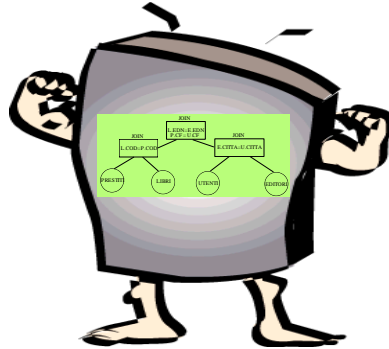


Metodi per l'esecuzione di join



Dario Maio

<http://bias.csr.unibo.it/maio/>

Metodi di join



1

Algoritmi di join: generalità

- Nell'implementazione più semplice il join di due relazioni r e s prevede il confronto di ogni tupla di r con ogni tupla di s , con complessità $O(NT_r \times NT_s)$.
- La molteplicità di metodi di join in letteratura testimonia lo sforzo compiuto per ridurre l'elevato numero di operazioni di I/O che si genera da questa semplice strategia. **Il problema dell'esecuzione di un join è stato affrontato da diversi punti di vista, tra cui:**
 - ✦ **access path selection:** come viene influenzata l'esecuzione di un join dal cammino di accesso alle relazioni?
 - ✦ **optimal nesting:** qual è l'ordine migliore di esecuzione di n join?
 - ✦ **clustering:** in quale modo l'ordinamento fisico dei dati incide sui costi del join?
 - ✦ **buffer:** come devono essere allocate le pagine di buffer per favorire l'esecuzione di un join?
 - ✦ **hardware support:** qual è un buon supporto hardware per i join?
 - ✦ **parallel processing:** come si sfrutta la presenza di più processori e/o dischi?
 - ✦ **physical database design:** come si determinano gli indici utili all'esecuzione di un join?

Metodi di join



2

Two-way e multi-way join

- Una prima distinzione riguardante le interrogazioni in cui sono presenti predicati di join è tra:
 - two-way join** : coinvolge due relazioni $r_1 \bowtie_F r_2 = \sigma_F(r_1 \times r_2)$
 - multi-way join** : coinvolge più di due relazioni.
- Un multi-way join tra n relazioni può sempre risolversi tramite l'esecuzione di n-1 two-way join, indipendenti l'uno dall'altro.
- ESEMPIO:

In linea di principio è possibile eseguire un join alla volta, producendo i risultati parziali in **relazioni temporanee**, come rappresentato dal seguente programma in algebra relazionale in cui, per semplicità, non si considerano le proiezioni:

```

Temp1 ← Libri ⋈ PRESTITI /* join su COD (codice libro) */
Temp2 ← Editori ⋈ Temp1 /* join su EDC (codice editore) */
Result ← Utenti ⋈ Temp2 /* join su Città e CF (codice fiscale) */

