Basi di dati

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 ∪ 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 ∩ 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 ∪ 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

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

Genitore	Figlio
Adamo	Abele
Adamo	Caino
Abramo	Isacco

Paternità

Padre	Figlio
Adamo	Abele
Adamo	Caino
Abramo	Isacco

Maternità

Madre	Figlio
Eva	Abele
Eva	Set
Sara	Isacco

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

Genitore	Figlio
Adamo	Abele
Adamo	Caino
Abramo	Isacco

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

Genitore	Figlio
Eva	Abele
Eva	Set
Sara	Isacco

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

Genitore	Figlio
Adamo	Abele
Adamo	Caino
Abramo	Isacco

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

Genitore	Figlio
Eva	Abele
Eva	Set
Sara	Isacco

$REN_{Genitore \leftarrow Padre}$ (Paternità) $\cup$ $REN_{Genitore \leftarrow Madre}$ (Maternità)

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

$REN_{Sede, Retribuzione \leftarrow Ufficio, Stipendio}$ (Impiegati) $\cup$ $REN_{Sede, Retribuzione \leftarrow Fabbrica, Salario}$ (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

Impiegati

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

SEL$_{Stipendio > 50}$ (Impiegati)

- impiegati che guadagnano più di 50 e lavorano a Milano

SEL$_{Stipendio > 50 AND Filiale = 'Milano'}$ (Impiegati)

Matricola	Cognome	Filiale	Stipendio
5998	Neri	Milano	64

SEL$_{Stipendio > 50 AND Filiale = 'Milano'}$ (Impiegati)

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

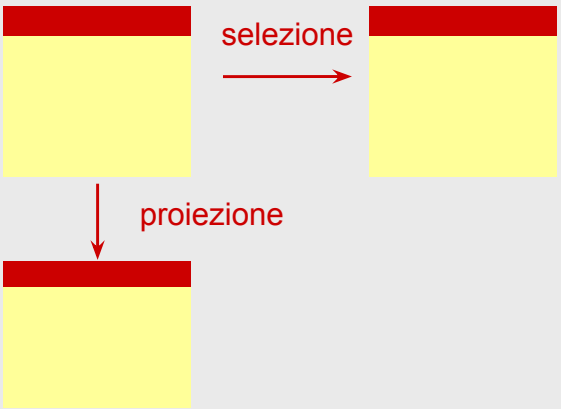
SEL $_{Cognome = Filiale}$(Impiegati)

Matricola	Cognome	Filiale	Stipendio
9553	Milano	Milano	44

SEL $_{Cognome = Filiale}$(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

  $\text{PROJ}_{\text{ListaAttributi}}(\text{Operando})$
- 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

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

- cognome e filiale di tutti gli impiegati

Cognome	Filiale
Neri	Napoli
Neri	Milano
Rossi	Roma

$\text{PROJ}_{\text{Cognome, Filiale}}(\text{Impiegati})$

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 $\text{PROJ}_X(R)$ 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

$$PROJ_{Matricola,Cognome}\ (SEL_{Stipendio > 50}\ (Impiegati))$$

- 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 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
Rossi	A
Neri	B
Bianchi	B

Reparto	Capo
A	Mori
B	Bruni

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 x 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
Rossi	A
Neri	B
Bianchi	B

Reparto	Capo
B	Mori
C	Bruni

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 $JOIN_{LEFT}$ 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 $JOIN_{RIGHT}$ 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 $JOIN_{FULL}$ Reparti

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

Join e proiezioni

Impiegato	Reparto
Rossi	A
Neri	B
Bianchi	B

Reparto	Capo
B	Mori
C	Bruni

Impiegato	Reparto	Capo
Neri	B	Mori
Bianchi	B	Mori

Impiegato	Reparto
Neri	B
Bianchi	B

Reparto	Capo
B	Mori

Join e proiezioni

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

$$PROJ_{X1}(R_1 \text{ JOIN } R_2) \subseteq R_1$$

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

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

$$PROJ_{X1}(R_1 \text{ JOIN } R_2) \subseteq R_1$$

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

$$(PROJ_{X1}(R)) \text{ JOIN } (PROJ_{X2}(R)) \supseteq R$$

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 ϑ è uno degli operatori di confronto (=, >, <, …)

