

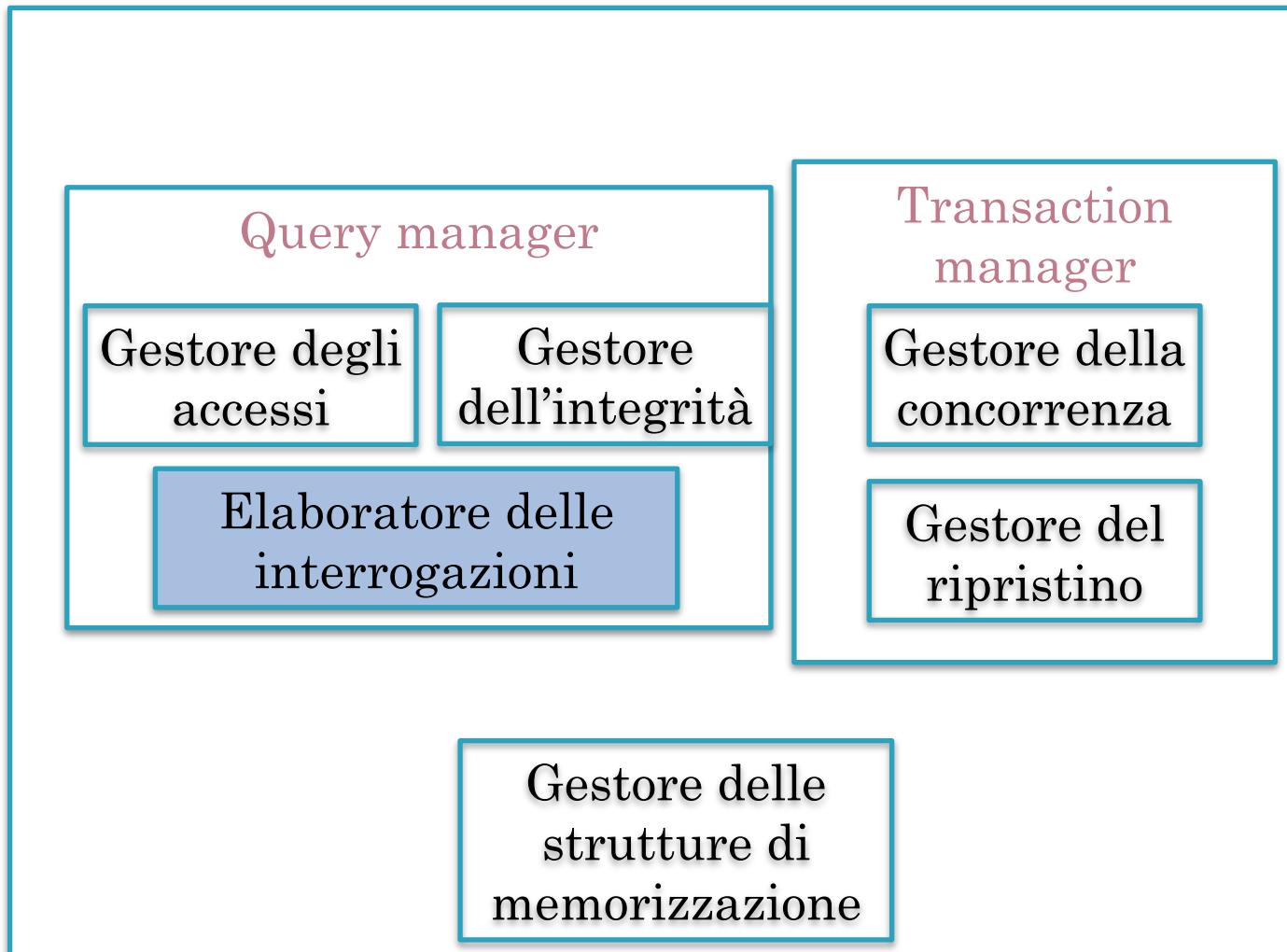
## ELABORAZIONE DI INTERROGAZIONI

# RICAPITOLIAMO

- Finora abbiamo visto
  - come organizzare il contenuto della base di dati su disco
  - quali strutture ausiliarie si possono creare per rendere più efficiente l'accesso ai record dei file
- Adesso
  - vediamo come il sistema esegue una interrogazione, sfruttando le strutture ausiliarie di accesso e la struttura attribuita ai file del livello fisico per determinare l'algoritmo di esecuzione più efficiente

