Espressioni relazionali

Gabriele Frassi

Indice

1 Interrogazioni

L'interrogazione è un'operazione di lettura che non modifica la base di dati. In certe circostanze un'interrogazione può coinvolgere più tabelle! Abbiamo due tipi di semantiche:

- **dichiarativa**, in cui si esprimono le proprietà del risultato. Essa ricorre al *calcolo relazionale* e consiste nell'effettiva semantica del linguaggio! Le interrogazioni sono espresse ad alto livello.
- **operazionale**, si specificano le modalità di calcolo adottate dal gestore della base di dati per ottenere il risultato. In questo caso si ricorre all'*algebra relazionale*. Poichè si parla di modalità di calcolo possiamo individuare i costi per l'esecuzione di una certa interrogazione

1.1 *query processor*

All'interno del gestore abbiamo un modulo detto *query processor*, dove è definito il processo di esecuzione delle interrogazioni. Ciò permette al gestore di definire una strategia di esecuzione ottimale. Si hanno tre fasi:

- **Analisi lessicale, sintattica e semantica** della query scritta: si ottiene una corrispondente espressione algebrica;
- **Ottimizzazione algebrica** dell'espressione: si ottiene una nuova espressione algebrica attraverso ottimizzazioni valide in ogni caso (Esempio: *push selections down* e *push projections down*)
- **Ottimizzazione basata sui costi**: sfruttando quanto contenuto nel cosiddetto *catalogo*, cioè informazioni quantitative (*profili*) riguardanti il database (numero di tuple, dimensione della tupla o dei valori, numero di valori distinti, valori minimi e massimi degli attributi...), possiamo stimare la dimensione dei risultati intermedi ed effettuare ulteriori ottimizzazioni (riparleremo di questa ottimizzazione nell'ultima lezione).

Ottimizzazione algebrica

Le due fasi di ottimizzazione si basano sull'euristica: sulla base dei dati a disposizione decidiamo quali procedure adottare per eseguire un'istruzione. Parlare di "euristica", tuttavia, è un po' improprio: il processo si basa soprattutto sulla nozione di equivalenza.

Equivalenza Due espressioni sono equivalenti se producono lo stesso risultato!

2 Algebra relazionale

Nell'algebra relazionale abbiamo operatori applicati su una o più relazioni. Gli operatori producono nuove relazioni e possono essere composti. Analizzeremo **operatori su insiemi** (unione, intersezione e differenza) ed **operatori su relazioni** (ridenominazione, selezione, proiezione, join). I primi operatori sono applicati a relazioni che presentano gli stessi attributi: la relazione prodotta avrà gli stessi attributi. I secondi, ad eccezione del join (il più complesso tra quelli che vedremo), sono monadici.

Nelle successive definizioni considereremo X come l'insieme degli attributi.

2.1 Unione ($r_1 \cup r_2$)

Date due relazioni $r_1(X), r_2(X)$, l'unione $r_1(X) \cup r_2(X)$ consiste in una nuova relazione che contiene sia le tuple di r_1 che quelle di r_2.

Esempio dalle diapositive Abbiamo le relazioni **laureati_triennali** e **laureati_magistrali**. Ottengo una nuova relazione che elenca le persone che hanno conseguito almeno una laurea. Nei risultati della nuova relazione vediamo Neri e Verdi che hanno conseguito sia la laurea triennale che quella magistrale, un altro Neri che ha solo la laurea magistrale, Rossi che ha solo la laurea triennale.

Laureati triennali

Matricola	Nome	Età
7274	Rossi	32
7432	Neri	24
9824	Verdi	25

Laureati magistrali

Matricola	Nome	Età
9297	Neri	33
7432	Neri	24
9824	Verdi	25

Laureati triennali ∪ Laureati magistrali

Matricola	Nome	Età
7274	Rossi	32
7432	Neri	24
9824	Verdi	25
9297	Neri	33

2.2 Intersezione ($r_1 \cap r_2$)

Date due relazioni $r_1(X), r_2(X)$, l'intersezione $r_1(X) \cap r_2(X)$ consiste in una nuova relazione che contiene solo le tuple appartenenti ad entrambe le relazioni.

Esempio dalle diapositive Prendiamo lo stesso esempio di prima. Ottengo una nuova relazione che elenca le persone che hanno preso sia la laurea magistrale che quella triennale. Gli unici che hanno conseguito entrambe le lauree sono Neri e Verdi: nell'istanza avrò soltanto le loro tuple.

Laureati triennali

Matricola	Nome	Età
7274	Rossi	32
7432	Neri	24
9824	Verdi	25

Laureati magistrali

Matricola	Nome	Età
9297	Neri	33
7432	Neri	24
9824	Verdi	25

Laureati triennali ∩ Laureati magistrali

Matricola	Nome	Età
7432	Neri	24
9824	Verdi	25

2.3 Differenza ($r_1 - r_2$)

Date due relazioni $r_1(X), r_2(X)$, la differenza $r_1(X) - r_2(X)$ consiste in una nuova relazione che contiene le tuple di r_1 che non appartengono anche ad r_2. L'operatore è rappresentato dal segno meno.