```

Metodi di join

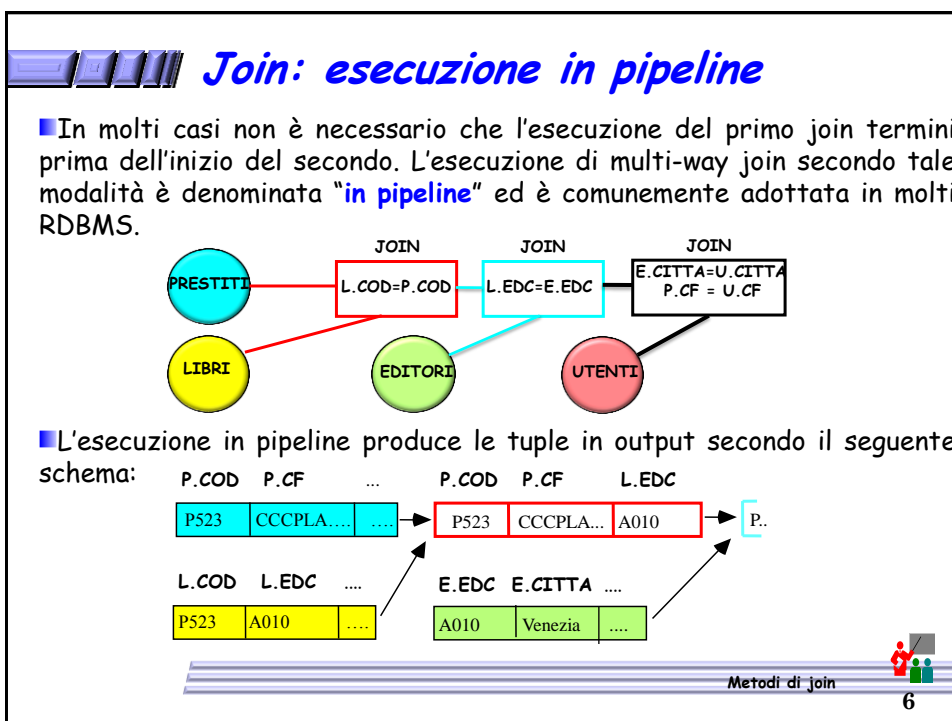
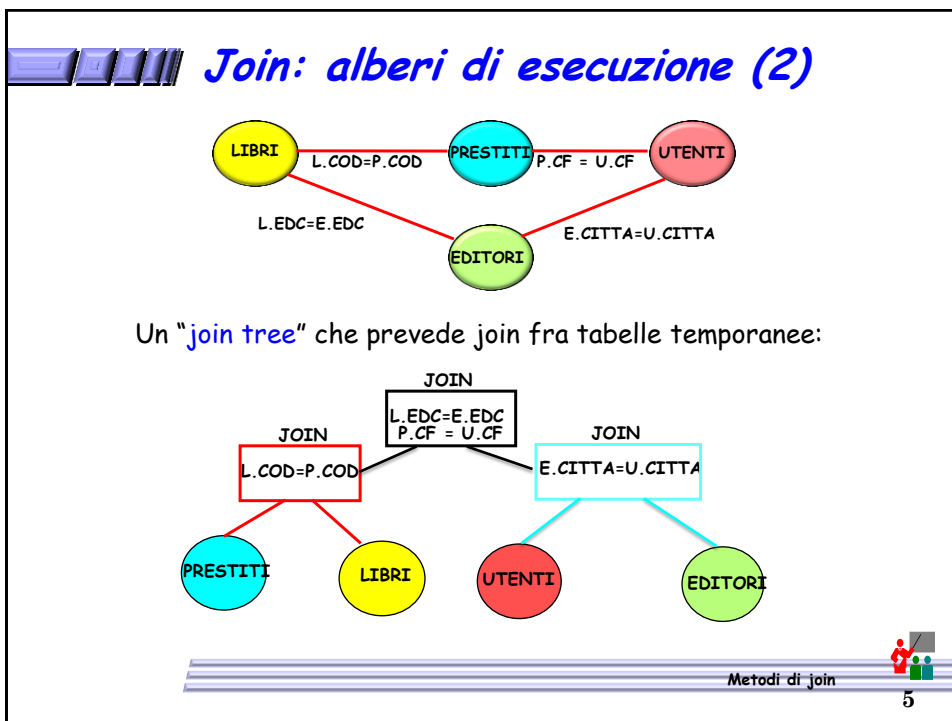
3

Join: alberi di esecuzione (1)

Un "join tree" che prevede a ogni passo l'uso di almeno una table del DB:

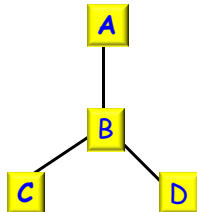
Metodi di join

4



Piani di esecuzione

- ✚ Esempio di schema natural join **a stella**: possibili sequenze di esecuzione $4! = 24$, ognuna composta da 3 join.
- ✚ Ipotezzando di avere a disposizione come algoritmo solo il **nested loops**, si può limitare lo spazio delle soluzioni adottando alcuni euristici, ad esempio:



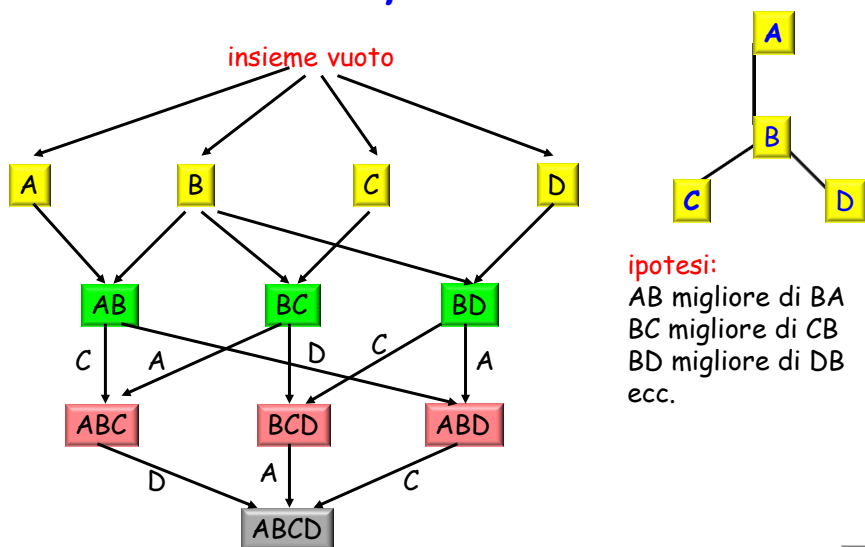
- ✚ 1: si escludono i piani di accesso che prevedono prodotti cartesiani;
- ✚ 2: si sceglie il modo migliore per ottenere una relazione intermedia: ad esempio se AB è migliore di BA non si considerano più BAD e BADC, ma solo ABD e ABDC.

Metodi di join



7

Limitare lo spazio delle soluzioni



Metodi di join



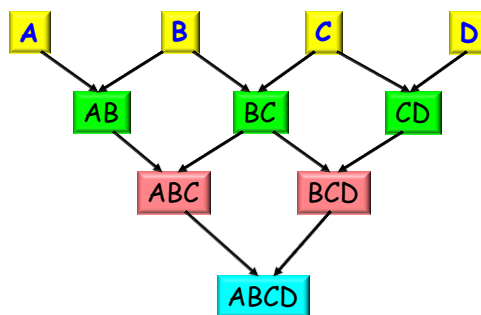
8



Esempio schema a catena



Nell'ipotesi di disporre solo di nested loops e applicando le regole 1 e 2:



Metodi di join



9



Nested loops join (1)

- È il metodo di join più semplice e deriva direttamente dalla definizione dell'operazione. Una delle due relazioni coinvolte è designata come **esterna** e l'altra come **interna**.

```

SELECT <select-list>
FROM   R, S
WHERE  FR /* predicati locali su R */
AND    FS /* predicati locali su S */
AND    FJ /* predicato/i di join */
  
```

- L'algoritmo, per ogni tupla t_r di r che verifica F_R , accede a s ricercando tutte le tuple t_s che soddisfano F_S e che possono concatenarsi con t_r .
- Definiamo **SCAN_R** e **SCAN_S** rispettivamente i cammini d'accesso su r e su s rispettivamente. **Si supponga r esterna e s interna.**

Metodi di join



10


Nested loops join (2)

```

/* NESTED-LOOPS JOIN ALGORITHM */

OPEN scan ... /* SCAN_R */
DO while more-to-come on SCAN_R;
  NEXT using SCAN_R, ...;
  IF found THEN
    IF  $F_R$  THEN
      DO;
        OPEN scan ... /* SCAN_S */
        DO while more-to-come on SCAN_S;
          NEXT using SCAN_S, ...;
          IF found THEN
            IF ( $F_S$  AND  $F_J$ ) THEN
              <join & output selected fields>;
            END;
          CLOSE scan SCAN_S;
        END;
      END;
    END;
  CLOSE scan SCAN_R;

```



 Metodi di join

 11

Nested loops: costi di esecuzione

✚ Il costo di esecuzione si esprime, in generale, come:

$C_a(r) + ET_r \times C_a(s)$

dove

$C_a(r)$ è il costo di accesso a r

$ET_r = f_{FR} \times NT_r$ è il numero atteso di tuple residue di r

$C_a(s)$ è il costo di accesso a s , per ogni tuple residua di r .


