Ottimizzazione di interrogazioni

ALBERTO BELUSSI

ANNO ACCADEMICO 2018-2019

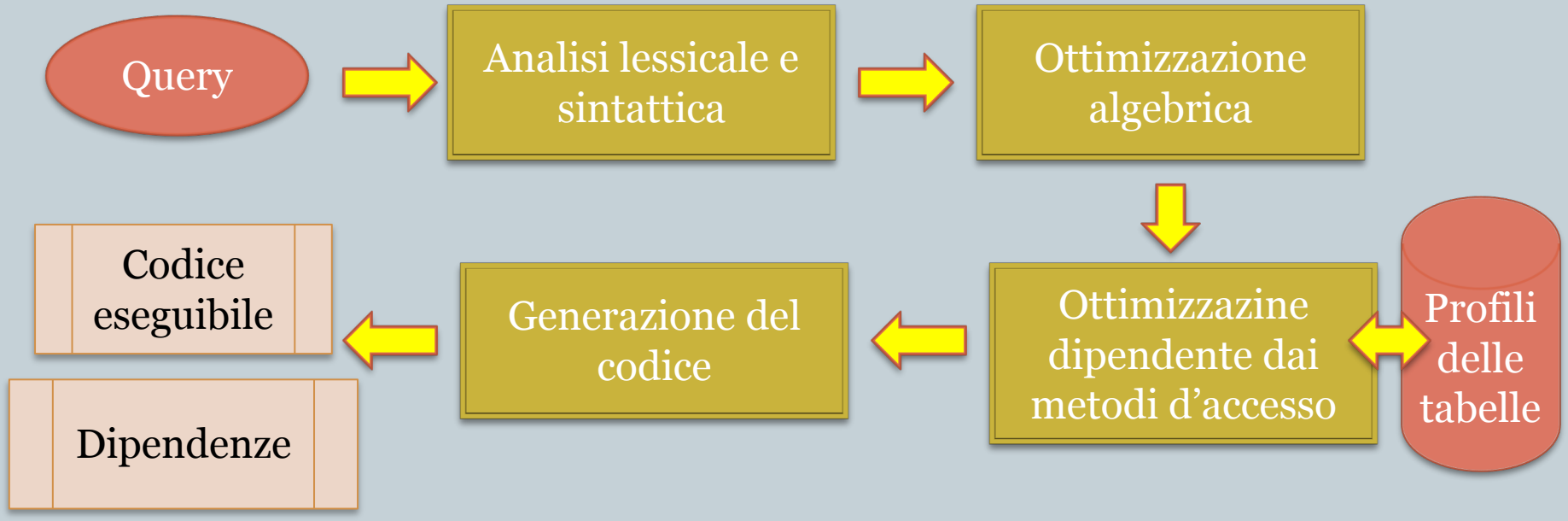
Osservazione

- Ogni interrogazione sottomessa al DBMS viene espressa in un linguaggio dichiarativo (ad esempio SQL)
- E' quindi necessario trovare un equivalente espressione in linguaggio procedurale (ad esempio in algebra relazionale) per generare un piano di esecuzione.
- L'espressione algebrica va ottimizzata rispetto alle caratteristiche del DBMS a livello fisico (metodi d'accesso disponibili) e della base di dati corrente (statistiche del dizionario dei dati)

Ottimizzazione

"Compilazione" di un'interrogazione

- Analisi lessicale e sintattica
- Ottimizzazione algebrica (indipendente dal modello di costo)
- Ottimizzazione basata sui costi di esecuzione



Ottimizzazione algebrica

L'ottimizzazione algebrica si basa fondamentalmente sulle regole di ottimizzazione già note dell'algebra relazionale:

- Anticipo delle selezioni (selection push)
- Anticipo delle proiezioni (projection push)

Ottimizzazione dipendente dai metodi di accesso

Metodi di accesso disponibili:

- Scansione (scan) delle tuple di una relazione
- Ordinamento di un insieme di tuple
- Accesso diretto alle tuple attraverso indice
- Diverse implementazioni del join

Scansione

Varianti:

- **SCAN + PROIEZIONE SENZA ELIMINAZIONE DI DUPLICATI**
- **SCAN + SELEZIONE IN BASE AD UN PREDICATO**
- **SCAN + INSERIMENTO/CANCELLAZIONE/MODIFICA**

Costo di una scansione sulla relazione **R**:

$$NP(\mathbf{R})$$

NP(**R**) = *Numero Pagine* dati della relazione **R**.

