Lezione 5 – Algebra relazionale III

Prof.ssa Maria De Marsico
demarsico@di.uniroma1.it

SAPIENZA
UNIVERSITÀ DI ROMA

Parte del materiale è tratto dal materiale associato al testo Atzeni, Ceri, Paraboschi, Torlone
Basi di dati - Modelli e linguaggi di interrogazione , McGraw-Hill, 2002.

Condizioni che implicano il quantificatore universale

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

- Fino ad ora abbiamo visto query che implicavano condizioni equivalenti al quantificatore esistenziale

  ∃ (esiste almeno un)

  - Il meccanismo di valutazione consente di rispondere facilmente a questo tipo di query.
  - Infatti … **in qualunque posizione** appaiono nell'espressione di algebra relazionale, la valutazione delle **condizioni** avviene in **sequenza**, tupla per tupla, e quando si incontra una tuple che soddisfa le condizioni, questa viene inserita nel risultato (eventualmente parziale)

Condizioni che implicano il quantificatore universale

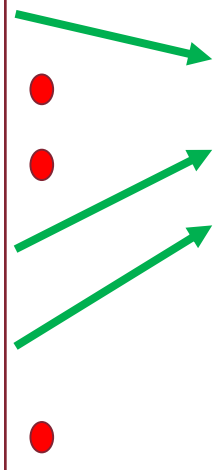
SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

Query: Codici dei clienti che hanno effettuato un ordine (sottinteso:almeno) per più di 100 pezzi

Ordine

C#	A#	N-pezzi
C1	A1	200
C2	A2	100
C1	A2	50
C4	A3	150
C2	A2	200
C1	A3	100

$\sigma_{\text{N-pezzi}>100}(\text{Ordine})$



C#	A#	N-pezzi
C1	A1	200
C4	A3	150
C2	A2	200

Poi proiettiamo sui codici dei clienti (in questo caso potrebbe non interessarci mantenere i duplicati)

Condizioni che implicano il quantificatore universale

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

Query: Codici dei clienti che hanno effettuato un ordine (sottinteso:almeno) per più di 100 pezzi

Ordine

C#	A#	N-pezzi
C1	A1	200
C2	A2	100
C1	A2	50
C4	A3	150
C2	A2	200
C1	A3	100

$\sigma_{\text{N-pezzi}>100}(\text{Ordine})$

C#	A#	N-pezzi
C1	A1	200
C4	A3	150
C2	A2	200

Poi proiettiamo sui codici dei clienti (in questo caso potrebbe non interessarci mantenere i duplicati)

Condizioni che implicano il quantificatore universale

- La condizione potrebbe richiedere la valutazione di gruppi **interi** di tuple **prima** di decidere se inserirle tutte, qualcuna o nessuna nella risposta e …
- … le tuple **non sono ordinate** e …
- … la valutazione delle condizioni avviene in **sequenza, tupla per tupla**, e una volta inserita una tupla nel risultato **non possiamo più eliminarla**
- In questo caso la condizione equivale a valutare il quantificatore universale

  ∀ (per ogni)

  oppure

  !∃ (non esiste nessun)

Esempio 1 reloaded

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

Query : Nomi e città dei clienti che hanno ordinato più di 100 pezzi per almeno un articolo

Cliente

Nome	C#	Città
Rossi	C1	Roma
Rossi	C2	Milano
Bianchi	C3	Roma
Verdi	C4	Roma

Ordine

C#	A#	N-pezzi
C1	A1	100
C2	A2	200
C3	A2	150
C4	A3	200
C1	A2	200
C1	A3	100

$$\pi_{\text{Nome, Città}}(\sigma_{\text{N-pezzi}>100}(\text{Cliente} \triangleright\!\triangleleft \text{Ordine}))$$

Esempio 1 reloaded

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

Nome	C#	Città	A#	N-pezzi
Rossi	C1	Roma	A1	100
Rossi	C1	Roma	A2	200
Rossi	C1	Roma	A3	100
Rossi	C2	Milano	A2	200
Bianchi	C3	Roma	A2	150
Verdi	C4	Roma	A3	200

Cliente $\triangleright\triangleleft$Ordine

Ora basta scandire le tuple una ad una e quando ne troviamo una che soddisfa la condizione N-pezzi>100 la inseriamo nel risultato

Esempio 1 reloaded

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

Nome	C#	Città	A#	N-pezzi
Rossi	C1	Roma	A2	200
Rossi	C2	Milano	A2	200
Bianchi	C3	Roma	A2	150
Verdi	C4	Roma	A3	200

$$\sigma_{\text{N-pezzi}>100}(\text{Cliente} \triangleright\!\triangleleft \text{Ordine})$$