✚ **Casi notevoli**

1. In assenza di predicati locali sulla relazione esterna:

$C_a(r) + NT_r \times C_a(s)$
2. Facendo uso di scan sequenziali:

$NP_r + ET_r \times NP_s$
3. Se valgono sia 1 sia 2 il costo diventa:

$NP_r + NT_r \times NP_s$



 Metodi di join

 12

Scelta della relazione esterna

- Se valgono sia 1 sia 2 il costo diventa:

caso r esterna $NP_r + NT_r \times NP_s = NP_r + NP_r \times TP_r \times NP_s$

caso s esterna $NP_s + NT_s \times NP_r = NP_s + NP_s \times TP_s \times NP_r$

essendo rispettivamente TP_r e TP_s il numero di tuple per pagina di r e di s.

- Si deduce che, per relazioni "grandi", conviene scegliere come esterna quella con il minor numero di tuple per pagina in quanto ciò minimizza la componente del costo di peso maggiore.
- Nel caso particolare in cui si abbia $TP_r = TP_s$, si sceglie come relazione esterna quella con il minor numero di pagine.

Metodi di join



13

Nested loops: cammini di accesso

- Il cammino di accesso alla relazione esterna determina l'ordine (primario) con cui vengono generate le tuple del risultato.

| PARTI | P# | DESCR | COLORE | PESO | CITTÀ |
|-------|------|----------|---------|------|-------|
| | 1234 | vite | oro | 0.05 | NA |
| | 4611 | bullone | nero | 0.09 | NA |
| | 2527 | chiodo | argento | 0.04 | PG |
| | 1093 | chiave | argento | 0.20 | VE |
| | 1101 | tassello | oro | 0.11 | PG |

1

2

3

4

5

| FORNITORI | F# | NOME | REGIONE | CITTÀ |
|-----------|-----|-------|----------|-------|
| | 192 | ROSSI | VENETO | VE |
| | 215 | VERDI | CAMPANIA | NA |
| | 296 | ROSSI | CAMPANIA | NA |
| | 142 | NERI | UMBRIA | PG |

a

b

c

d

PARTI ▷◁ FORNITORI

- Se la relazione esterna è PARTI e si adotta uno **scan sequenziale**, le tuple vengono generate secondo l'ordine:

(1b),(1c),(2b),(2c),(3d),(4a),(5d)

- Viceversa, se si accede con un indice su P# l'ordine risultante è:

(4a),(5d),(1b),(1c),(3d),(2b),(2c)

Metodi di join



14

Cammini di accesso interessanti

- Nell'ipotesi di r esterna e s interna:

Per la relazione esterna r :

- ✚ scansione sequenziale
- ✚ accesso via indice (**indici**) su un attributo (**su attributi**) che compare (**compaiono**) nei predicati locali F_R .

Per la relazione interna s :

- ✚ scansione sequenziale
- ✚ accesso via indice (**indici**) su un attributo (**su attributi**) che compare (**compaiono**) nei predicati locali F_S .
- ✚ accesso via indice (**indici**) su un attributo (**su attributi**) di join .

Metodi di join



15

Nested loops: esempio (1)

- Sono date le due relazioni:

Paper(Pcode, Title, First_Author, Score, Conference, Session)

Author(Pcode, Authorcode)

e la query:

```
SELECT Authorcode, Title
FROM Paper P, Author A
WHERE Score = 'High'
AND Conference = 'VLDB'
AND Session = 'Plenary'
AND P.Pcode = A.Pcode
```

Le informazioni statistiche rilevanti sono:

NTP = 4000 NTA = 8000
NPP = 800 NPA = 400

I fattori di selettività dei predicati sono:


| | |
|---------------------|--------|
| Score = 'High' | 1/5 |
| Conference = 'VLDB' | 1/20 |
| Session = 'Plenary' | 1/10 |
| A.Pcode = <valore> | 1/4000 |

Si assuma che Paper sia la relazione esterna e che esista un indice unclustered sull'attributo Session, di altezza 2, con 20 foglie.