Ordinamento

L'ordinamento viene utilizzato per:

- ordinare il risultato di un'interrogazione (clausola order by),
- eliminare duplicati (select distinct),
- raggruppare tuple (group by);

Ordinamento su memoria secondaria: "*Z-way Sort-Merge*"

FASI:

- Sort interno: si leggono una alla volta le pagine della tabella; le tuple di ogni pagina vengono ordinate facendo uso di un algoritmo di sort interno (es. Quicksort); ogni pagina così ordinata, detta anche "*run*", viene scritta su memoria secondaria in un file temporaneo
- Merge: applicando uno o più passi di fusione, le "*run*" vengono unite, fino a produrre un'unica "*run*"

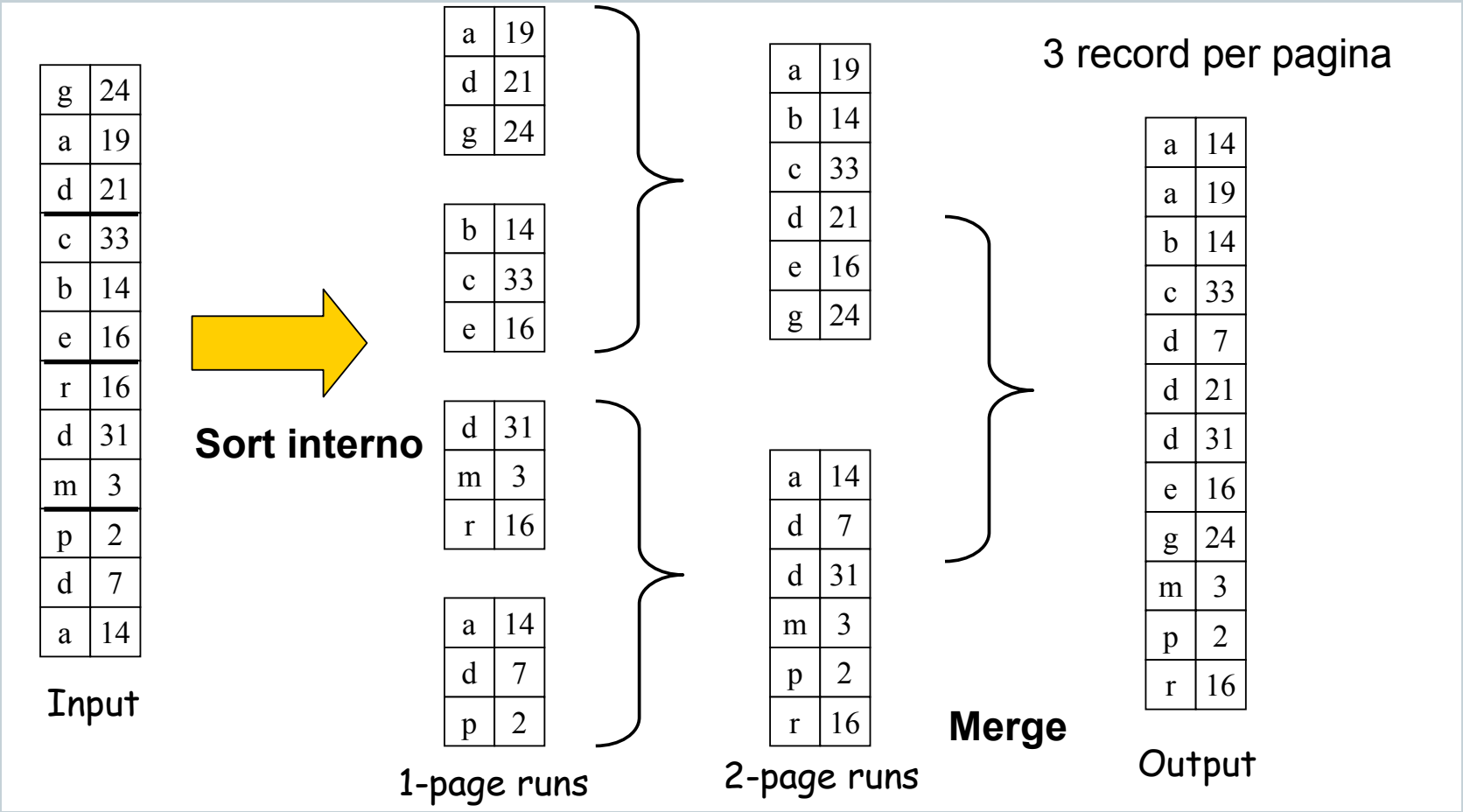
Z-way Sort-Merge

Esempio

Supponiamo di dover ordinare un input che consiste di una tabella di *NP* pagine e di avere a disposizione solo *NB* buffer in memoria centrale, con $NB < NP$.

Per semplicità consideriamo il caso base a due vie ($Z = 2$), e supponiamo di avere a disposizione solo 3 buffer in memoria centrale ($NB = 3$).