Esempio dalle diapositive Idem con patate. Ottengo una relazione in cui si hanno gli elementi di **laureati_triennali** non presenti in **laureati_magistrali**, cioè coloro che hanno conseguito la laurea triennale ma non quella magistrale.
Nella prima relazione abbiamo Rossi, Neri e Verdi: osserviamo che Neri e Verdi sono presenti anche nella seconda relazione. Segue che l'unica tupla presente nell'istanza sarà quella di Rossi.

Laureati triennali

Matricola	Nome	Età
7274	Rossi	32
7432	Neri	24
9824	Verdi	25

Laureati magistrali

Matricola	Nome	Età
9297	Neri	33
7432	Neri	24
9824	Verdi	25

Laureati triennali – Laureati magistrali

Matricola	Nome	Età
7274	Rossi	32

2.4 Ridenominazione ($\rho_{a_1,\ldots,a_n \leftarrow b_1,\ldots,b_n}$)

L'operatore monadico di ridenominazione ci permette di modificare lo schema di una relazione alterando gli attributi. Questa cosa ci permetterà di applicare gli operatori di insieme a relazioni che hanno schema simile ma non equivalente.

Esempio dalle diapositive Abbiamo le relazioni **paternita**($padre, figlio$) e **maternita**($madre, figlio$) a cui voglio applicare l'operatore unione. Essi hanno schema simile, ma un attributo diverso. Mediante l'operatore di ridenominazione modifico l'attributo *padre* in *genitore*. Faccio la stessa cosa con *madre*. A questo punto posso applico l'operatore di insieme.

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

Genitore	Figlio
Adamo	Abele
Adamo	Caino
Abramo	Isacco

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

Genitore	Figlio
Eva	Abele
Eva	Set
Sara	Isacco

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

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

2.5 Selezione (σ_F) - decomposizione orizzontale

L'operatore monadico di selezione restituisce una nuova relazione con lo stesso schema ma con solo una parte delle tuple. Viene mostrato un sottoinsieme i cui elementi sono determinati a partire da un'espressione booleana F: se l'espressione è vera la tupla è considerata appartenente alla nuova relazione, altrimenti viene esclusa.

Esempio dalle diapositive Ho una relazione **Impiegati**($matricola, cognome, filiale, stipendio, eta$). Voglio mostrare soltanto gli impiegati con uno stipendio superiore ai 50.000 euro. Posso rendere la cosa più articolata mostrando soltanto gli impiegati che soddisfano la condizione precedente e che lavorano nella filiale di Milano.

$\sigma_{(Stipendio > 50000)\ AND\ (Filiale = 'Milano')}(Impiegati)$

Matricola	Cognome	Filiale	Stipendio
5998	Neri	Milano	64000

Attenzione ai valori nulli Prendiamo la stessa relazione e consideriamo le persone con $eta > 40$. Le tuple con età nulla vengono automaticamente escluse: ciò può essere evitato indicando che il valore può essere anche nullo (Ho le forme *IS NULL* e *IS NOT NULL*).

$$\sigma_{Eta>40}(Impiegati) \qquad \sigma_{(Eta>40)\ OR\ (Eta\ \textbf{IS NULL})}(Impiegati)$$

2.6 Proiezione (π_Y) - decomposizione verticale

L'operatore monadico di proiezione restituisce una nuova relazione che contiene un insieme di tuple ristrette agli attributi Y (ho un sottoinsieme degli attributi, fondamentalmente).

Esempio dalle diapositive Riprendiamo l'esempio utilizzato negli operatori di insieme: creo una nuova relazione in cui mostro soltanto matricole e cognomi.

Matricola	Cognome
7309	Neri
5998	Neri
9553	Rossi
5698	Rossi

$\pi_{Matricola,\ Cognome}(Impiegati)$

Cardinalità delle proiezioni Può capitare che una proiezione produca una relazione con un numero inferiore di tuple. Ciò può avvenire escludendo la chiave *matricola*. Osserviamo che sono presenti nella relazione iniziale due Neri e due Rossi. Nella relazione finale ho due Neri e un *solo* Rossi poichè abbiamo ignorato l'attributo che identifica le tuple in modo univoco: si osserva, inoltre, che entrambi i Rossi lavorano nella filiale di Roma.

Cognome	Filiale
Neri	Napoli
Neri	Milano
Rossi	Roma

$\pi_{Cognome,\ Filiale}(Impiegati)$

2.7 Composizione degli operatori

Come già detto posso creare una composizione di operatori: per esempio posso unire l'operatore selezione con l'operatore proiezione. Nella composizione è necessario porre attenzione all'ordine degli operatori: un ordine diverso può portare a risultati diversi.