# COMPONENTI DI UN DBMS

DBMS



# MOTIVAZIONI

# QUERY DI ESEMPIO

- Consideriamo la seguente interrogazione in SQL:

```
SELECT B, D  
FROM R, S  
WHERE R.A = "c"  ∧  S.E = 2  ∧  R.C=S.C
```

- Come sappiamo SQL è dichiarativo
- La stessa query si può rappresentare in algebra (linguaggio operazionale) con la seguente espressione:

$$\Pi_{B,D} [\sigma_{R.A="c" \wedge S.E=2 \wedge R.C=S.C} (R \times S)]$$

# RISULTATO

R	A	B	C	S	C	D	E
a	1	10		10	x	2	
b	1	20		20	y	2	
c	2	10		30	z	2	
d	2	35		40	x	1	
e	3	45		50	y	3	

B	D
2	x

# PIANO LOGICO I

Un primo algoritmo di esecuzione di alto livello deriva dall'espressione algebrica

$$\Pi_{B,D} [\sigma_{R.A = "c"} \wedge S.E = 2 \wedge R.C = S.C] (R \times S)$$

Algoritmo logico:

- eseguire il prodotto  
Cartesiano
- selezionare le tuple
- effettuare la proiezione

R.A	R.B	R.C	S.C	S.D	S.E
a	1	10	10	x	2
a	1	10	20	y	2
.	.	.	.	.	.
c	2	10	10	x	2
.	.	.	.	.	.

B	D
2	x

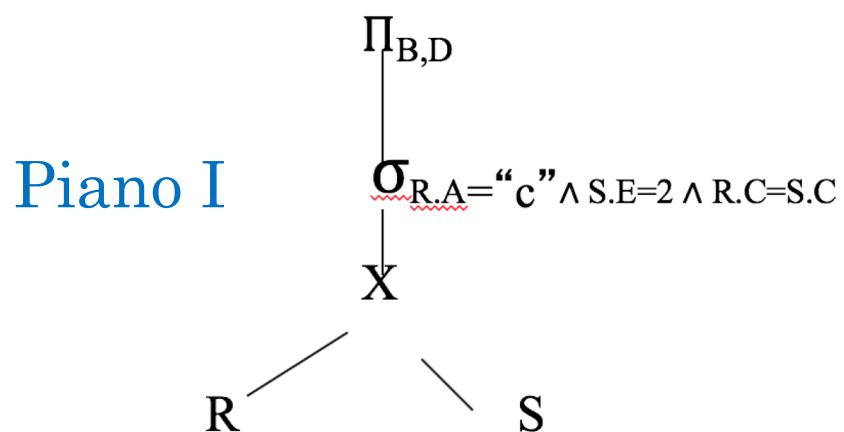
# PIANO LOGICO I

- L'espressione algebrica rappresenta un algoritmo (logico) di esecuzione che opera su tabelle e si può rappresentare come un albero

$$\Pi_{B,D} [\sigma_{R.A = "c" \wedge S.E = 2 \wedge R.C = S.C} (R \times S)]$$

## ○ Piano di esecuzione logico

- *Gli operatori manipolano tabelle e tuple, elementi del livello logico*



## PIANO LOGICO II

- Altra possibile strategia corrispondente a piano logico alternativo (quindi a espressione algebrica alternativa)

$$\prod_{B,D} [\sigma_{R.A="c"}(R) \bowtie \sigma_{S.E=2}(S)]$$

## Piano II

The diagram shows the composition of two permutations,  $\sigma(R)$  and  $\sigma(S)$ , mapping from sets  $R$  and  $S$  to a common target set  $T$ .