Ora manca solo la proiezione sul nome

$$\pi_{\text{Nome, Città}}(\sigma_{\text{N-pezzi}>100}(\text{Cliente} \triangleright\!\triangleleft \text{Ordine}))$$

In pratica abbiamo risposto alla query: Trovare i clienti per cui esiste (∃) almeno un ordine per più di 100 pezzi

Esempio 1 modificato

Query : Nomi e città dei clienti che hanno **SEMPRE** ordinato più di 100 pezzi per un articolo

Cliente

Nome	C#	Città
Rossi	C1	Roma
Rossi	C2	Milano
Bianchi	C3	Roma
Verdi	C4	Roma

Ordine

C#	A#	N-pezzi
C1	A1	100
C2	A2	200
C3	A2	150
C4	A3	200
C1	A2	200
C1	A3	100

$$???(\sigma_{\text{N-pezzi}>100}(\text{Cliente} \triangleright\!\triangleleft \text{Ordine}))$$

Esempio 1 modificato

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

Nome	C#	Città	A#	N-pezzi
Rossi	C1	Roma	A1	100
Rossi	C1	Roma	A2	200
Rossi	C1	Roma	A3	100
Rossi	C2	Milano	A2	200
Bianchi	C3	Roma	A2	150
Verdi	C4	Roma	A3	200

Cliente $\triangleright\triangleleft$ Ordine

La seconda tupla soddisfa la condizione $\sigma_{\text{N-pezzi}>100}(\text{Cliente} \triangleright\triangleleft \text{Ordine})$

… ma non possiamo inserirla …. Dovremmo **memorizzare** il fatto che Rossi compariva in quella precedente che **non soddisfaceva** la condizione

Esempio 1 modificato

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

Nome	C#	Città	A#	N-pezzi
Rossi	C1	Roma	A2	200
Rossi	C1	Roma	A1	100
Rossi	C1	Roma	A3	100
Rossi	C2	Milano	A2	200
Bianchi	C3	Roma	A2	150
Verdi	C4	Roma	A3	200

Scambiamo le tuple. La prima tupla soddisfa la condizione

$$\sigma_{\text{N-pezzi}>100}(\text{Cliente} \triangleright\!\triangleleft \text{Ordine})$$

… ma non possiamo inserirla …. Dovremmo **prevedere** il fatto che Rossi **non comparirà** in tuple **successive** che **non** soddisfano la condizione (è proprio quello che accade)

Importante: logica

La **negazione** di «**per ogni»**

«∀ elemento la condizione è VERA »

non è «per nessuno»

«∀ elemento la condizione è FALSA»,

ma

«∃ un elemento per cui la condizione è FALSA»

Es. la negazione di

«TUTTI i miei amici si chiamano Paolo»

non è

«NESSUN mio amico si chiama Paolo»,

ma

«ho UN amico che non si chiama Paolo»

Soluzione

- Risolviamo il problema usando la doppia negazione
- Eseguiamo la query con la condizione **negata**
- Troviamo gli oggetti che ALMENO una volta soddisfano la condizione CONTRARIA → non possono soddisfare SEMPRE quella che ci interessa
- ELIMINIANO gli oggetti ottenuti dalla risposta con l'operatore differenza

Esempio 1 modificato

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

Nome	C#	Città	A#	N-pezzi
Rossi	C1	Roma	A1	100
Rossi	C1	Roma	A2	200
Rossi	C1	Roma	A3	100
Rossi	C2	Milano	A2	200
Bianchi	C3	Roma	A2	150
Verdi	C4	Roma	A3	200

$$\text{Cliente} \triangleright\!\triangleleft \text{Ordine}$$

$$\sigma_{\text{N-pezzi}<=100}(\text{Cliente} \triangleright\!\triangleleft \text{Ordine})$$

Esempio 1 modificato

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

Nome	C#	Città	A#	N-pezzi
Rossi	C1	Roma	A1	100
Rossi	C1	Roma	A3	100

$$\sigma_{\text{N-pezzi}<=100}(\text{Cliente} \triangleright\!\triangleleft \text{Ordine})$$

Facciamo la proiezione sul nome e città

$$\pi_{\text{Nome, Città}}(\sigma_{\text{N-pezzi}<=100}(\text{Cliente} \triangleright\!\triangleleft \text{Ordine}))$$