Matricola	Cognome
7309	Rossi
5998	Neri
5698	Neri

$$\pi_{\text{Matricola,Cognome}} (\sigma_{\text{Stipendio} > 50} (\text{Impiegati}))$$

Adesso spingiamoci oltre: con gli operatori introdotti fino ad ora non possiamo correlare relazioni profondamente diverse tra loro!

2.8 Operatore Join

Con l'operatore JOIN ossiamo correlare dati appartenenti a relazione diverse!

2.8.1 Prodotto cartesiano ($r_1 \times r_2$)

Operatore binario, di fatto il più costoso per il DBMS. Abbiamo due relazioni: combiniamo le tuple di r_1 con le tuple di r_2. Ottengo una nuova relazione dove

- lo schema consiste nell'unione degli attributi delle due relazioni
- l'istanza è caratterizzata da tuple ottenute mediante il prodotto cartesiano matematico.

Impiegato	Reparto
Rossi	A
Neri	B
Bianchi	B

Reparto	Capo
B	Mori
C	Bruni

R.Impiegato	R.Reparto	Q.Capo	Q.Reparto
Neri	B	Mori	B
Bianchi	B	Mori	B
Neri	B	Bruni	C
Bianchi	B	Bruni	C
Rossi	A	Bruni	C
Rossi	A	Mori	B

2.8.2 Join naturale ($r_1 \bowtie r_2$)

Il *join naturale* è un operatore binario che restituisce una relazione con schema uguale all'unione degli attributi degli schemi di $r_1(X_1)$ e $r_2(X_2)$ $(X_1 \cup X_2)$. Si individua che

$$r_1 \bowtie r_2 = \{t \in X_1 \cup X_2 \mid t[X_1] \in r_1, t[X_2] \in r_2\}$$

Il join naturale consiste in un insieme di tuple da cui ottengo tuple appartenenti alla relazione r_1 se applico una proiezione limitata a X_1 e tuple appartenenti alla relazione r_2 se applico una proiezione limitata a X_2. Abbiamo due situazioni:

- **presenza di attributi con lo stesso nome**: le tuple vengono costituite se il valore degli attributi comuni è uguale
- **senza attributi con nome comune**: quello che si ottiene equivale al prodotto cartesiano. Ho un numero di tuple pari al prodotto delle cardinalità degli operandi (le tuple sono tutte combinabili)

Osservazione Solitamente non è possibile tornare alle relazioni originali dopo aver applicato il join.

Esempio dalle diapositive Riprendiamo le relazioni *impiegati* e *caporeparti*. Rossi non è tra le tuple poichè non è combinabile con nessun capo reparto. Si osserva che il capo del reparto C non può essere combinato con nessun impiegato (non si hanno impiegati al reparto C).

Impiegato	Reparto
Rossi	A
Neri	B
Bianchi	B

Reparto	Capo
B	Mori
C	Bruni

Impiegato	Reparto	Capo
Neri	B	Mori
Bianchi	B	Mori

Potrebbe succedere che per un reparto (il B in questo caso) si abbiano due capireparto: Neri e Bianchi (del reparto B) vengono combinati due volte!

Impiegato	Reparto
Neri	B
Bianchi	B
Verdi	A

Reparto	Capo
B	Mori
B	Bruni
A	Bini

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

Esempio senza attributi comuni

Impiegati

Impiegato	Reparto
Rossi	A
Neri	B
Bianchi	B

Reparti

Codice	Capo
A	Mori
B	Bruni

Impiegati $\triangleright\triangleleft$ 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

In generale

- Date due relazioni $R_1(X_1), R_2(X_2)$ si osserva che la proiezione di X_1 del join naturale tra R_1 e R_2 è inclusa in R_1.

  $$\pi_{X_1}(R_1 \bowtie R_2) \subseteq R_1$$

  Se non si hanno attributi comuni avviene il prodotto cartesiano e il risultato dell'espressione coincide con R_1. Se si hanno attributi comuni il risultato non coincide con il prodotto

cartesiano e due tuple vengono associate solo se si hanno le condizioni. Segue che il non-JOIN di alcune tabelle potrebbe comportarmi l'esclusione di alcune tuple di R_1 e quindi avere come risultato un sottoinsieme di R_1.