Z-way Sort-Merge: esempio



g	24
a	19
d	21
c	33
b	14
e	16
r	16
d	31
m	3
p	2
d	7
a	14

a	19
d	21
g	24

b	14
c	33
e	16

d	31
m	3
r	16

a	14
d	7
p	2

a	19
b	14
c	33
d	21
e	16
g	24

a	14
d	7
d	31
m	3
p	2
r	16

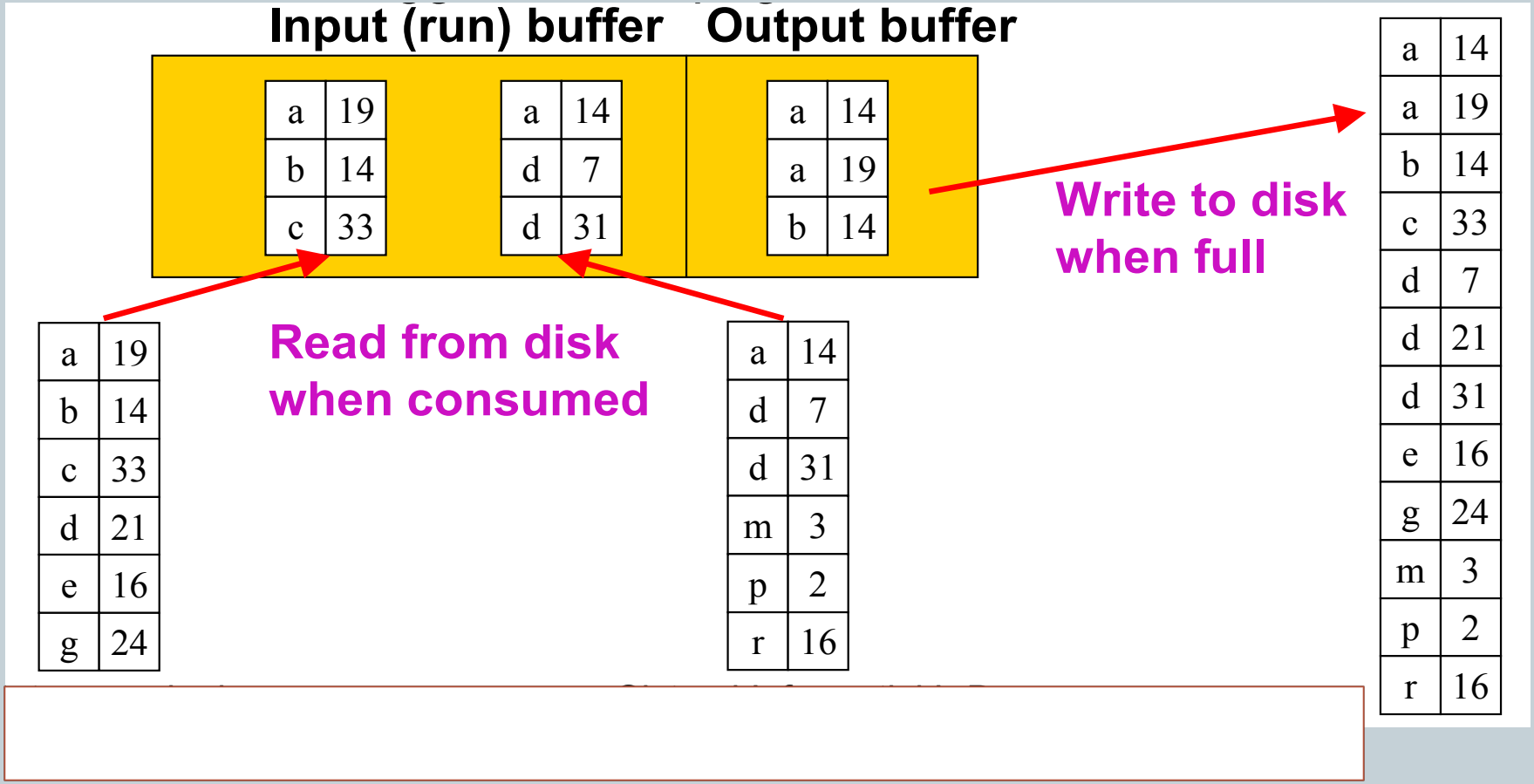
a	14
a	19
b	14
c	33
d	7
d	21
d	31
e	16
g	24
m	3
p	2
r	16

Z-way Sort-Merge

Dopo la fase di "sort interno", nel caso base $Z = 2$ si fondono 2 *run* alla volta:

- Con $NB = 3$, si associa un buffer a ognuna delle *run*, il terzo buffer serve per produrre l'output, 1 pagina alla volta.
- Si legge la prima pagina da ciascuna *run* e si genera la prima pagina dell'output; quando tutti i record di una pagina di *run* sono stati consumati, si legge un'altra pagina della *run*.

Z-way Sort-Merge



Z-way Sort-Merge: costo

Consideriamo come costo solo il numero accessi a memoria sec.

Nel caso base $Z = 2$ e con $NB = 3$ si può osservare che:

- Nella fase di sort interno si leggono e si riscrivono NP pagine
- Ad ogni passo di merge si leggono e si riscrivono NP pagine

Il numero di passi di merge («fusione») è pari a:

$$\lceil \log_2(NP) \rceil$$

in quanto ad ogni passo il numero di *run* si dimezza ($Z = 2$).

Il costo complessivo è pertanto pari a:

$$2 \times NP \times \left(\lceil \log_2 NP \rceil + 1 \right)$$

Accesso diretto via indice

Interrogazioni che fanno uso dell'indice:

- Selezioni con condizione atomica di uguaglianza (A = v):
  - richiede indice hash o B^+-tree.
- Selezioni con condizione di range (A >= v1 AND A <= v2):
  - richiede indice B^+-tree.
- Selezioni con condizione costituita da una congiunzione di condizioni di uguaglianza (A = v1 AND B = v2):
  - in questo caso si sceglie per quale delle condizioni di uguaglianza utilizzare l'indice; la scelta ricade sulla condizione più selettiva. L'altra si verifica direttamente sulle pagine dati.
- Selezioni con condizione costituita da una disgiunzione di condizioni di uguaglianza (A = v1 OR B = v2):
  - in questo caso è possibile utilizzare più indici in parallelo, facendo un merge dei risultati eliminando i duplicati oppure, se manca anche solo uno degli indici, è necessario eseguire una scansione sequenziale.