Metodi di join



16



Nested loops: esempio (2)

- Accesso alla relazione esterna (Paper):**
 Facendo uso dell'indice IX(P.Session) si ha:

$$C_a(IX(P.Session) \text{ uncl}) = 1 + \lceil f_{p(P.Session)} \times NL_{(P.Session)} \rceil + \Phi(f_{p(P.Session)} \times NT_p, NP_p) = 1 + 2 + 315 = 318$$
- Numero di tuple residue della relazione esterna**
 Il numero di tuple residue di Paper è pari a:

$$ET_p = f_p \times NT_p = (0.2 \times 0.05 \times 0.1) \times 4000 = 4$$
- Accesso alla relazione interna (Author)**

$$C_a(Seq. Author) = NP_A = 400$$

 Il costo complessivo è pertanto $318 + 4 \times 400 = 1918$
- Se si disponesse di un indice unclustered su A.Pcode, di altezza 3, con 100 foglie, si avrebbe:**

$$C_a(IX(A.Pcode) \text{ uncl}) = 2 + \lceil f_{p(A.Pcode)} \times NL_{(A.Pcode)} \rceil + \Phi(f_{p(A.Pcode)} \times NT_A, NP_A) = 2 + 1 + 2 = 5$$

 per un costo complessivo pari a $318 + 4 \times 5 = 338$

Metodi di join

17



Nested loops: varianti (1)

- Zig-Zag:** questa variante (Kim 1980) prevede che la relazione interna sia alternativamente letta dall'inizio alla fine e dalla fine all'inizio.
 Ciò consente di risparmiare a ogni passo (tranne il primo) la lettura di una pagina della relazione interna, in quanto già presente in memoria, per un costo complessivo pari a:

$$C_a(r) + C_a(s) + (ET_r - 1) \times (C_a(s) - 1)$$
- Uso di relazioni temporanee:** una variante utilizzata originariamente in INGRES effettua, prima di verificare la condizione di join, **restrizione** e **proiezione** delle due relazioni sulla base dei predicati locali e dei campi richiesti nel risultato, facendo uso di due relazioni temporanee r' e s'.
 La condizione di join è verificata accedendo sequenzialmente a entrambe le relazioni temporanee.

Metodi di join

18

Nested loops: varianti (2)

- La scelta della relazione (temporanea) **esterna** si può basare sull'analisi dei costi effettivi, dati da:

$$r' \text{ esterna: } C_a(s) + C_c(s') + C_a(r) + C_c(r') + NP_{r'} + NT_{r'} \times NP_{s'}$$

$$s' \text{ esterna: } C_a(s) + C_c(s') + C_a(r) + C_c(r') + NP_{s'} + NT_{s'} \times NP_{r'}$$

dove $C_c(r')$ e $C_c(s')$ indicano, rispettivamente, il costo di costruzione di r' e di s' .

- Questa variante preclude l'uso di un indice di join sulla relazione interna.** Tuttavia, se il numero di pagine della temporanea interna è abbastanza piccolo, il costo globale risulta spesso inferiore a quello del nested-loops classico.
- Nested-loops parallelo:** il nested-loops è un algoritmo di join che ben si presta a essere parallelizzato, ad esempio avendo più processori che elaborano le tuple della relazione esterna e confrontano (in parallelo) quelle residue con le tuple della relazione interna.

Metodi di join



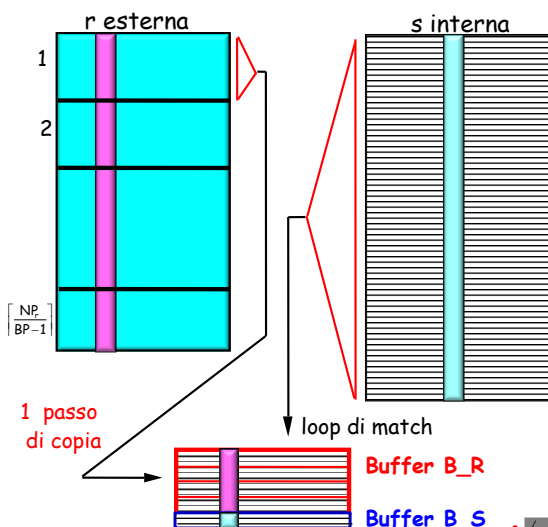
19

Nested-block join

- Nel caso di predicati poco selettivi e/o di scan sequenziale sulla relazione interna il **nested-loops può dar luogo a costi proibitivi**.

- L'algoritmo di **nested-block join** riduce il numero di scan sulla relazione interna, e quindi i costi di I/O, facendo uso di un **buffer di BP pagine**, di cui **BP-1** sono riservate per la lettura della relazione esterna e 1 per la relazione interna.

