
Basi di Dati
Modulo Tecnologie

Ottimizzazione di interrogazioni (2/2)

Dr. Sara Migliorini

Algoritmi per il join

- Il join è l'operazione più gravosa per un DBMS.
- Le implementazioni più diffuse si riconducono ai seguenti operatori fisici:
 - Nested Loop Join
 - Merge Scan Join
 - Hash-based Join
- Si noti che, benché logicamente il join sia commutativo, dal punto di vista fisico vi è una chiara distinzione, che influenza anche le prestazioni, tra operando **sinistro** (o “**esterno**”, “**outer**”) e operando **destro** (o “**interno**”, “**inner**”).
- Per semplicità nel seguito parliamo di **relazione esterna** e **relazione interna** per riferirci agli input del join, ma va ricordato che in generale l'input può derivare dall'applicazione di altri operatori.

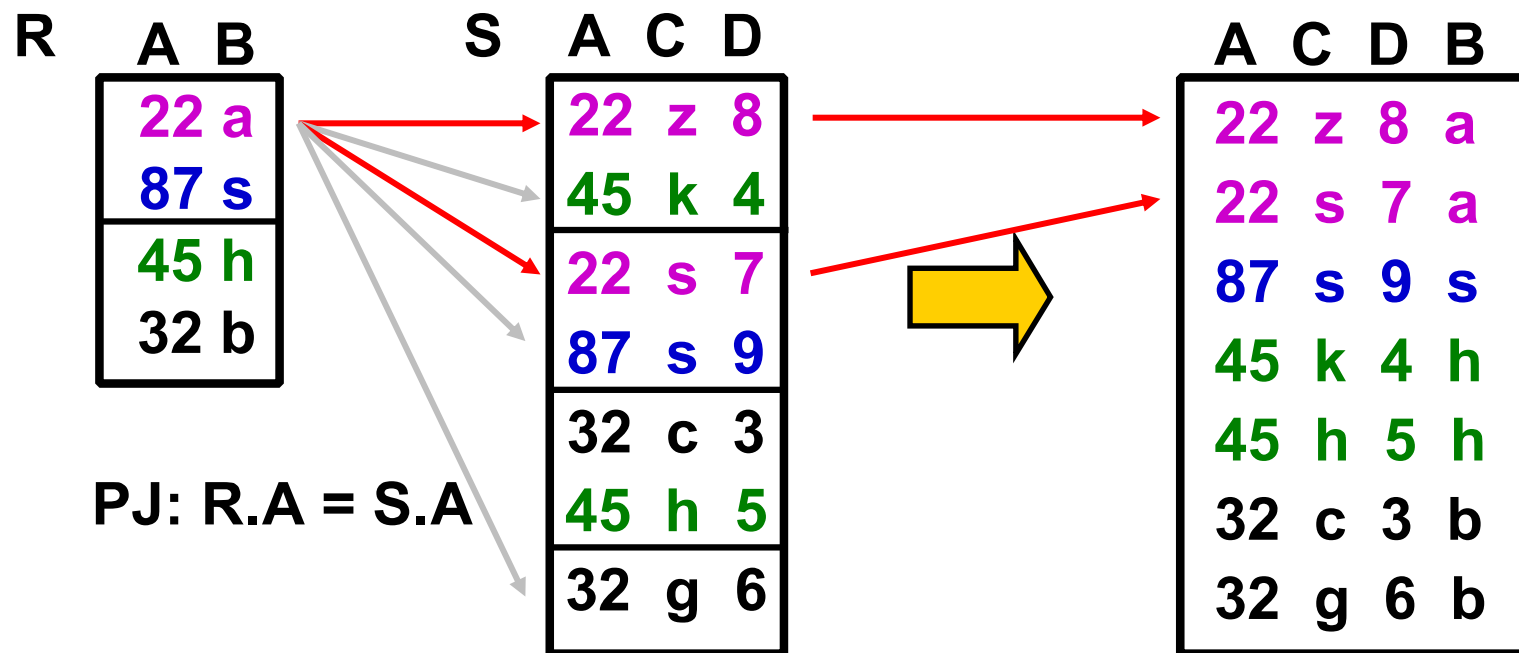
Nested-Loop JOIN

- Date 2 relazioni in input R e S tra cui sono definiti i predicati di join PJ , e supponendo che R sia la relazione esterna, l'algoritmo opera come segue:

```
per ogni tupla  $t_R$  in  $R$  {  
    per ogni tupla  $t_S$  in  $S$  {  
        se la coppia  $(t_R, t_S)$  soddisfa  $PJ$   
            allora aggiungi  $(t_R, t_S)$  al risultato  
    }  
}
```