Algoritmi per il join

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.

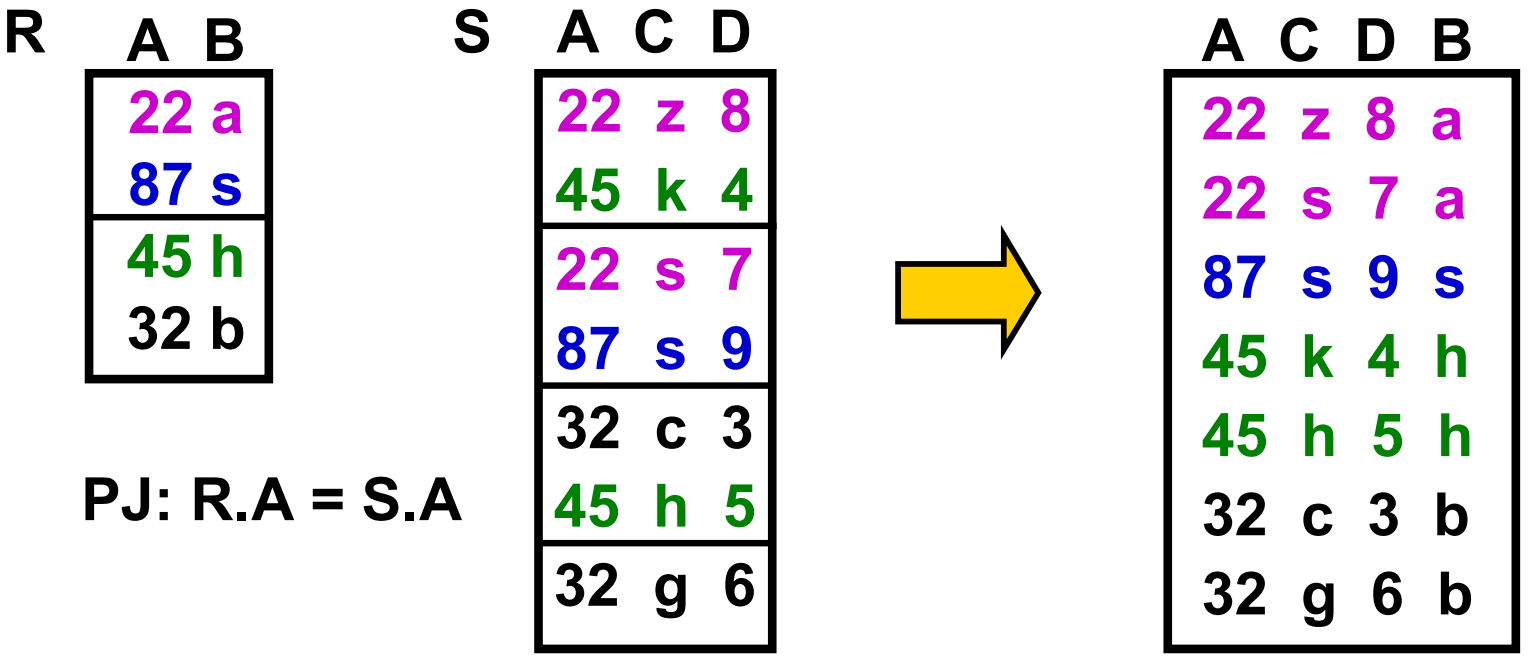
Nested-Loop JOIN

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**)