**Set R:**

A	B	C
a	1	10
b	1	20
c	2	10
d	2	35
e	3	45

**Set S:**

C	D	E
10	x	2
20	y	2
30	z	2
40	x	1
50	y	3

**Permutation  $\sigma(R)$ :**

A	B	C
c	2	10

**Permutation  $\sigma(S)$ :**

C	D	E
10	x	2
20	y	2
30	z	2

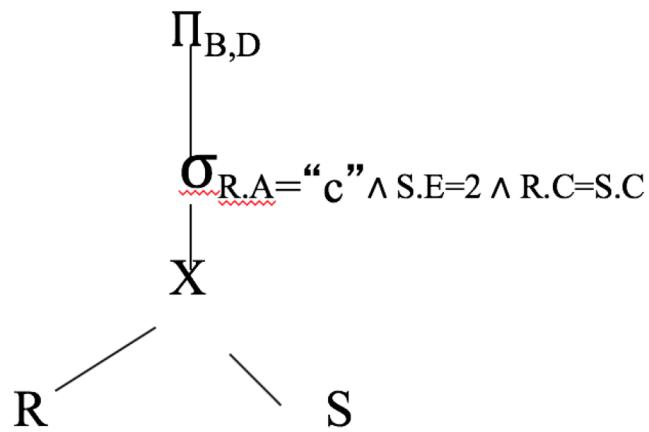
**Target Set T:**

10	x	2
20	y	2
30	z	2
40	x	1
50	y	3

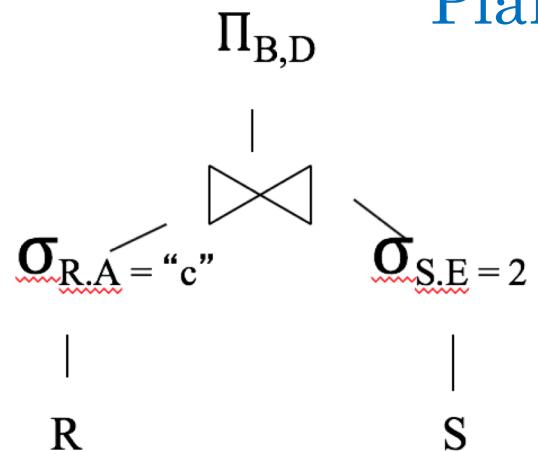
Arrows indicate the mapping:  $\sigma(R)$  maps row  $c$  to column  $B$  in  $\sigma(S)$ ;  $\sigma(S)$  maps column  $C$  to row  $10$  in the target set; and the target set maps back to column  $C$  in  $\sigma(S)$ .