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 $JOIN_{Reparto=Codice}$ 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

$PROJ_{Impiegato,Reparto,Capo}(SEL_{Reparto=Codice}$
(Impiegati JOIN $REN_{Codice \leftarrow Reparto}$ (Reparti)))

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

$SEL_{Stipendio>40}$(Impiegati)

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

$PROJ_{Matricola, Nome, Età}$ ($SEL_{Stipendio>40}$(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

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

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

$$\text{PROJ}_{\text{Matr,Nome,Stip,MatrC,NomeC,StipC}}\ (\text{SEL}_{\text{Stipendio>StipC}}(\ \text{REN}_{\text{MatrC,NomeC,StipC,EtàC}\leftarrow\text{Matr,Nome,Stip,Età}}\ (\text{Impiegati})\ \text{JOIN}_{\text{MatrC=Capo}}\ (\text{Supervisione JOIN}_{\text{Impiegato=Matricola}}\ \text{Impiegati})))$$

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

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

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)

$$\text{SEL}_{\text{A=10}}\,(R_1\ \text{JOIN}\ R_2) = R_1\ \text{JOIN}\ \text{SEL}_{\text{A=10}}\,(R_2)$$

- 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) ∪ $SEL_{Età≤30}$ (Persone) ≠ Persone

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

$SEL_{Età>30 \vee Età≤30}$ (Persone) ≠ 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:

$SEL_{Età>30}$ (Persone) ∪
$SEL_{Età≤30}$ (Persone) ∪
$SEL_{Età\ IS\ NULL}$ (Persone)
=
$SEL_{Età>30 \vee Età≤30 \vee Età\ IS\ NULL}$ (Persone)
=
Persone

Impiegati

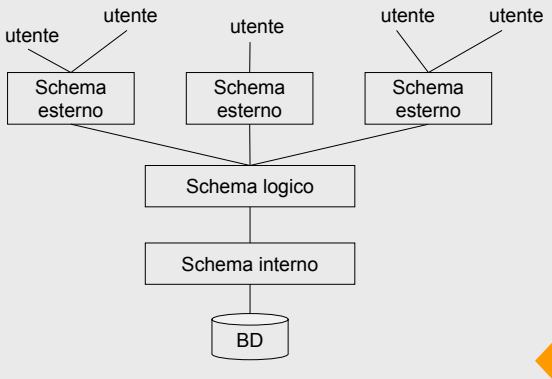
Matricola	Cognome	Filiale	Età
5998	Neri	Milano	45
9553	Bruni	Milano	NULL

$SEL_{(Età > 40) OR (Età\ IS\ NULL)}$ (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 =
PROJ $_{\text{Impiegato, Capo}}$ (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:

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

viene eseguita come

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

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:

$$PROJ_{Impiegato}((Afferenza\ JOIN\ Direzione)\ JOIN\ REN_{ImpR,RepR \leftarrow Imp,Reparto}(SEL_{Impiegato='Rossi'}(Afferenza\ JOIN\ Direzione)))$$

- Con la vista:

$$PROJ_{Impiegato}(Supervisione\ JOIN\ REN_{ImpR \leftarrow Imp}(SEL_{Impiegato='Rossi'}(Supervisione)))$$