Nested-Loop JOIN

Esempio

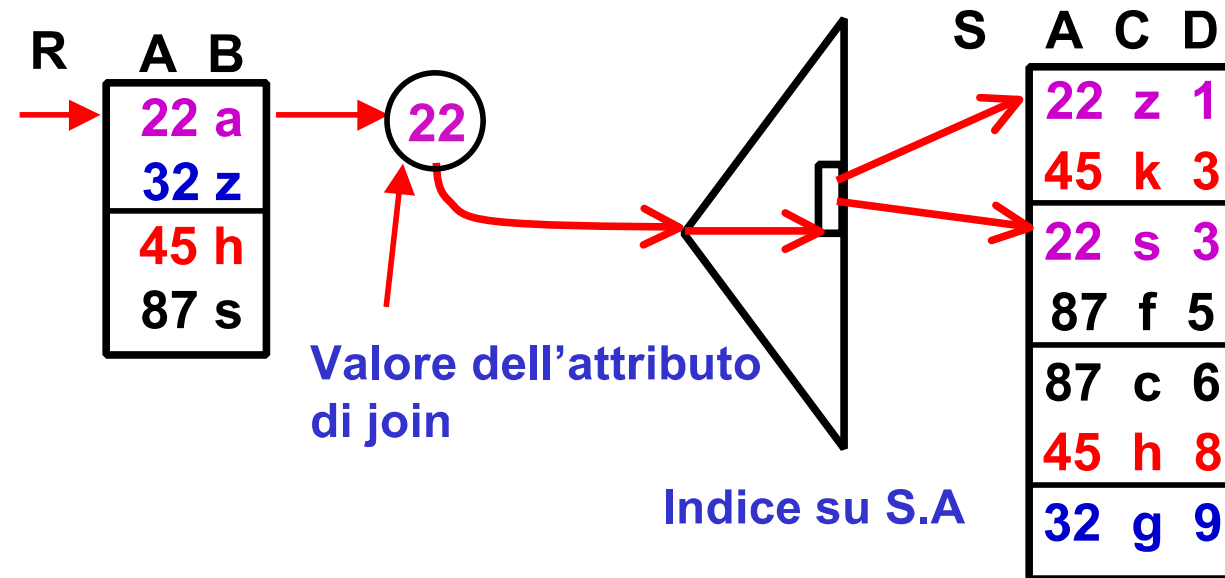


Nested-Loop JOIN: costo

- Il costo di esecuzione dipende dallo spazio a disposizione nei buffer.
- Nel caso base in cui vi sia 1 buffer per R e 1 buffer per S:
 - bisogna leggere 1 volta R e
 - $NR(R)$ volte S, ovvero tante volte quante sono le tuple della relazione esterna,
 - per un totale di $NP(R) + NR(R) * NP(S)$ accessi a memoria secondaria
- Se è possibile allocare $NP(S)$ buffer per la relazione interna il costo si riduce a $NP(R) + NP(S)$
- Si ricordi che:
 - $NR(R)$ = numero tuple (row) di R
 - $NP(R)$ = numero di pagine di R

Nested-Loop JOIN con indice

- Data una tupla della relazione esterna R, la scansione completa della relazione interna S può essere sostituita da una scansione basata su un indice costruito sugli attributi di join di S (nell'esempio A), secondo il seguente schema:



Nested-Loop JOIN con indice B+-tree: costo

- Nel caso base in cui vi sia 1 buffer per R e 1 buffer per S, bisogna leggere 1 volta R e accedere NR(R) volte a S, ovvero tante volte quante sono le tuple della relazione esterna, per un totale di:

SELETTIVITÀ di A



$$NP(R) + NR(R) * (ProfIndice + NR(S)/VAL(A, S))$$

accessi a memoria secondaria

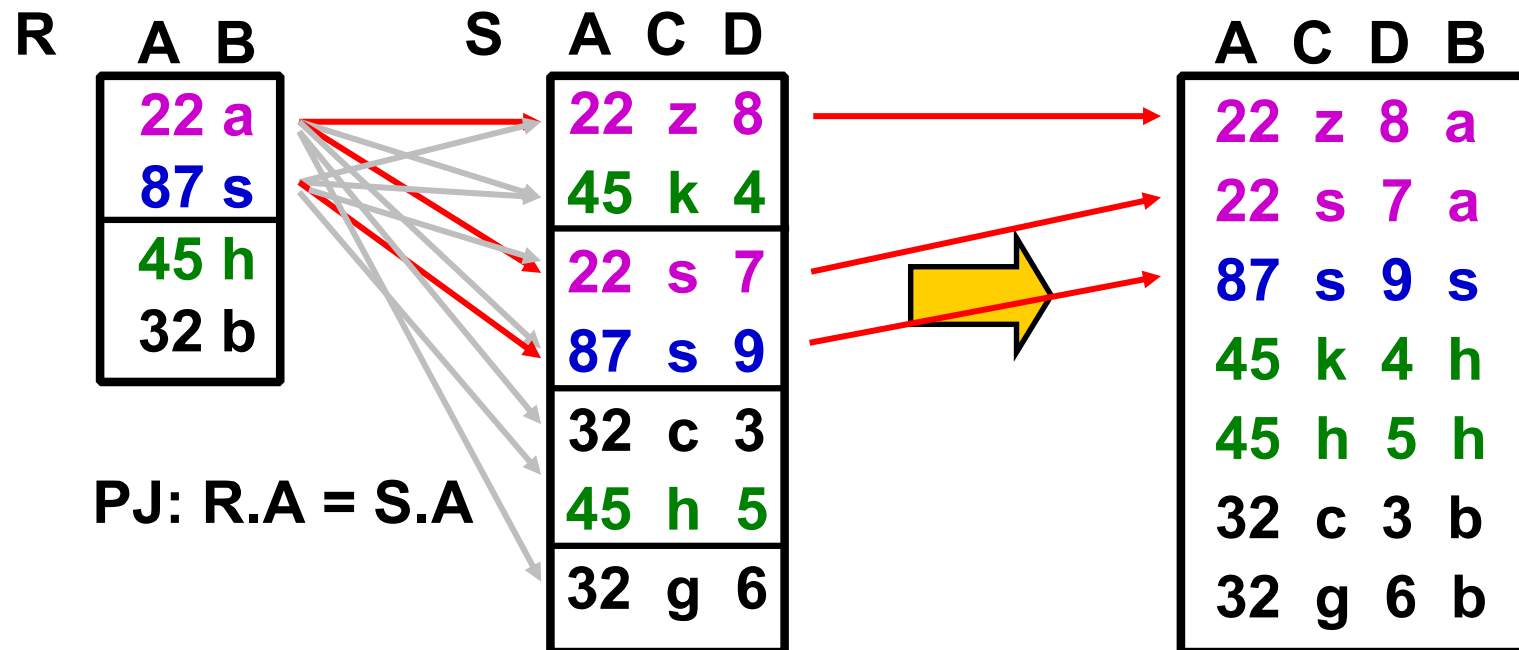
dove:

VAL(A,S) rappresenta il numero di valori distinti dell'attributo A che compaiono nella relazione S.

Block Nested-Loop JOIN

Esempio

Variante che bufferizza le righe lette della tabella esterna per ridurre il numero di volte che deve essere letta nel ciclo interno.



Block Nested-Loop JOIN: costo

- Il costo di esecuzione dipende dallo spazio a disposizione nei buffer.
- Nel caso base in cui vi sia 1 buffer per R e 1 buffer per S:
 - bisogna leggere 1 volta R e
 - $NP(R)$ volte S, ovvero tante volte quanti sono i blocchi della relazione esterna,
 - per un totale di $NP(R) + NP(R) * NP(S)$ accessi a memoria secondaria
- Se è possibile allocare $NP(S)$ buffer per la relazione interna il costo si riduce a $NP(R) + NP(S)$

Merge-Scan JOIN

- Il Merge-Scan Join è applicabile quando entrambi gli insiemi di tuple in input sono ordinati sugli attributi di join e il join è un equi-join.
- Ciò accade se per entrambe le relazioni di input **R** e **S** vale una almeno delle seguenti condizioni:
 - La relazione è fisicamente ordinata sugli attributi di join (file sequenziale ordinato come struttura fisica).
 - Esiste un indice sugli attributi di join della tabella che consente una scansione ordinata delle tuple.