# PIANO LOGICO I E PIANO LOGICO II

Piano I



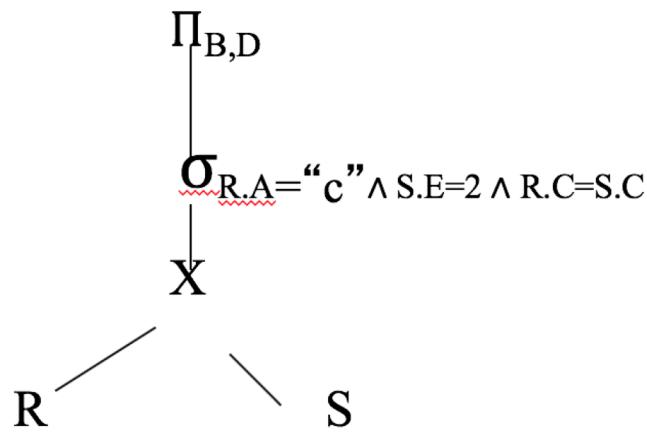
Piano II



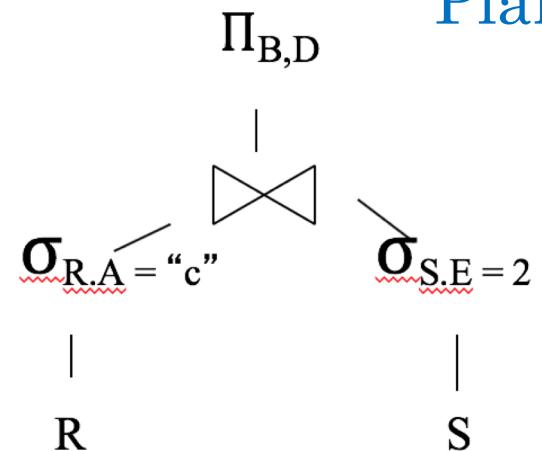
- Quale piano logico porta ad una esecuzione più efficiente?

# PIANO LOGICO I E PIANO LOGICO II

Piano I



Piano II



- Quale piano logico porta ad una esecuzione più efficiente?
- Piano II, in quanto evita l'esecuzione del prodotto cartesiano riducendo la dimensione dei risultati intermedi generati e il numero di operazioni eseguite

# DAL PIANO LOGICO AL PIANO FISICO

- I piani logici forniscono alcune indicazioni su come eseguire l’interrogazione ma non corrispondono ad algoritmi utilizzabili dal sistema per l’esecuzione