Sono quelli che NON ci interessano.
Il Rossi di Roma ha anche un ordine con 100 pezzi, ma noi vogliamo quelli che ne hanno SEMPRE ordinati di più.

$$\pi_{\text{Nome, Città}}(\text{Cliente} \triangleright\!\triangleleft \text{Ordine}) - \pi_{\text{Nome, Città}}(\sigma_{\text{N-pezzi}<=100}(\text{Cliente} \triangleright\!\triangleleft \text{Ordine}))$$

Restano il Rossi di Milano, Bianchi e Verdi che hanno SEMPRE ordinato più di 100 pezzi (non ne hanno ordinato MAI meno di 100)

Esempio 1 - nota

Se supponiamo di inserire ul cliente solo in concomitanza con il primo ordine, allora potremmo usare anche l'espressione più semplice

$\pi_{\text{Nome, Città}}(\text{Cliente}) - \pi_{\text{Nome, Città}}(\sigma_{\text{N-pezzi<=100}}(\text{Cliente} \triangleright\!\triangleleft \text{Ordine}))$

Se questo non è vero, in Cliente potremmo avere clienti **che non hanno mai** effettuato ordini, che sarebbero **preservati** dalla sottrazione. Alla fine avremmo quindi un risultato scorretto. Quindi occorre utilizzare il join **che assicura di effettuare la sottrazione non a partire da tutti i clienti, ma solo da quelli che hanno effettuato almeno un ordine.**

$\pi_{\text{Nome, Città}}(\text{Cliente} \triangleright\!\triangleleft \text{Ordine}) - \pi_{\text{Nome, Città}}(\sigma_{\text{N-pezzi<=100}}(\text{Cliente} \triangleright\!\triangleleft \text{Ordine}))$

Nel dubbio, la seconda soluzione è sempre corretta

Esempio 1 modificato

- **NOTA BENE** - Nell'esempio appena concluso, se avessimo proiettato solo sul nome, nella sottrazione sarebbe sparito anche il Rossi di Milano. Quindi anche se fossimo stati veramente interessati solo ai nomi, avremmo avuto un risultato mancante

- **ATTENZIONE. -** Valutare con cura le proiezioni su gruppi di attributi che non sono unici. C'è il rischio di perdere informazione e avere risultati errati

Esempio 1 modificato bis

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

Query : Nomi e città dei clienti che non hanno **MAI** ordinato più di 100 pezzi per un articolo

Cliente

Nome	C#	Città
Rossi	C1	Roma
Rossi	C2	Milano
Bianchi	C3	Roma
Verdi	C4	Roma

Ordine

C#	A#	N-pezzi
C1	A1	100
C2	A2	200
C3	A2	150
C4	A3	200
C1	A2	200
C1	A3	100

$$???(\sigma_{\text{N-pezzi}<=100}(\text{Cliente} \triangleright\!\triangleleft \text{Ordine}))$$

Esempio 1 modificato bis

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

Nome	C#	Città	A#	N-pezzi
Rossi	C1	Roma	A1	100
Rossi	C1	Roma	A2	200
Rossi	C1	Roma	A3	100
Rossi	C2	Milano	A2	200
Bianchi	C3	Roma	A2	150
Verdi	C4	Roma	A3	200

Cliente $\triangleright\triangleleft$ Ordine

La prima tupla soddisfa la condizione $\sigma_{\text{N-pezzi<=100}}$(Cliente $\triangleright\triangleleft$ Ordine)

… ma non possiamo inserirla …. Dovremmo **prevedere** il fatto che Rossi compare in quella successiva che **non soddisfa** la condizione.

Esempio 1 modificato

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

Nome	C#	Città	A#	N-pezzi
Rossi	C1	Roma	A1	100
Rossi	C1	Roma	A2	200
Rossi	C1	Roma	A3	100
Rossi	C2	Milano	A2	200
Bianchi	C3	Roma	A2	150
Verdi	C4	Roma	A3	200

$$\text{Cliente} \triangleright\!\triangleleft \text{Ordine}$$

Applichiamo il ragionamento di prima e selezioniamo prima i nomi e città di clienti che NON ci interessano

$$\sigma_{\text{N-pezzi}>100}(\text{Cliente} \triangleright\!\triangleleft \text{Ordine})$$

Esempio 1 modificato

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

Nome	C#	Città	A#	N-pezzi
Rossi	C1	Roma	A2	200
Rossi	C2	Milano	A2	200
Bianchi	C3	Roma	A2	150
Verdi	C4	Roma	A3	200

$$\sigma_{\text{N-pezzi}>100}(\text{Cliente} \triangleright\!\triangleleft \text{Ordine})$$