- Data una relazione $R(X)$, dove $X = X_1 \cup X_2$, il join naturale tra la proiezione di X_1 di R e la proiezione di X_2 di R contiene R

$$(\pi_{X_1}(R)) \bowtie (\pi_{X_2}(R)) \supseteq R$$

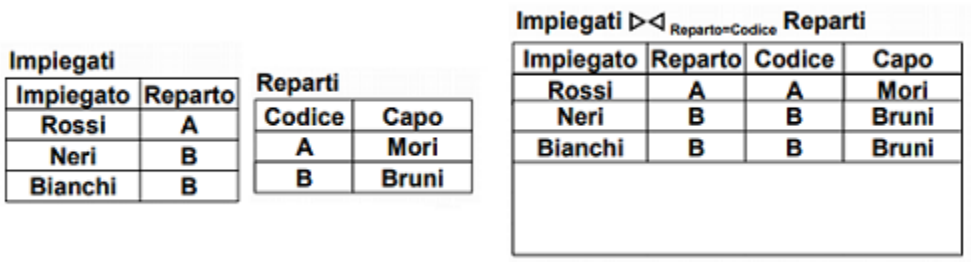
  Se non si hanno attributi comuni tra X_1 e X_2 segue un prodotto cartesiano che mi porta ad avere un numero di tuple maggiore rispetto a prima. Ho quindi un insieme più grande che contiene l'insieme R iniziale. Se ho attributi comuni tra X_1 e X_2 allora può esserci la possibilità che il risultato sia uguale ad R originario.

2.8.3 *theta-join* (spesso *equi-join*)

Posso ridurre il prodotto cartesiano attraverso una relazione del tipo $\sigma_F(R_1 \times R_2)$. La stessa operazione può essere fatta attraverso un operatore derivato detto *theta-join*: prendo soltanto le tuple che fanno JOIN e che soddisfano delle condizioni.

$$R_1 \bowtie_F R_2$$

La condizione F è spesso una congiunzioni di atomi di confronto caratterizzati da un operatore di confronto e attributi di relazioni diverse. Se l'operatore di confronto è l'uguale allora si ha l'*equi-join*.
Riprendiamo l'esempio di prima dove non si hanno attributi comuni. Svolgiamo l'equi-join stabilendo un legame valido tra tuple se Reparto=Codice.



Impiegati

Impiegato	Reparto
Rossi	A
Neri	B
Bianchi	B

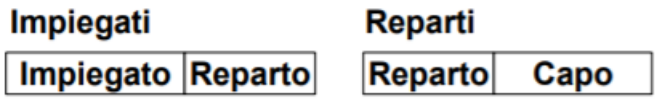
Reparti

Codice	Capo
A	Mori
B	Bruni

Impiegati $\triangleright\triangleleft_{\text{Reparto=Codice}}$ Reparti

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

2.9 JOIN operatore non primitivo



Impiegati

Impiegato	Reparto

Reparti

Reparto	Capo

Il Join non è un operatore primitivo (diciamo, un "prodotto cartesiano schietto"). Individuiamo che

Relazione tra JOIN Naturale e prodotto cartesiano

$Impiegati \bowtie Reparti = \pi_{Impiegato,Reparto,Capo}(\sigma_{I.reparto=reparto}(\rho_{I.reparto \leftarrow Reparto}(Impiegati) \times Reparti))$

- Applico la ridenominazione alla relazione *Impiegati* per poter usare la selezione
- Eseguo il prodotto cartesiano tra la relazione $\rho_{I.Reparto \leftarrow Reparto}(Impiegati)$ e *Reparti*.
- Applico la selezione, scegliendo i record che soddisfano la condizione $I.Reparto = Reparto$
- Proietto gli attributi $Impiegato, Reparto, Capo$. $I.Reparto$ ha lo stesso valore di *Reparto*, non serve proiettarlo nuovamente.

Relazione tra JOIN Naturale ed equi-join

$Impiegati \bowtie Reparti = \pi_{Impiegato,Reparto,Capo}(\rho_{I.reparto \leftarrow Reparto}(Impiegati) \bowtie_{I.Reparto=Reparto} Reparti)$

- Applico la ridenominazione alla relazione *Impiegati* per poter usare la selezione
- Eseguo l'equi-JOIN tra la relazione $\rho_{I.Reparto \leftarrow Reparto}(Impiegati)$ e *Reparti*, scegliendo soltanto le combinazioni di record che soddisfano l'uguaglianza $I.Reparto = Reparto$
- Proietto gli attributi $Impiegato, Reparto, Capo$. $I.Reparto$ ha lo stesso valore di *Reparto*, non serve proiettarlo nuovamente.

2.10 Equivalenza di espressioni

Due espressioni sono equivalenti se producono lo stesso risultato qualunque sia l'istanza attuale della base di dati. L'equivalenza è un concetto importante poichè i DBMS, molto spesso, eseguono espressioni equivalenti a quelle date, meno costose. Vediamo due esempi di euristica fondamentale in cui si applica uno dei principi base dell'ottimizzazione: esecuzione di selezioni e proiezioni il più presto possibile per ridurre la dimensioni dei risultati intermedi (e quindi il costo dell'operazione)

2.10.1 *Pushing selections down*

Dato un attributo A di R_2, che è attributo di interesse, individuo che

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

Le condizioni della selezione riguardano un attributo di R_2: eseguo la selezione prima di fare JOIN.