- Per ogni "gruppo" di BP-1 pagine di R (esterna) caricate nel buffer si apre uno scan su S (interna).



Metodi di join



20

Nested-block: algoritmo

- L'algoritmo può essere descritto concisamente assumendo che B_R sia il buffer per la relazione esterna, B_S quello di 1 pagina per la relazione interna e

<match B_R and B_S tuples>

indichi la procedura che esegue il join delle tuple nel buffer, verificando i predicati locali e quello/i di join.

```

/* NESTED-BLOCK JOIN ALGORITHM */
OPEN scan ... /* SCAN_R */
DO while more-to-come on SCAN_R;
  FILL_BUFFER B_R;
  OPEN scan ... /* SCAN_S */
  DO while more-to-come on SCAN_S;
    FILL_BUFFER B_S;
    <match B_R and B_S tuples>
    <output selected fields>;
    FREE_BUFFER B_S;
  END;
  CLOSE scan SCAN_S;
  FREE_BUFFER B_R;
END;
CLOSE scan SCAN_R;

```

Metodi di join



21

Nested-block: costi

- ✚ L'ordine delle tuple nel risultato è diverso rispetto a quello del **nested-loops**, ovvero le tuple non sono più prodotte rispettando l'ordinamento logico stabilito dal cammino d'accesso alla relazione esterna.
- ✚ Il costo dell'algoritmo, nel caso di scan sequenziali e assenza di predicati locali, è pari a:

$$NP_r + \lceil NP_r / (BP-1) \rceil \times NP_s$$

- ✚ Facendo uso di indici vale quanto visto per il nested-loops, ovvero NP_r e NP_s diventano rispettivamente $C_a(r)$ e $C_a(s)$; ad esempio:

$$C_a(r) + \lceil EP_{p(R,A_i)} / (BP-1) \rceil \times C_a(s)$$

dove $EP_{p(R,A_i)}$ indica il numero di pagine dati di r reperite dallo scan sull'indice $IX(R,A_i)$ usando il predicato $p(R,A_i)$.

- ✚ Per stimare $C_a(s)$ nel caso di accesso con indice su attributo di join, si considera l'estrazione dei valori di join di r dal buffer e l'esecuzione di una query di set sulla relazione interna s .

Metodi di join



22

Sort-Merge join

- Il sort-merge join è eseguito in due passi. Nel primo passo (**sort**) le relazioni vengono ordinate sugli attributi di join. Durante il secondo passo (**merge**) entrambe le relazioni vengono scandite nell'ordine degli attributi di join e le tuple che soddisfano la condizione di join sono fuse per costituire il risultato.
- La forma dell'algoritmo dipende dal tipo di join (**operatore 9**) e dal fatto che gli attributi di join siano o meno a valori unici.
- L'algoritmo nel caso di equi-join $R.J = S.J$, con $R.J$ chiave primaria e $S.J$ chiave importata (foreign key), e in assenza di predicati locali.

```
/* SORT-MERGE JOIN ALGORITHM */  
/* Sort phase */  
  sort relation r on R.J;  
  sort relation s on S.J;  
  
/* Merge phase */  
  getnext( $t_r$ );  
  getnext( $t_s$ );  
  while not EOF(r) and not EOF(s)  
  do  
    { if  $t_r[J] = t_s[J]$   
      then { <join & output selected fields>;  
            getnext( $t_s$ ); }  
      else if  $t_r[J] < t_s[J]$   
        then getnext( $t_r$ )  
        else getnext( $t_s$ ) }
```

Metodi di join



23

Sort-Merge join: costi

- Il sort-merge join nel caso considerato comporta un costo pari a:

$$C(\text{Sort } r) + C(\text{Sort } s) + NP_r + NP_s$$

assumendo di leggere tutte le pagine delle relazioni ordinate.

- Sempre riferendosi a equi-join tra primary e foreign key, il costo della fase di merge può essere minore se $\max(R.J) > \max(S.J)$, in quanto non è necessario leggere tutte le tuple di r .
- In presenza di predicati locali è possibile effettuare preventivamente la restrizione e la proiezione delle due relazioni, memorizzando i risultati in due temporanee che poi vengono ordinate sull'attributo di join.