Facciamo la proiezione sul nome e città

$$\pi_{\text{Nome, Città}}(\sigma_{\text{N-pezzi}>100}(\text{Cliente} \triangleright\!\triangleleft \text{Ordine}))$$

Sono quelli che NON ci interessano.

$$\pi_{\text{Nome, Città}}(\text{Cliente} \triangleright\!\triangleleft \text{Ordine}) - \pi_{\text{Nome, Città}}(\sigma_{\text{N-pezzi}>100}(\text{Cliente} \triangleright\!\triangleleft \text{Ordine}))$$

La query restituisce un risultato **VUOTO**, perché anche il Rossi di Roma ha ordinato una volta un numero di pezzi maggiore di 100

Condizioni che richiedono il prodotto (join SQL-style) di una relazione con sé stessa

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

- Come negli esempi precedenti abbiamo visto casi in cui oggetti di relazioni diverse vengono associati, ci sono anche casi in cui sono in qualche modo associati oggetti della stessa relazione.

Condizioni che richiedono il prodotto (join SQL-style) di una relazione con sé stessa

Query: Nomi e codici degli impiegati che guadagnano quanto o più del loro capo

Impiegati

Nome	C#	Dipart	Stip	Capo#
Rossi	C1	B	100	C3
Pirlo	C2	A	200	C3
Bianchi	C3	A	500	NULL
Verdi	C4	B	200	C2
Neri	C5	B	150	C1
Tosi	C6	B	100	C1

Condizioni che richiedono il prodotto (join SQL-style) di una relazione con sé stessa

- Le informazioni sullo stipendio di un impiegato e su quello del suo capo si trovano in tuple **diverse**

- **Per poter confrontare valori di attributi diversi , questi devono trovarsi nella stessa tupla**

- Che si fa?

Condizioni che richiedono il prodotto (join SQL-style) di una relazione con sé stessa

- Le informazioni sullo stipendio di un impiegato e su quello del suo capo si trovano in tuple **diverse**

- Creiamo una **copia** della relazione ed effettuiamo un prodotto in maniera da combinare le informazioni su di un impiegato con quelle del suo capo, che a questo punto possono essere confrontate. ImpiegatiC sarà collegata in join ad Impiegati combinando le tuple col valore di C# uguale a Capo#. In questo modo accodiamo i dati del capo a quelli dell'impiegato

- Utilizziamo la **ridenominazione**, e facciamo in modo che i nuovi nomi aiutino a distinguere il ruolo delle due parti nel join finale

Esempio 1

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

$$ImpiegatiC = \rho_{Nome, C\#, Dipart, Stip, Capo\# \leftarrow CNome, CC\#, Cdipart, Cstip, Ccapo\#}(Impiegati)$$

Nota= poiché abbiamo attributi con lo stesso nome nelle due relazioni, per distinguerli usiamo una ridenominazione

Nome	C#	Dipart	Stip	Capo#	CNome	CC#	CDipart	CStip	CCapo#
Rossi	C1	B	100	C3	Bianchi	C3	A	500	NULL
Pirlo	C2	A	200	C3	Bianchi	C3	A	500	NULL
Verdi	C4	B	200	C2	Pirlo	C2	A	200	C3
Neri	C5	B	150	C1	Rossi	C1	B	100	C3
Tosi	C6	B	100	C1	Rossi	C1	B	100	C3

$$\sigma_{Capo\#=CC\#}(Impiegati \times ImpiegatiC)$$

ATTENZIONE! Qui il join naturale (senza ridenominazione) ci farebbe combinare un impiegato con sé stesso! Inoltre C3 non ha capo, quindi la tupla non entra nel join dal lato impiegato

Condizioni che richiedono il prodotto (join SQL-style) di una relazione con sé stessa

Ricordiamo che l'algebra relazionale è un linguaggio formale, e nel caso di join di due istanze della stessa relazione potremmo utilizzare diverse strategie, basate su diverse convenzioni.