- I dati sono memorizzati su disco quindi gli algoritmi di esecuzione delle interrogazioni devono operare su record memorizzati nei file
- **Dato un certo database fisico**
  - Come vengono acceduti i file?
  - Come vengono eseguiti i vari operatori sul livello fisico?
  - Come vengono passati i risultati generati da un operatore all’altro?
- **Dal piano di esecuzione logico al piano di esecuzione fisico**
  - *Ogni operatore viene implementato tramite un algoritmo che manipola record*

# PIANO FISICO

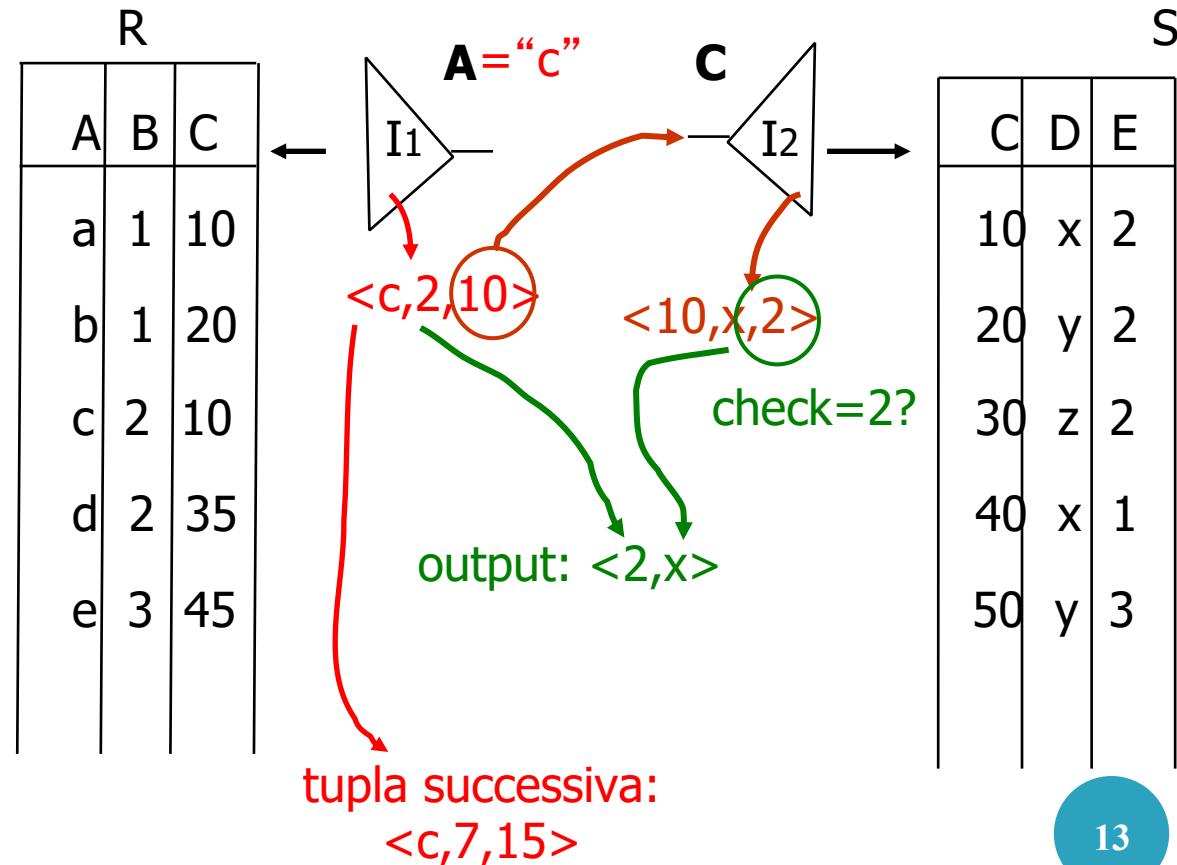
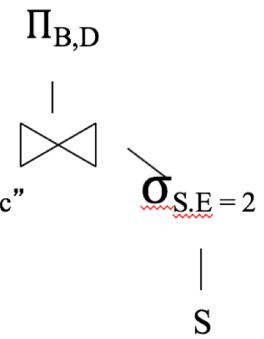
- Piano fisico