Metodi di join

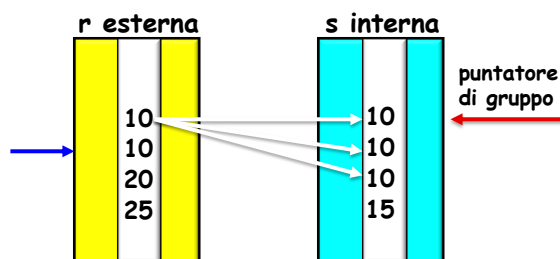


24



Sort-Merge join con place holder

- In presenza di valori duplicati per entrambi gli attributi di join (**associazione del tipo m:n**), è necessario modificare l'algoritmo per poter effettuare backtracking.



- Le tuple delle due relazioni vengono scandite parallelamente per valori crescenti dell'attributo di join. Un **puntatore di gruppo** ("place holder") individua, a ogni passo, la prima tupla della relazione interna che verifica il predicato di join con la corrente tupla esterna.

Metodi di join



25



SM-join con place holder: costi

- Rispetto al caso precedente, in generale si ha un incremento dei costi di accesso alla relazione interna. Sotto le ipotesi che alla relazione interna s sia riservata una sola pagina di buffer e che $\lceil NP_s / NK_{s,j} \rceil > 1$ (altrimenti non si leggerebbe più di una volta la stessa pagina di s), il costo di accesso a s è valutabile nel caso peggiore come:

$$NT_r \times \lceil NP_s / NK_{s,j} \rceil$$

- Il vantaggio del sort-merge rispetto al nested-loops è evidente se le relazioni sono già ordinate. Se la selettività del join, definita come:

$$f_{\text{F}_j} = \frac{|R \bowtie_{\text{F}_j} S|}{R \times S} = \frac{\text{cardinalità del join}}{\text{cardinalità del prodotto cartesiano}}$$

è bassa, il numero di tuple confrontate è considerevolmente inferiore. Infatti non è necessario riscandire le tuple della relazione interna che precedono il puntatore di gruppo.

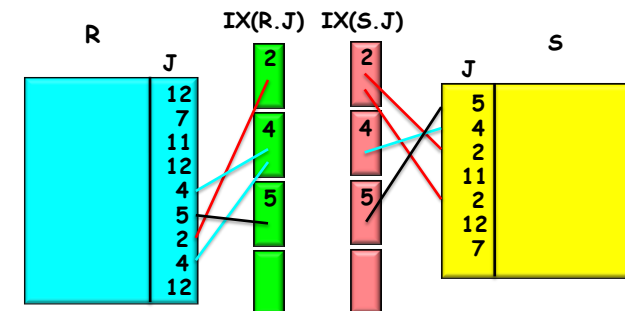
Metodi di join



26

Merging scans join (1)

- È una generalizzazione dell'algoritmo di sort-merge che richiede unicamente di poter accedere alle tuple secondo l'ordine stabilito dai valori degli attributi di join.
- Si consideri un equi-join R.J e S.J. Se esistono gli indici IX(R.J) e IX(S.J), non è necessario eseguire il sort, ma si possono usare gli indici per accedere alle tuple secondo l'ordine desiderato.



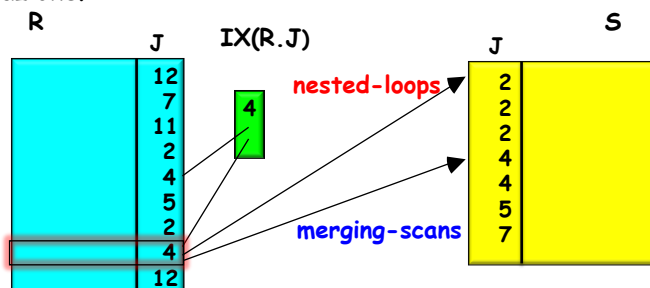
Metodi di join



27

Merging scans join (2)

- Se entrambi i cammini di accesso sono via indici unclustered, il costo può risultare comunque elevato.
- L'algoritmo di merging-scans si differenzia da quello di nested-loops anche quando entrambi utilizzano IX(S.J) per accedere a S, in quanto nested-loops riapre ogni volta lo scan, mentre merging-scans fa uso di place holder sulle foglie ed esegue un singolo scan. Se l'accesso alla relazione interna S è sequenziale, e S è clustered sull'attributo di join, la situazione peggiora per il nested-loops, che esegue ogni volta uno scan completo della relazione.