Condizioni che richiedono il prodotto (join SQL-style) di una relazione con sé stessa

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

- Possiamo assumere, anche dopo il prodotto, di avere ancora a disposizione il nome della relazione di partenza. In quel caso è sufficiente salvare la relazione in una nuova variabile relazionale con nome diverso da utilizzare come seconda istanza lasciano invariati i nomi degli attributi

ImpiegatiC = Impiegati $\quad \sigma_{\text{Impiegati.Capo\#=ImpiegatiC.C\#}}(\textit{Impiegati} \times \textit{ImpiegatiC})$

- Possiamo effettuare il join della relazione con una istanza temporanea in cui i nomi degli attributi sono stati ridenominati

$\textit{Impiegati} \times \rho_{\textit{Nome, C\#, Dipart, Stip, Capo\#} \leftarrow \textit{CNome, CC\#, Cdipart, Cstip, Ccapo\#}}(\textit{Impiegati})$

$\sigma_{\text{Capo\#=CC\#}}(\textit{Impiegati} \times \rho_{\textit{Nome, C\#, Dipart, Stip, Capo\#} \leftarrow \textit{CNome, CC\#, Cdipart, Cstip, Ccapo\#}}(\textit{Impiegati}))$

- Possiamo salvare la relazione in una nuova variabile relazionale con nome diverso da utilizzare come seconda istanza (ed eventualmente da riutilizzare), con i nomi degli attributi ridenominati, che a quel punto sono distinguibili

$\textit{ImpiegatiC} = \rho_{\textit{Nome, C\#, Dipart, Stip, Capo\#} \leftarrow \textit{CNome, CC\#, Cdipart, Cstip, Ccapo\#}}(\textit{Impiegati})$

$\sigma_{\text{Capo\#=CC\#}}(\textit{Impiegati} \times \textit{ImpiegatiC})$

Esempio 1

- A questo punto basta confrontare lo stipendio dell'impiegato con quello del capo per selezionare gli impiegati che ci interessano, e infine proiettare

$$r=(\sigma_{Stip>=CStip}(\sigma_{Capo\#=CC\#}(Impiegati \times ImpiegatiC)))$$

Nome	C#	Dipart	Stip	Capo#	CNome	CC#	CDipart	CStip	CCapo#
Verdi	C4	B	200	C2	Pirlo	C2	A	200	C3
Neri	C5	B	150	C1	Rossi	C1	B	100	C3
Tosi	C6	B	100	C1	Rossi	C1	B	100	C3

Nome	C#
Verdi	C4
Neri	C5
Tosi	C6

$\pi_{Nome, C\#}(r)$

Potremmo proiettare anche su nome e codice del capo

Esempio 2

- **Query:** Nomi e codici dei capi che guadagnano più di tutti i loro impiegati

Ricordiamo dalla scorsa lezione che non è possibile ripondere a queste query senza la doppia negazione, in quanto le tuple vengono scandite una per volta e non possiamo memorizzare né prevedere se un atupla precedente o successiva soddifa la condizione

Esempio 2

- Riprendiamo la query dell'esempio precedente che trova gli impiegati che guadagnano **quanto o più del loro capo**. I capi che compaiono **anche una sola volta** in questo risultato (cioè C1 e C2) sono quelli che **NON ci interessano**

$$r=(\sigma_{Stip>=CStip}(\sigma_{Capo\#=CC\#}(Impiegati \times ImpiegatiC)))$$

Nome	C#	Dipart	Stip	Capo#	CNome	CC#	CDipart	CStip	CCapo#
Verdi	C4	B	200	C2	Pirlo	C2	A	200	C3
Neri	C5	B	150	C1	Rossi	C1	B	100	C3
Tosi	C6	B	100	C1	Rossi	C1	B	100	C3

$\pi_{CNome,CC\#}(\sigma_{Capo\#=CC\#}(Impiegati \times ImpiegatiC)) - \pi_{CNome,CC\#}(r)$ →

Bianchi	C3

Esempio 2

- Lo stesso esercizio può essere svolto in altri modi altrettanto corretti, ma attenzione invece a quelli non corretti

$$\pi_{Nome,C\#}(Impiegati) - \pi_{CNome,CC\#}(r)$$

è sbagliato perché ci sono impiegati che non sono capi e non verrebbero sottratti alla prima proiezione.

- $(\pi_{Nome,Capo\#}(Impiegati) - \pi_{CNome,CC\#}(r)$

è sbagliato perché nella prima proiezione in nome è dell'impiegato e il codice è del capo.

- Una alternativa corretta è

$$\pi_{Nome,C\#}((\pi_{Capo\#}(Impiegati) - \pi_{CC\#}(r)) \triangleright\!\triangleleft_{Capo\#=Impiegati.C\#} Impiegati)$$

che estrae prima i codici, effettua un join per ottenere le altre informazioni (un capo è anche un impiegato) e poi effettua la proiezione. Attenzione alle proiezioni! Vogliamo che i codici da cui sottraiamo siano sicuramente di capi.