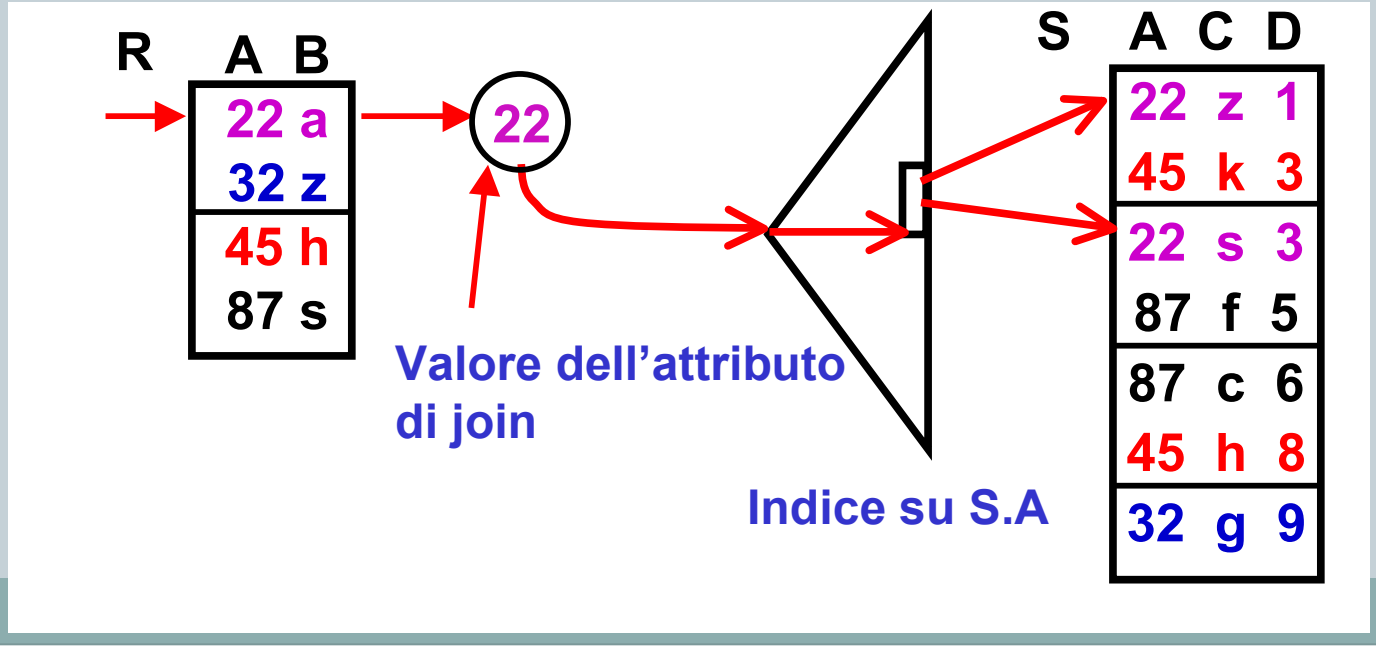
Si ricordi che:

NR(**R**) = numero tuple 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

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:

SELETTIVITA' di A

$$NP(\mathbf{R}) + NR(\mathbf{R}) * (\text{ProfIndice} + NR(\mathbf{S})/VAL(\mathbf{A},\mathbf{S}))$$

accessi a memoria secondaria

dove:

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

Merge-Scan JOIN

Il Merge-Scan Join è applicabile quando entrambi gli insiemi di tuple in input sono ordinati sugli attributi di 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