2.10.2 *Pushing projections down*

Date due relazioni $R_1(X_1)$ e $R_2(X_2)$ con $Y_2 \subseteq X_2$, individuo che

$$\pi_{X_1 Y_2}(R_1 \bowtie R_2) = R_1 \bowtie \pi_{Y_2}(R_2)$$

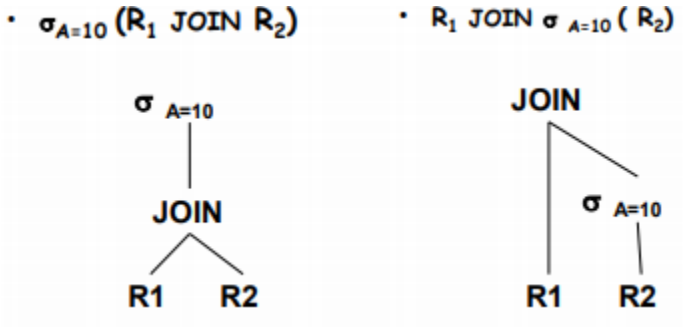
Proietto solo una parte degli attributi di X_2: eseguo la proiezione prima di fare JOIN.

2.11 Procedura euristica dell'ottimizzatore

- Decomporre le selezioni congiuntive in successive selezioni atomiche
- Anticipare il più possibile le selezioni
- In una sequenza di selezioni anticipare le più selettive
- Combinare prodotti cartesiani e selezioni per formare JOIN
- Anticipare il più possibile le proiezioni (anche introducendone di nuove)

2.12 Alberi per la rappresentazione di interrogazioni

Possiamo utilizzare gli alberi per rappresentare le nostre interrogazioni: le foglie consistono nei dati (relazioni, file), i nodi intermedi negli operatori applicati.



3 Estensione dell'algebra relazionale

Il modello relazionale può essere facilmente estesso a comprendere operatori SQL non direttamente riconducibili agli operatori algebrici introdotti. Ciò non comporta una modifica del modello.

3.1 JOIN Esterno

Ho tre tipi di OUTER JOIN: left, right e full. Questi permettono di combinare tuple che normalmente non fanno JOIN.

- Il LEFT JOIN: mantiene tutte le tuple della prima relazione
- Il RIGHT OUTER JOIN: mantiene tutte le tuple della seconda relazione
- Il FULL OUTER JOIN: mantiene tutte le tuple

Esempio di JOIN *Trovare la matricola dei capi degli impiegati che guadagnano tutti più di 40.000 euro*

Impiegati

Matricola	Nome	Età	Stipendio
7309	Rossi	34	45000
5998	Bianchi	37	38000
9553	Neri	42	35000
5698	Bruni	43	42000
4076	Mori	45	50000
8123	Lupi	46	60000

Supervisione

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

Si individuano impiegati non subordinati a nessuno, cioè i capi stessi. Con un JOIN normale (con I.Matricola e S.Impiegato) i record riguardanti questi impiegati sparirebbero. Facciamo il JOIN sinistro

$$\pi_{Capo}(\sigma_{MatricolaISNULL}(Supervisione =\bowtie_{Impiegato=Matricola} \sigma_{Stipendio\leq 40.000}(Impiegati)))$$

ottenendo

Matricola	Nome	Età	Stipendio	Impiegato	Capo
5998	Bianchi	37	38000	5998	5698
9553	Neri	42	35000	9553	4076
NULL	NULL	NULL	NULL	4076	8123

Ho due impiegati che soddisfano le condizioni e un capo che non ha impiegati che guadagnano più di 40.000 euro. Ho tuple che fanno JOIN e tuple che non hanno fatto JOIN. Quanto fatto permette di svolgere l'operazione di differenza, che vedremo più avanti

3.2 Proiezione generalizzata

$$\pi_{F_1,F_2,F_3}(E)$$

Dove F_1, F_2, F_3 sono espressioni aritmetiche su attributi di E e costanti. Creo "un nuovo" attributo derivante da un'espressione algebrica dipendente da altri attributi e da costanti. Vediamo un esempio con la seguente tabella

Conto

Cliente	Credito	Spese
Andrea	6000	1000
Andrea	4000	500
Maria	10000	2000
Anna	3000	1500
Filippo	3000	1000
Luigi	5000	1800
Franco	5000	2000
Maria	6000	2000
Andrea	10000	5000
Anna	5000	1000

Posso scrivere, per esempio $\pi_{Cliente,Credito-Spese}(Conto)$, ottenendo

Conto

Cliente	Credito	Spese
Andrea	6000	1000
Andrea	4000	500
Maria	10000	2000
Anna	3000	1500
Filippo	3000	1000
Luigi	5000	1800
Franco	5000	2000
Maria	6000	2000
Andrea	10000	5000
Anna	5000	1000