- si usa l'indice su R.A per selezionare i record corrispondenti alle tuple di R con R.A = "c"
- per ogni valore di R.C trovato, si usa l'indice su S.C per trovare i record corrispondenti alle tuple in join
- si eliminano i record di S tali che S.E ≠ 2
- si concatenano di record di R e S risultanti, proiettando su B e D e si restituisce la tupla
- si ripete il procedimento per ogni altra tupla restituita dall'indice su R.A

Partiamo da Piano logico II

*Ipotesi: il livello fisico contiene*

- File per R ordinato rispetto ad A  $\sigma_{R.A} = "c"$
- File per S ordinato rispetto a C
- Indice ordinato su R.A
- Indice ordinato su S.C



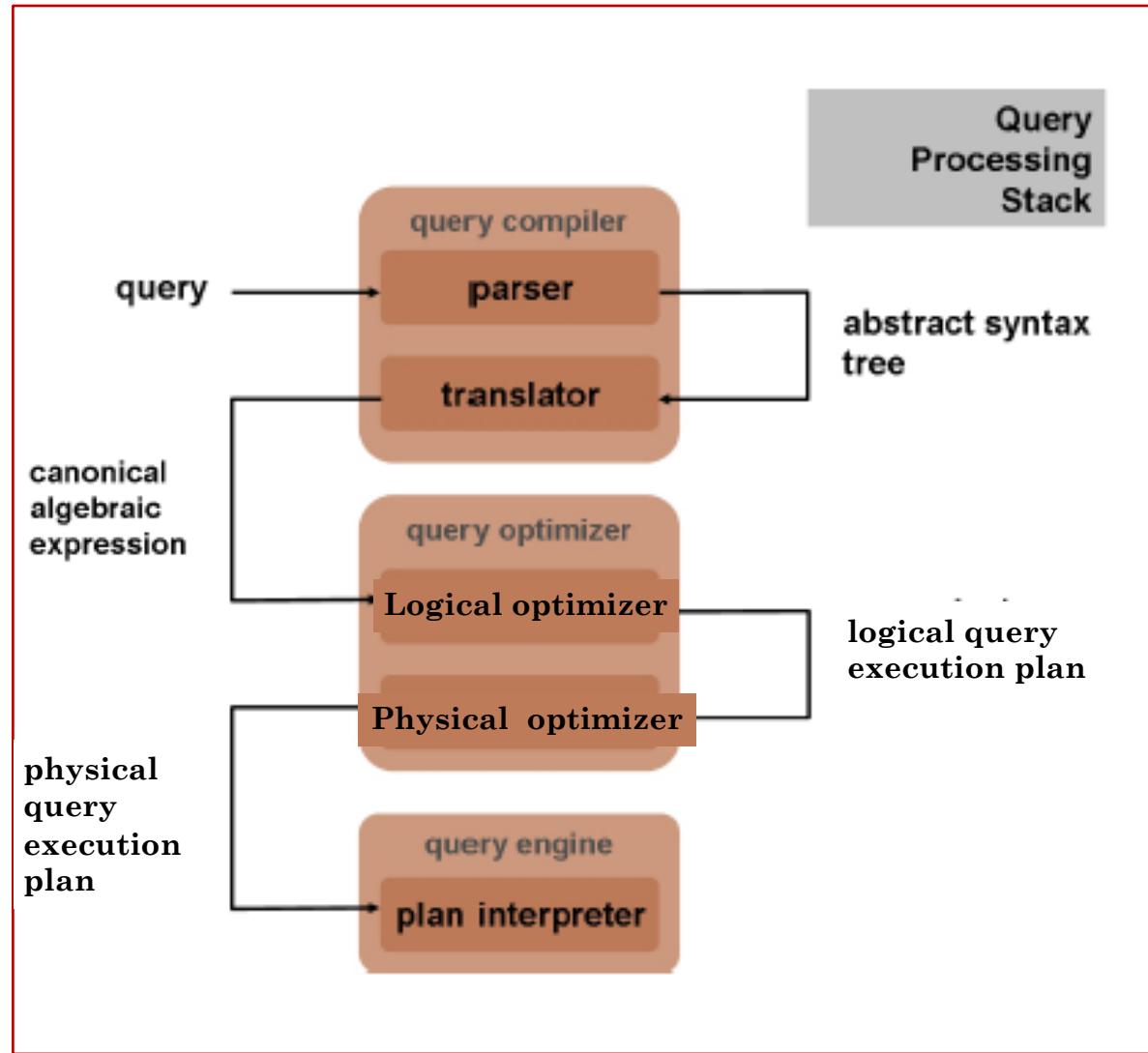
# ELABORAZIONE E OTTIMIZZAZIONE

- Per interrogazioni complesse esistono molteplici strategie logiche possibili
  - Molti piani di esecuzione logici
- **Scelto un piano logico e dato uno schema fisico,** esistono molteplici strategie fisiche che lo realizzano
  - Per ogni operatore algebrico esistono diversi possibili algoritmi
  - Molti piani fisici di esecuzione
- Il costo di determinare la strategia ottima può essere elevato
- Il vantaggio in termini di tempo di esecuzione che se ne ricava è tuttavia tale da rendere preferibile eseguire l'ottimizzazione

# PASSI NELL'ELABORAZIONE DI UNA INTERROGAZIONE

# PASSI NELL'ESECUZIONE DI UNA INTERROGAZIONE

Query  
processor



# QUERY DI ESEMPIO

- Schema:

StarsIn(title, year, starName)

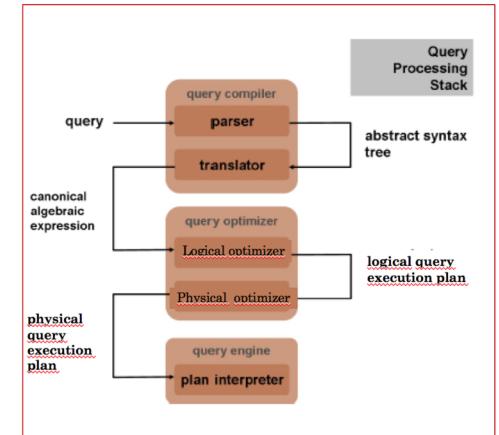
MovieStar(name, address, gender,birthdate)