Metodi di join



28

Simple hash join (1)

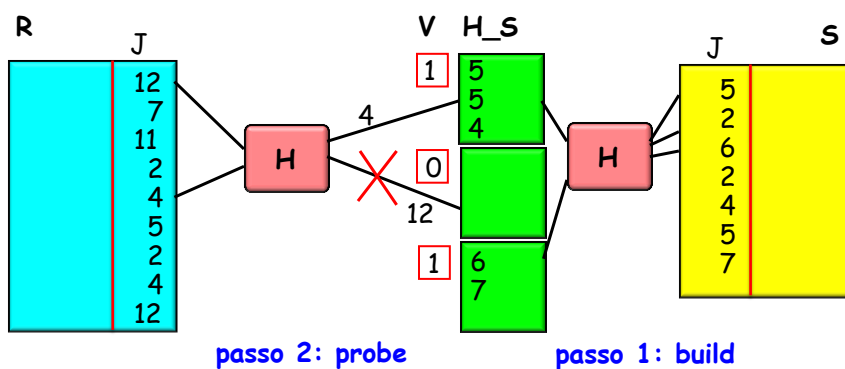
- Il successo del sort-merge (e del merging-scans) è principalmente dovuto al fatto che riduce il numero di confronti fra le tuple delle relazioni coinvolte.
- I metodi che usano tecniche hash mirano a ottenere lo stesso effetto evitando l'ordinamento delle relazioni e senza richiedere l'uso di indici sugli attributi di join. La famiglia dei join basati su tecniche hash, utilizzabili nel caso di equi-join (a meno di non fare uso di funzioni hash che preservano l'ordine), presenta in generale prestazioni migliori rispetto ad altri metodi.
- L'algoritmo di **simple-hash join** è tra i più semplici della famiglia, e consta di due passi:
 - passo 1 (build)**: si applica una funzione hash H ai valori degli attributi di join di una delle due relazioni (s) generando una hash table H_s . Si usa un vettore di bit V per tener traccia di quali bucket di H_s sono vuoti o meno. La relazione s è detta interna.
 - passo 2 (probe)**: si scandisce la seconda relazione (r), detta esterna, e, per ogni tupla, si accede a H_s tramite la stessa funzione hash H , solo se il bucket colpito non è vuoto.

Metodi di join



29

Simple hash join (2)



Metodi di join



30

Simple hash join (3)

- L'algoritmo di simple-hash join, nel caso $R.J = S.J$ e senza predicati locali, si esprime come segue:

```
/* SIMPLE-HASH JOIN ALGORITHM */
/* Phase 1 */
FOR EACH  $t_s$ 
DO {  $i = H(t_s[J])$ ;
     $V[i] = 1$ ;
    INSERT( $t_s, H\_s[i]$ ) };
/* Phase 2 */
FOR EACH  $t_r$ 
DO {  $i = H(t_r[J])$ ;
    IF  $V[i] = 1$ 
    THEN { READ( $H\_s[i]$ );
        FOR EACH  $t_s$  IN  $H\_s[i]$ 
        DO IF  $t_r[J] = t_s[J]$ 
        THEN <join & output selected fields>
    };
```

Metodi di join



31

Simple hash join (4)

- ✚ Una valutazione di prima approssimazione del costo è:

$$\text{✚ } NP_s + 2 \times NT_s + NP_r + NT_r$$

$NP_s + 2 \times NT_s$ è il costo del passo 1

(lettura di s e costruzione di H_s)

$NP_r + NT_r$ è il costo del passo 2

(lettura di r e accesso a H_s)

✚ Osservazioni

- La relazione utilizzata per costruire la hash table è di solito quella con minore cardinalità, o con il minor numero di tuple residue nel caso siano presenti anche predicati locali.
- L'efficienza decresce all'aumentare delle collisioni, in quanto cresce il numero di accessi e confronti inutili.
- Il metodo consente, con un semplice variante, di eseguire left e right outer-join.
- Unità hardware per l'hashing hanno reso fattibile l'implementazione hardware di questo metodo.

Metodi di join



32