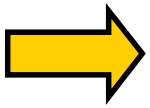
R

A	B
22	a
32	s
45	h
87	b

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

PJ: R.A = S.A

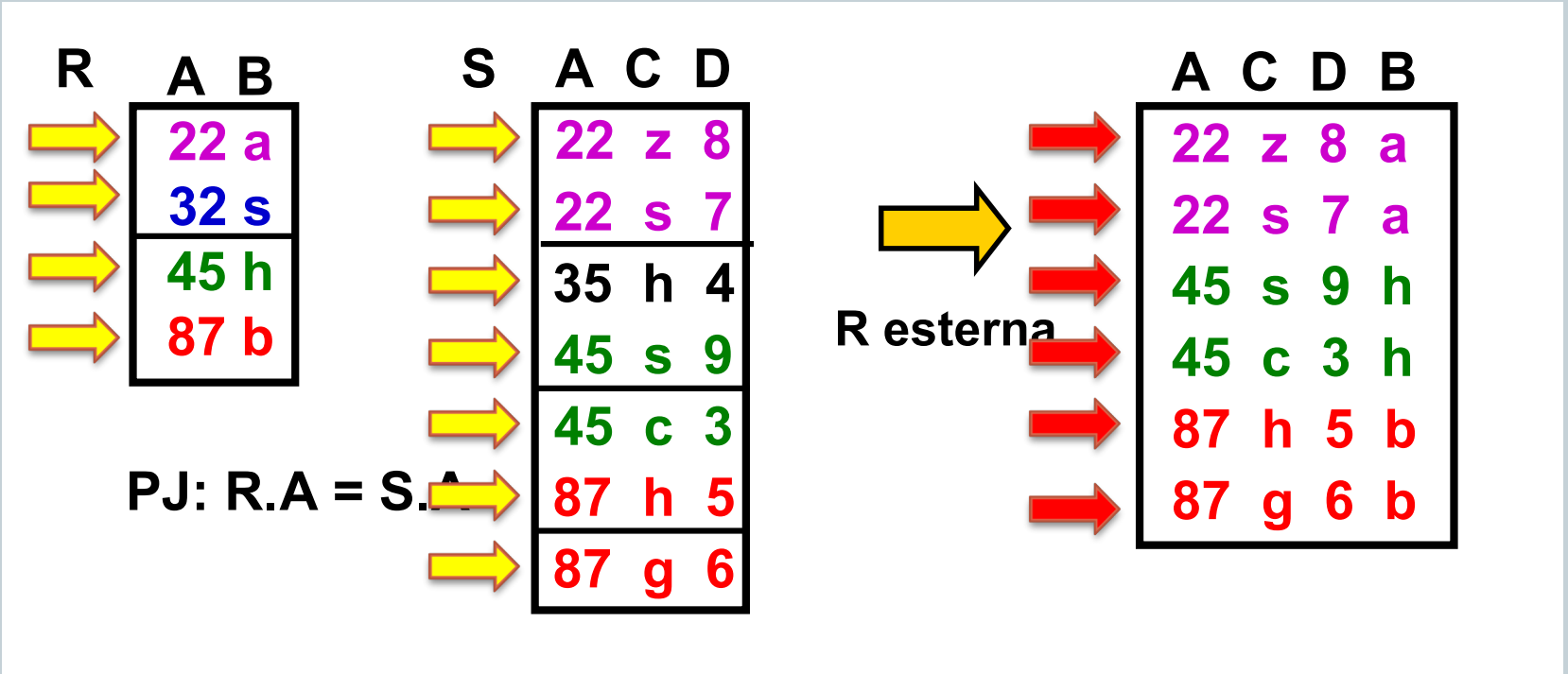


R esterna

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

Esempio di scansione dei due file durante il merge join



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:

$$NP(\mathbf{R}) + NP(\mathbf{S})$$

se si accede sequenzialmente alle due relazioni.

Con indici il costo può arrivare al massimo a:

$$NR(\mathbf{R}) + NR(\mathbf{S})$$

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 $t_R.\mathbf{J} = t_S.\mathbf{J}$ solo se $H(t_R.\mathbf{J}) = H(t_S.\mathbf{J})$
- Se, viceversa, $H(t_R.\mathbf{J}) <> H(t_S.\mathbf{J})$, allora sicuramente è $t_R.\mathbf{J} <> t_S.\mathbf{J}$

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 risulta essere dell'ordine di:

$$NP(\mathbf{R}) + NP(\mathbf{S})$$

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)

Scelta finale del piano di esecuzione

- Si sceglie l'albero con costo approssimato minimo.

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 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**)