3.3 Funzioni aggregate

Si possono usare nelle espressioni dei nomi di funzione (operatori) che si applicano a insiemi e producono un valore scalare come risultato. Gli operatori aggregati sono:

- **Somma**: $sum_{Spese}(Conto)$
- **Massimo**: $max_{Credito}(Conto)$
- **Minimo**: $min_{Credito}(Conto)$
- **Conteggio**: $count_{Cliente}(Conto)$
- **Conteggio di elementi distinti**: count-distinct$_{Cliente}(Conto)$

3.4 Raggruppamento

Posso raggruppare gli elementi di una relazione attraverso un apposito operatore. Posso raggruppare dei conti in base al codice cliente ravvicinando i conti correnti dei vari clienti. Posso applicare a questi gruppi degli operatori aggregati e ottenere un result set con una riga per ogni cliente.

$$_{\textbf{Cliente}}G_{sum(Credito)}(Conto)$$

Posso avere più attributi con cui compiere il raggruppamento (a sinistra), ma anche più funzioni di aggregazione (a destra).

3.5 Divisione

Operatore di tipo universale (parola chiave: tutti). Posso porlo sintatticamente all'interno di una query, ovviamente non è primitivo perchè corrisponde a un'espressione estremamente complessa.

Esempio *Trovare i nomi dei clienti che hanno un conto corrente in tutte le filiali di banca di Pisa.*
Abbiamo le seguenti relazioni:

```
Branch(bank_name,branch_name, branch_city)
Account(branch_name, bank_name, account_number, branch_city)
Depositor(account_number, customer_name)
```

Possiamo risolvere l'esercizio attraverso la seguente espressione algebrica:

$$\Pi_{CN,BrN}(depositor \bowtie account) \div \Pi_{BrN}(\sigma_{BC='Pisa'}(branch))$$

Il risultato sono i clienti che hanno un conto in tutte le filiali di Pisa. L'espressione precedente, derivata, corrisponde alla seguente

$$\begin{aligned}\Pi_{CN}(depositor \bowtie \sigma_{BC='Pisa'}(account)) - \Pi_{CN}(\\ ((\Pi_{CN}(depositor \bowtie \sigma_{BC='Pisa}(account)) \bowtie \Pi_{BrN}(\sigma_{BC='Pisa}(branch))\\ -\\ \Pi_{CN,BrN}(depositor \bowtie account)\\ )\end{aligned}$$

- Prendo i clienti che esistono e hanno almeno un conto in una filiale di Pisa.
- Attraverso un prodotto cartesiano (osservo dalle proiezioni che non metto insieme tuple con attributi comuni) genero tutte le coppie possibili tra i clienti che hanno conti a Pisa e le sedi di Pisa. Il risultato intermedio è caratterizzato da coppie veritiere (coloro che hanno effettivamente un conto in quella sede) e coppie false (persone che non hanno un conto in quella sede)
- Da questo prodotto cartesiano sottraggo le coppie veritiere (tutte, incluse quelle non di Pisa)
- La differenza più esterna mi porta ad ottenere coloro che hanno conti solo a Pisa.

- **Persona che ha conti solo a Pisa**: le coppie ottenute attraverso il prodotto cartesiano sono tutte veritiere. Segue che la differenza più interna le rimuoverà. Quindi la differenza più esterna non mi va a rimuovere questa persona.
- **Persona che non ha conti in tutte le filiali di Pisa**: le coppie ottenute attraverso il prodotto cartesiano sono in parte veritiere e in parte false. Attraverso la differenza più interna rimuovo le coppie veritiere. Mi rimangono le coppie false. Se ci sono coppie false significa che l'utente non ha conti in tutte le filiali di Pisa

4 Calcolo relazionale

Con **calcolo relazionale** intendiamo una famiglia di linguaggi dichiarativi che permette di specificare le proprietà del risultato di una certa interrogazione. Ne abbiamo due versioni:

- il calcolo relazionale **applicato ai domini** (che presenta in modo naturale le proprietà dei predicati)
- il calcolo **su tuple con dichiarazioni di range** (versione adottata da SQL)

4.1 Calcolo su domini

$$\{A_1 : x_1, \ldots, A_n : x_n | f\}$$

- $A_1, \ldots, A_n$ sono nomi di attributi, $x_1, \ldots, x_n$ sono nomi di variabili
- La lista delle coppie $A_i : x_i$ è detta *target list* e definisce la struttura del risultato
- f è una formula. Al suo interno possiamo porre:
  - $R(A_1 : x_1, \ldots, A_n : x_n)$, dove $R(A_1 \ldots A_n)$ è uno schema di relazione $x_1 \ldots x_n$ sono variabili. Vera sui valori $x_1, \ldots, x_n$ che formano una tupla della relazione r sullo schema R, nell'istanza di base di dati a cui l'espressione viene applicata.
  - $x\theta y$, $x\theta c$: x, y sono variabili, c è una costante e θ è un operatore di confronto $(=, \neq, \leq, \geq, >, <)$. Vera sui valori x, y, c che soddisfano il confronto con operatore θ.
  - quantificatori nella forma

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

    Cioè: *Esiste almeno una variabile x che soddisfa la formula f*

  Se f_1 e f_2 sono formule allora lo sono anche $\neg f_1$, $\neg f_2$, $f_1 \wedge f_2$, $f_1 \vee f_2$.