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 aggionamenti 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

$$\text{PROJ}_{\text{Matr,Nome,Stip,MatrC,NomeC,StipC}}$$
$$(\text{SEL}_{\text{Stipendio>StipC}}($$
$$\text{REN}_{\text{MatrC,NomeC,StipC,EtàC} \leftarrow \text{Matr,Nome,Stip,Età}}(\text{Impiegati})$$
$$\text{JOIN}_{\text{MatrC=Capo}}$$
$$(\text{Supervisione JOIN}_{\text{Impiegato=Matricola}} \text{Impiegati})))$$

$$\text{PROJ}_{\text{Matr,Nome,Stip,MatrC,NomeC,StipC}}$$
$$(\text{SEL}_{\text{Stip>StipC}}($$
$$\text{REN}_{\text{MatrC,NomeC,StipC,EtàC} \leftarrow \text{Matr,Nome,Stip,Età}}(\text{Imp})$$
$$\text{JOIN}_{\text{MatrC=Capo}}$$
$$(\text{Sup JOIN}_{\text{Imp=Matr}} \text{Imp})))$$

Capi := Imp

$$\text{PROJ}_{\text{Imp.Matr, Imp.Nome, Imp.Stip,Capi.Matr,Capi.Nome, Capi.Stip}}$$
$$(\text{SEL}_{\text{Imp.Stip>Capi.Stip}}(\ \text{Capi}$$
$$\text{JOIN}_{\text{Capi.Matr=Capo}}$$
$$(\text{Sup JOIN}_{\text{Imp=Imp.Matr}} \text{Imp})))$$

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

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

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

Esempio 0b

- Trovare matricola, nome ed età di tutti gli impiegati

$$PROJ_{Matricola, Nome, Età}(Impiegati)$$

{ Matricola: m, Nome: n, Età: e | ∃ s (Impiegati(Matricola: m, Nome: n, Età: e, Stipendio: s)}

{ Matricola: m, Nome: n, Età: e | Impiegati(Matricola: m, Nome: n, Età: e, Stipendio: s)}

Esempio 1

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

$$PROJ_{Matricola, Nome, Età}(SEL_{Stipendio>40}(Impiegati))$$

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

Esempio 2

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

$$PROJ_{Capo}(\ Supervisione\ JOIN_{Impiegato=Matricola}(SEL_{Stipendio>40}(Impiegati)))$$

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

Esempio 3

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

$$PROJ_{NomeC,StipC}(\ REN_{MatrC,NomeC,StipC,EtàC \leftarrow Matr,Nome,Stip,Età}(Impiegati)\ JOIN_{MatrC=Capo}(Supervisione\ JOIN_{Impiegato=Matricola}(SEL_{Stipendio>40}(Impiegati))))$$

{ 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

$$PROJ_{Matr,Nome,Stip,MatrC,NomeC,StipC}(SEL_{Stipendio>StipC}(REN_{MatrC,NomeC,StipC,EtàC \leftarrow Matr,Nome,Stip,Età}(Impiegati)\ JOIN_{MatrC=Capo}(Supervisione\ JOIN_{Impiegato=Matricola}(Impiegati))))$$

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

Esempio 5

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

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

{Matricola: c, Nome: n |
Impiegati(Matricola: c, Nome: n, Età: e, Stipendio: s) ∧ Supervisione (Capo:c, Impiegato:m) ∧
¬ ∃m'(∃n'(∃e'(∃s'(Impiegati(Matr: m', Nome: n', Età: e', Stip: s') ∧ Supervisione(Capo:c, Impiegato:m') ∧ s' ≤ 40))))}

Quantificatori esistenziali o universali?

- Sono intercambiabili, per le leggi di De Morgan:

{Matricola: c, Nome: n |
Impiegati(Matricola: c, Nome: n, Età: e, Stipendio: s) ∧
Supervisione(Capo:c, Impiegato:m) ∧
¬ ∃m'(∃n'(∃e'(∃s'(Impiegati(Matr: m', Nome: n', Età: e', Stip: s') ∧ Supervisione(Capo:c, Impiegato:m') ∧ s' ≤ 40))))}

{Matricola: c, Nome: n |
Impiegati(Matricola: c, Nome: n, Età: e, Stipendio: s) ∧
Supervisione(Capo:c, Impiegato:m) ∧
∀m'(∀n'(∀e'(∀s'(¬(Impiegati(Matr:m', Nome:n', Età:e', Stip:s') ∧ Supervisione(Capo:c, Impiegato:m')) ∨ s' > 40))))}

Calcolo su domini, discussione

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

{ A: x | ¬ R(A: x) }
{ A: x, B: y | R(A: x) }
{ A: x, B: y | R(A: x) ∧ 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:

{ *TargetList* | *RangeList* | *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 ϑ c, x.A ϑ y.B
    - connettivi
    - quantificatori che associano un range alle variabili

∃x(R)(...) ∀x(R)(...)