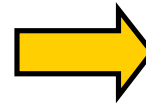
Merge-Scan JOIN

Esempio

R	A	B
	22	a
	32	s
	45	h
	87	b

PJ: R.A = S.A

S	A	C	D
	22	z	8
	22	s	7
	35	h	4
	45	s	9
	45	c	3
	87	h	5
	87	g	6



R esterna

A	C	D	B
22	z	8	a
22	s	7	a
45	s	9	h
45	c	3	h
87	h	5	b
87	g	6	b

Merge-Scan JOIN: costo

- La logica dell'algoritmo sfrutta il fatto che **entrambi gli input sono ordinati** per evitare di fare inutili confronti, il che fa sì che il numero di letture totali sia dell'ordine di:

$$NP(R) + NP(S)$$

se si accede sequenzialmente alle due relazioni ordinate fisicamente.

- Con **indici B+-tree** su entrambe le relazioni il costo può arrivare al massimo a (indici già nel buffer):

$$NR(R) + NR(S)$$

- Con una relazione ordinata e un solo indice al massimo a:

$$NP(R) + NR(S) \text{ (o viceversa)}$$

Hash-based JOIN

- L'algoritmo di Hash Join, applicabile solo in caso di **equi-join**, non richiede né la presenza di indici né input ordinati, e risulta particolarmente vantaggioso in caso di relazioni molto grandi. L'idea su cui si basa l'Hash Join è semplice:
- Si suppone di avere a disposizione una funzione hash h , che viene applicata agli attributi di join delle due relazioni (ad es. $R.J$ e $S.J$).
 - Se t_R e t_S sono 2 tuple di R e S , allora è possibile che sia $t_{R.J} = t_{S.J}$ solo se $h(t_{R.J}) = h(t_{S.J})$
 - Se, viceversa, $h(t_{R.J}) \neq h(t_{S.J})$, allora sicuramente è $t_{R.J} \neq t_{S.J}$

Hash-based JOIN

- A partire da questa idea si hanno diverse implementazioni, che hanno in comune il fatto che **R** e **S** vengono partizionate sulla base dei valori della funzione di hash h , e che la ricerca di matching tuples avviene solo tra partizioni relative allo stesso valore di h .

Hash-based JOIN: costo

- Il costo può essere così valutato:

- A - costruzione hash map:

$$NP(R) + NP(S)$$

- B - accesso alle «matching tuples» che nel caso pessimo può costare (block nested loop join):

$$NP(R) * NP(S)$$

- ma dipende anche fortemente dal numero di buffer a disposizione e dalla distribuzione dei valori degli attributi di join (il caso uniforme è quello migliore).

Scelta finale del piano di esecuzione

Data una espressione ottimizzata in algebra:

- Si generano tutti i possibili piani di esecuzione (alberi) alternativi (o un sottoinsieme di questi) ottenuti considerando le seguenti dimensioni:
 - Operatori alternativi applicabili per l'accesso ai dati: ad esempio, scan sequenziale o via indice,
 - Operatori alternativi applicabili nei nodi: ad esempio, nested-loop join or hash-based join
 - L'ordine delle operazioni da compiere (associatività)
- Si valuta con formule approssimate il costo di ogni alternativa in termini di accessi a memoria secondaria richiesti (stima).
- Si sceglie l'albero con costo approssimato minimo.

Scelta finale del piano di esecuzione

- Nella valutazione delle stime di costo si tiene conto del profilo delle relazioni, solitamente memorizzato nel Data Dictionary e contenente per ogni relazione T:
 - Stima della cardinalità (#tuple): $CARD(T)$
 - Stima della dimensione di una tupla: $SIZE(T)$
 - Stima della dimensione di ciascun attributo A: $SIZE(A,T)$
 - Stima del numero di valori distinti per ogni attributo A: $VAL(A,T)$
 - Stima del valore massimo e minimo per ogni attributo A: $MAX(A,T)$ e $MIN(A,T)$