4.2 Calcolo su tuple con dichiarazione di range

$$\{x_1.Z_1, \ldots, x_n.Z_n | x_i(R_1), \ldots, x_j(R_m) | f\}$$

- $x_1.Z_1, \ldots, x_n.Z_n$ è la *target list* (con x_i nome di tupla e Z_j insieme di nomi di attributi)
- $x_i(R_1), \ldots, x_j(R_m)$ è la *range list* (campo di variabilità delle variabili)
- f è una formula. Al suo interno possiamo porre:

- formule atomiche del tipo $x_i.Z_i\,\theta\,x_j.Z_j$ o $x_i.Z_i\,\theta\,c$ (dove c è una costante).
- connettivi come nel calcolo su domini
- quantificatori nella seguente forma

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

Cioè: *Esiste nella relazione R almeno una variabile x che soddisfa la formula f*

4.3 Quantificatori esistenziali e universali

All'interno dei predicati possiamo introdurre i cosiddetti *quantificatori*. Abbiamo il quantificatore esistenziale $\exists$ e quello universale $\forall$. I due sono intercambiabili, nel senso che ne basta uno per esprimere qualunque cosa. Le *leggi di de Morgan* valgono, *mutatis mutandis*, anche per i quantificatori:

- $\exists x(f) = \neg(\forall x(\neg(f)))$
- $\forall x(f) = \neg(\exists x(\neg(f)))$

Ricordiamo le leggi di De Morgan, già viste a *Fondamenti di programmazione*

$$\boxed{\neg(f \wedge g) = \neg(f) \vee \neg(g)} \qquad \boxed{\neg(f \vee g) = \neg(f) \wedge \neg(g)}$$

Date due condizioni f e g:

1. La negazione dell'AND si ha quando almeno una delle condizioni è negata
2. La negazione dell'OR si ha quando entrambe le condizioni sono negate

Quando sono necessari? I quantificatori possono essere omessi in certe circostanze, ma in altre sono obbligatori: parliamo di interrogazioni più complesse come, per esempio, la differenza.

4.4 Esempi

Base di dati per gli esempi

`IMPIEGATI(Matricola, Nome, Età, Stipendio)`
`SUPERVISIONE(Capo, Impiegato)`

Trovare gli impiegati che guadagnano più di 40 milioni

- **Su domini**: { Matricola: m, Nome : n, Età: e, Stipendio: s | Impiegati(Matricola: m, Nome: n, Età: e, Stipendio: s) $\wedge$ s $>$ 40 }
- **Su tuple con dichiarazione di range**: {i.* | i(Impiegati) | i.Stipendio $>$ 40}

Trovare nome e matricola degli impiegati che guadagnano più di 40 milioni

- **Su domini**: { Matricola: m, Nome: n | Impiegati(Matricola: m, Nome: n, Età: e, Stipendio: s) $\wedge$ s $>$ 40}
- **Su tuple con dichiarazione di range**: { i.(Matricola, Nome) | i(Impiegati) | i.Stipendio $>$ 40 }

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

- **Primo metodo con calcolo sui domini**: { Matricola: c, Nome: n | Impiegati(Matricola: c, Nome: n, Età: e, Stipendio: s) $\wedge$ $\forall m'(\forall n'(\forall e'(\forall s'(\mathit{Impiegati}$(Matricola: m', Nome: n', Età: e', Stipendio: s'} $\wedge$ Supervisione(Capo:c, Impiegato:m') $\wedge$ s' > 40))))}
- **Secondo metodo con calcolo sui domini**: { Matricola: c, Nome: n | Impiegati(Matricola: c, Nome: n, Età: e, Stipendio: s) $\wedge$ $\neg(\exists m'(\exists n'(\exists e'(\exists s'(\mathit{Impiegati}$(Matricola: m', Nome: n', Età: e', Stipendio: s'} $\wedge$ Supervisione(Capo:c, Impiegato:m') $\wedge$ s' ≤ 40)))))}
- **Terzo metodo con calcolo su tuple**: {i.(Matricola, Nome) | s(Supervisione), i(Impiegati) | i.Matricola = s.Capo $\wedge$ $\neg(\exists i$(Impiegati)(s.Impiegato=i.Matricola $\wedge$ i.Stipendio ≤ 40))))}

4.5 Discussione sul calcolo su domini

Il calcolo su domini è dichiarativo, ma eccessivamente verboso con possibilità di scrivere espressioni sensa senso dipendenti dal dominio e aventi risultati di grandi dimensione. Nell'algebra, invece, tutte le espressioni hanno un senso e sono indipendenti dal dominio.

