze, etc.)
    - su insiemi di ennuple (somme, medie, etc.)
 le estensioni sono ragionevoli, le vedremo in SQL
  - interrogazioni inerentemente **ricorsive**, come la **chiusura transitiva**

# Chiusura transitiva

## Supervisione(Impiegato, Capo)

- Per ogni impiegato, trovare tutti i superiori (cioè il capo, il capo del capo, e così via)

Impiegato	Capo
Rossi	Lupi
Neri	Bruni
Lupi	Falchi

Impiegato	Superiore
Rossi	Lupi
Neri	Bruni
Lupi	Falchi
Rossi	Falchi

# Chiusura transitiva, come si fa?

- Nell'esempio, basterebbe il join della relazione con se stessa, previa opportuna ridenominazione
- Ma:

Impiegato	Capo
Rossi	Lupi
Neri	Bruni
Lupi	Falchi
Falchi	Leoni

Impiegato	Superiore
Rossi	Lupi
Neri	Bruni
Lupi	Falchi
Falchi	Leoni
Rossi	Falchi
Lupi	Leoni
Rossi	Leoni

# Chiusura transitiva, impossibile!



- Non esiste in algebra e calcolo relazionale la possibilità di esprimere l'interrogazione che, per ogni relazione binaria, ne calcoli la chiusura transitiva
- Per ciascuna relazione, è possibile calcolare la chiusura transitiva, ma con un'espressione ogni volta diversa:
  - quanti join servono?
  - non c'è limite!



# Datalog



- Un linguaggio di programmazione logica per basi di dati derivato dal Prolog
- Utilizza predicati di due tipi:
  - **estensionali**: relazioni della base di dati
  - **intensionali**: corrispondono alle viste
- Il linguaggio è basato su **regole** utilizzate per "definire" i predicati estensionali

# Datalog, sintassi

- Regole:

*testa*  $\leftarrow$  *corpo*

- *testa* è un predicato atomico (intensionale)
- *corpo* è una lista (congiunzione) di predicati atomici
- Le interrogazioni sono specificate per mezzo di predicati atomici (convenzionalmente preceduti da "?")

# Esempio -1

- Trovare matricola, nome, età e stipendio degli impiegati che hanno 30 anni

{ Matricola: m, Nome: n, Età: e, Stipendio: s |  
Impiegati(Matricola: m, Nome: n, Età: e, Stipendio: s)  $\wedge$  s  
= 30 }

? Impiegati(Matricola: m, Nome: n, Età: 30, Stipendio: s)

# Esempio 0a

- Trovare matricola, nome, età e stipendio degli impiegati che guadagnano più di 40

$\{ \text{Matricola: } m, \text{ Nome: } n, \text{ Età: } e, \text{ Stipendio: } s \mid$   
 $\text{Impiegati}(\text{Matricola: } m, \text{ Nome: } n, \text{ Età: } e, \text{ Stipendio: } s) \wedge s > 40 \}$

- Serve un predicato intensionale

$\text{ImpRicchi}(\text{Matricola: } m, \text{ Nome: } n, \text{ Età: } e, \text{ Stipendio: } s) \leftarrow$   
 $\text{Impiegati}(\text{Matricola: } m, \text{ Nome: } n, \text{ Età: } e, \text{ Stipendio: } s), s > 40$

$? \text{ImpRicchi}(\text{Matricola: } m, \text{ Nome: } n, \text{ Età: } e, \text{ Stipendio: } s)$

## Esempio 0b

- Trovare matricola, nome ed età di tutti gli impiegati

$\text{PROJ}_{\text{Matricola, Nome, Età}}(\text{Impiegati})$

$\{ \text{Matricola: m, Nome: n, Età: e} \mid$   
 $\text{Impiegati}(\text{Matricola: m, Nome: n, Età: e, Stipendio: s}) \}$

$\text{InfoPubbliche}(\text{Matricola: m, Nome: n, Età: e})$   
 $\leftarrow \text{Impiegati}(\text{Matricola: m, Nome: n, Età: e, Stipendio: s})$

?  $\text{InfoPubbliche}(\text{Matricola: m, Nome: n, Età: e})$

## Esempio 2

- Trovare le matricole dei capi degli impiegati che guadagnano più di 40

$\{ \text{Capo: } c \mid \text{Supervisione}(\text{Capo: } c, \text{Impiegato: } m) \wedge$   
 $\text{Impiegati}(\text{Matricola: } m, \text{Nome: } n, \text{Età: } e, \text{Stipendio: } s) \wedge s > 40 \}$

$\text{CapiDeiRicchi}(\text{Capo: } c) \leftarrow$   
 $\text{ImpRicchi}(\text{Matricola: } m, \text{Nome: } n, \text{Età: } e, \text{Stipendio: } s),$   
 $\text{Supervisione}(\text{Capo: } c, \text{Impiegato: } m)$

?  $\text{CapiDeiRicchi}(\text{Capo: } c)$

## Esempio 5

- Trovare matricola e nome dei capi i cui impiegati guadagnano tutti più di 40

- serve la negazione

**CapiDiNonRicchi (Capo:c) ←**

Supervisione (Capo:c,Impiegato:m),  
Impiegati (Matricola: m, Nome: n, Età: e, Stipendio: s) ,  
 $s \leq 40$

**CapiSoloDiRicchi (Matricola: c, Nome: n) ←**

Impiegati (Matricola: c, Nome: n, Età: e, Stipendio: s) ,  
Supervisione (Capo:c,Impiegato:m),  
not CapiDiNonRicchi (Capo:c)

**? CapiSoloDiRicchi (Matricola: c, Nome: n)**

## Esempio 6

- Per ogni impiegato, trovare tutti i superiori.
- Serve la ricorsione

Superiore (Impiegato: i, SuperCapo: c) ←  
Supervisione (Impiegato: i, Capo: c)

Superiore (Impiegato: i, SuperCapo: c) ←  
Supervisione (Impiegato: i, Capo: c'),  
Superiore (Impiegato: c', SuperCapo: c)



# Datalog, semantica

- La definizione della semantica delle regole ricorsive è delicata (in particolare con la negazione)
- Potere espressivo:
  - Datalog non ricorsivo senza negazione è equivalente al calcolo senza negazione e senza quantificatore universale
  - Datalog non ricorsivo con negazione è equivalente al calcolo e all'algebra
  - Datalog ricorsivo senza negazione e calcolo sono incomparabili
  - Datalog ricorsivo con negazione è più espressivo di calcolo e algebra