Esempio 0a

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

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

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

{ i.* | i(Impiegati) | i.Stipendio > 40 }

Esempio 0b

- Trovare matricola, nome ed età di tutti gli impiegati

$$PROJ_{Matricola, Nome, Età}(Impiegati)$$

{ Matricola: m, Nome: n, Età: e |
Impiegati(Matricola: m, Nome: n, Età: e, Stipendio: s)}

{ i.(Matricola,Nome,Età) | i(Impiegati) | }

Esempio 1

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

$$PROJ_{Matricola, Nome, Età}(SEL_{Stipendio>40}(Impiegati))$$

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

{ i.(Matricola,Nome,Età) | i(Impiegati) | i.Stipendio > 40 }

Esempio 2

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

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

{ s.Capo | i(Impiegati) , s(Supervisione) |
i.Matricola=s.Impiegato ∧ i.Stipendio > 40 }

Esempio 3

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

{ 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) }

{ NomeC,StipC: i'.(Nome,Stip) |
i'(Impiegati), s(Supervisione), i(Impiegati) |
i'.Matricola=s.Capo ∧ i.Matricola=s.Impiegato ∧
i.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

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

{ i.(Nome,Matr,Stip), NomeC,MatrC,StipC: i'.(Nome,Matr,Stip) |
i'(Impiegati), s(Supervisione), i(Impiegati) |
i'.Matricola=s.Capo ∧ i.Matricola=s.Impiegato ∧
i.Stipendio > i'.Stipendio}

Esempio 5

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

{Matricola: c, Nome: n |
Impiegati(Matricola: c, Nome: n, Età: e, Stipendio: s) ∧
Supervisione(Capo:c, Impiegato:m) ∧
¬ ∃m'(∃n'(∃e'(∃s'(Impiegati(Matr: m', Nome: n', Età: e', Stip: s') ∧
Supervisione(Capo:c, Impiegato:m') ∧ s' ≤ 40}

{ i.(Matricola, Nome) | s(Supervisione), i(Impiegati) |
i.Matricola=s.Capo ∧
¬(∃i'(Impiegati)(∃s'(Supervisione) (s.Capo=s'.Capo ∧
s'.Impiegato=i'.Matricola ∧
i'.Stipendio ≤ 40)))}

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 cosi' 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 ← *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) ∧ 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

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

- Serve un predicato intensionale

ImpRicchi(Matricola: m, Nome: n, Età: e, Stipendio: s) ← Impiegati (Matricola: m, Nome: n, Età: e, Stipendio: s) , s >40

? ImpRicchi(Matricola: m, Nome: n, Età: e, Stipendio: s)

Esempio 0b

- Trovare matricola, nome ed età di tutti gli impiegati

$$PROJ_{Matricola,\ Nome,\ Età}(Impiegati)$$

{ Matricola: m, Nome: n, Età: e |
Impiegati(Matricola: m, Nome: n, Età: e, Stipendio: s)}

InfoPubbliche(Matricola: m, Nome: n, Età: e)
← Impiegati(Matricola: m, Nome: n, Età: e, Stipendio: s)

? InfoPubbliche(Matricola: m, Nome: n, Età: e)

Esempio 2

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

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

CapiDeiRicchi (Capo:c) ←
ImpRicchi(Matricola: m, Nome: n, Età: e, Stipendio: s),
Supervisione (Capo:c,Impiegato:m)

? CapiDeiRicchi (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 ≤ 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