Indipendenza dal dominio Un'espressione si dice indipendente dal dominio se il suo risultato, su ciascuna istanza di base di dati, non varia al variare del dominio rispetto al quale l'espressione è valutata.

Esempio di espressione dipendente dal dominio Un esempio di espressione dipendente dal dominio è la seguente:

$$\{A_1 : x_1, A_2 : x_2 \mid R(A_1 : x_1, A_2 : x_2) \wedge x_2 = x_2\}$$

$x_2 = x_2$ è una condizione sempre vera: se il dominio include gli interi da 0 a 99 avremo 100 tuple, se invece il dominio va da 0 a 999 avremo 1000 tuple!

4.6 Discussione sul calcolo su tuple

Le variabili rappresentano tuple e si ha minore verbosità. Alcune interrogazioni importanti non possono essere espresse, in particolare le unioni:

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

Ogni variabile nel risultato ha un solo range, mentre vorremmo tuple sia della prima relazione che della seconda. Intersezione e differenza, comunque sia, sono esprimibili. Per questa motivazione SQL prevede esplicitamente un operatore unione, ma non tutte le versioni prevedono intersezione e differenza.

4.7 Calcolo e algebra relazionale

Calcolo e algebra relazionale 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)

Possono espresse buona parte delle interrogazioni, però ci sono cose non esprimibili come la *chiusura transitiva*.

4.8 Chiusura transitiva di una relazione

La chiusura transitiva non può essere definita poichè dovrei fare il JOIN innumerevoli volte per arrivare al risultato, alla tabella definitiva.

Esempio Consideriamo la seguente relazione

`Supervisione(Impiegato, Capo)`

Voglio trovare, per ogni impiegato, tutti i superiori (cioè il capo, il capo del capo, e così via). Nel seguente esempio (teniamo conto che vi è la ridenominazione dell'attributo *Campo*) la cosa pare semplice

Impiegato	Capo
Rossi	Lupi
Neri	Bruni
Lupi	Falchi

Impiegato	Superiore
Rossi	Lupi
Neri	Bruni
Lupi	Falchi
Rossi	Falchi

ma se abbiamo una nuova n-upla come nel secondo esempio risulta necessario stabilire ulteriori legami. Se il capo di Rossi è Lupi e il capo di Lupi è Falchi allora il capo di Rossi è Falchi.

Impiegato	Capo
Rossi	Lupi
Neri	Bruni
Lupi	Falchi
Falchi	Leoni

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

Conclusione Non possiamo calcolare la chiusura transitiva di una relazione qualunque. In algebra relazionale l'operazione si simulerebbe con un numero di JOIN illimitato.

5 Relazioni derivate

Relazioni Presentano un contenuto autonomo

Relazioni derivate Relazioni il cui contenuto è funzione del contenuto di altre relazioni. Quando compiamo un'interrogazione costruiamo dei risultati intermedi senza intaccare la base di dati. Potrei avere il bisogno di recuperare un risultato (anche più di una volta) assegnandogli un nome. Ciò consiste in una *vista*. Essa può essere materializzata (salvata e usata più volte) o virtuale (usata una volta sola).

- **Viste materializzate**: La memorizzazione risulta vantaggiosa perchè non hai da fare il calcolo tutte le volte (si hanno delle tabelle in carne ed ossa, ripensare a quanto visto con Pistolesi). I risultati sono immediatamente disponibili, ma potrebbero essere ridondanti e appesantire. Le viste materializzate non sono supportate da tutti i DBMS
- **Viste virtuali**: La vista virtuale è sempre ricalcolata ed è supportata da tutti i DBMS (nel db salviamo non il result set, ma lo snippet - il testo dell'interrogazione).

Colleghiamo a quanto fatto fino ad ora Ottenere una vista significare dare un nome a un espressione. Esempio:

$$Supervisione = \pi_{Impiegato,Capo}(Afferenza \bowtie Direzione)$$

Posso adottare questa vista anche in altre espressioni:

$$\sigma_{Capo='Leoni'}(Supervisioni)$$

viene eseguita come

$$\sigma_{Capo='Leoni'}(\pi_{Impiegato,Capo}(Afferenza \bowtie Direzione))$$

Vantaggi per il programmatore Possiamo semplificare la scrittura di interrogazioni seguendo il metodo del *divide et impera* (divisione di un problema in sottoproblemi)