- Trovare i clienti che in cui hanno recitato attori nati nel 1960

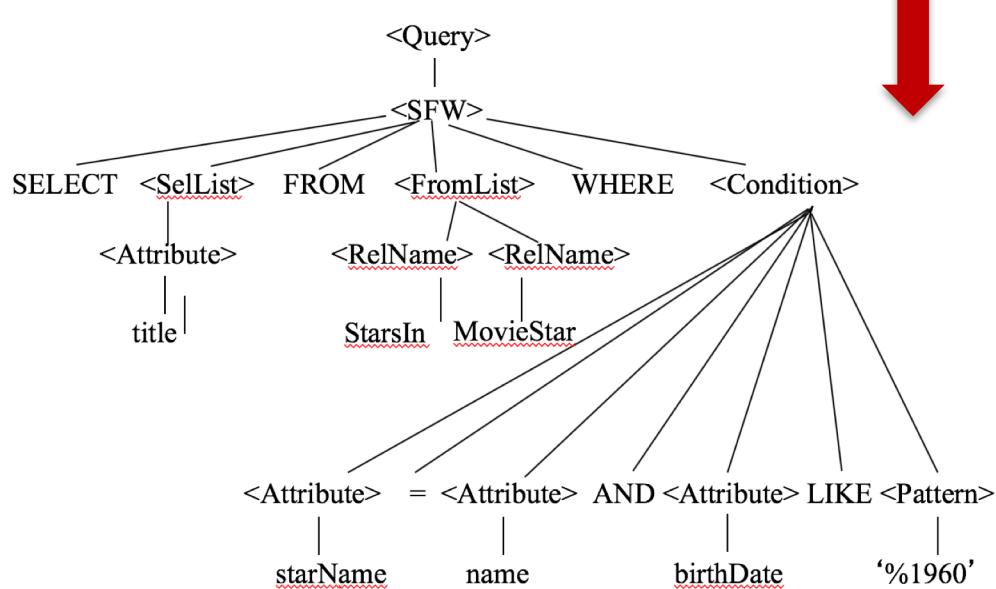
```
SELECT title  
FROM StarsIn, MovieStar  
WHERE starName = name AND  
      birthdate LIKE ‘%1960’;
```

# PASSI - PARSER

- **input:** interrogazione SQL
- **output:** parse tree (o abstract syntax tree)
- viene controllata la correttezza sintattica della query SQL e ne viene generata una rappresentazione interna (in termini di parse tree)

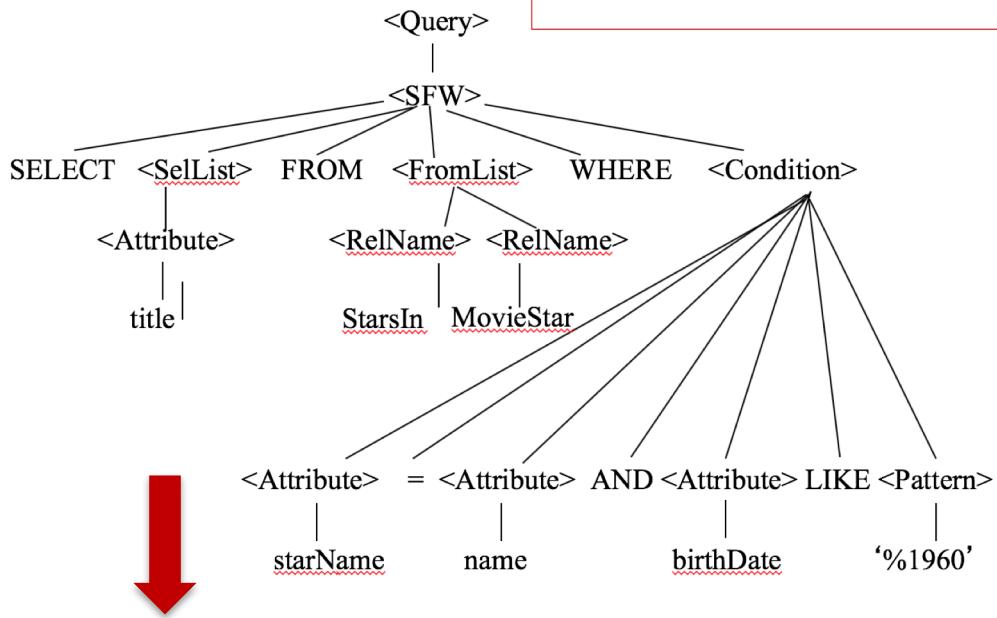


```
SELECT title  
FROM StarsIn, MovieStar  
WHERE starName = name AND  
birthdate LIKE '%1960';
```



# PASSI - TRANSLATOR

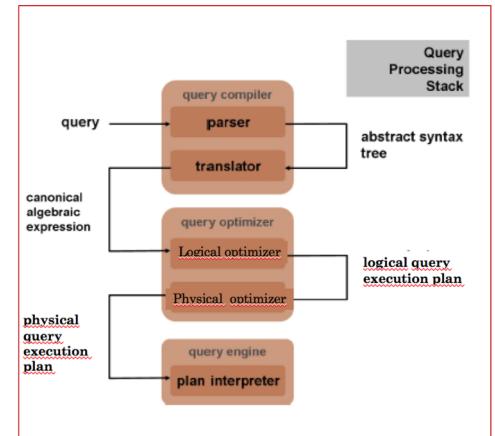
- **input:** parse tree
- **output:** espressione algebrica canonica
- viene generate una espressione algebrica corrispondente a
  - prodotto cartesiano delle relazioni in clausola FROM seguita da
  - condizioni di selezione, da clausola WHERE seguita da
  - condizioni di proiezione, da clausola SELECT



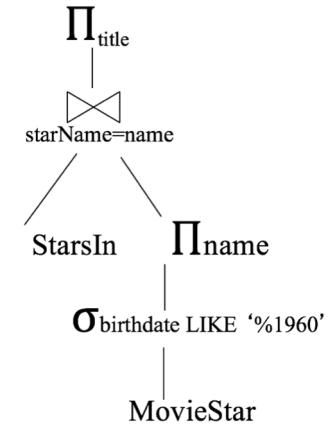
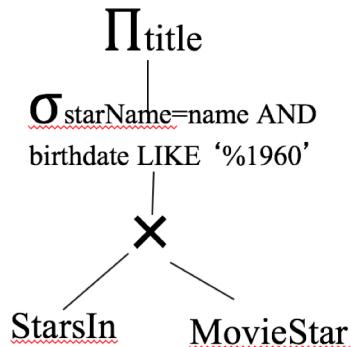
$$\Pi_{B,D} [\sigma_{\text{starName}=\text{name} \wedge \text{birthdate} \text{ LIKE } \%1960} \\ (\text{StarsIn} \times \text{MovieStar})]$$

# PASSI – OTTIMIZZAZIONE LOGICA

- **input:** espressione algebrica canonica
- **output:** piano di esecuzione logico ottimizzato (logical query plan – LQP)
- **Piano di esecuzione logico:** espressione algebrica (in un'algebra estesa) per l'interrogazione, rappresentata come albero
- L'espressione algebrica canonica viene rappresentata come piano di esecuzione logico
- Il piano di esecuzione logico iniziale viene poi trasformato in un piano equivalente ma più efficiente da eseguire, usando le proprietà dell'algebra relazionale

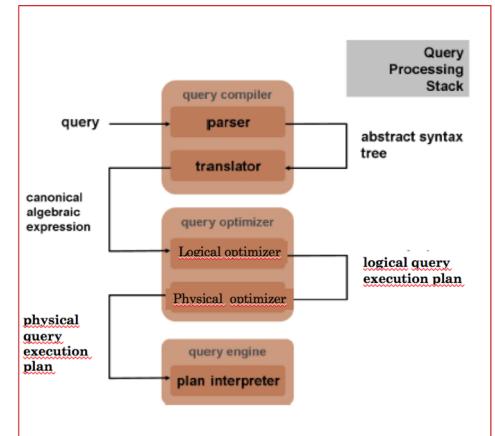


$\Pi_{B,D} [\sigma_{\text{starName}=\text{name} \wedge \text{birthdate} \text{ LIKE } \%1960}$   
 $(\text{StarsIn} \times \text{MovieStar})]$

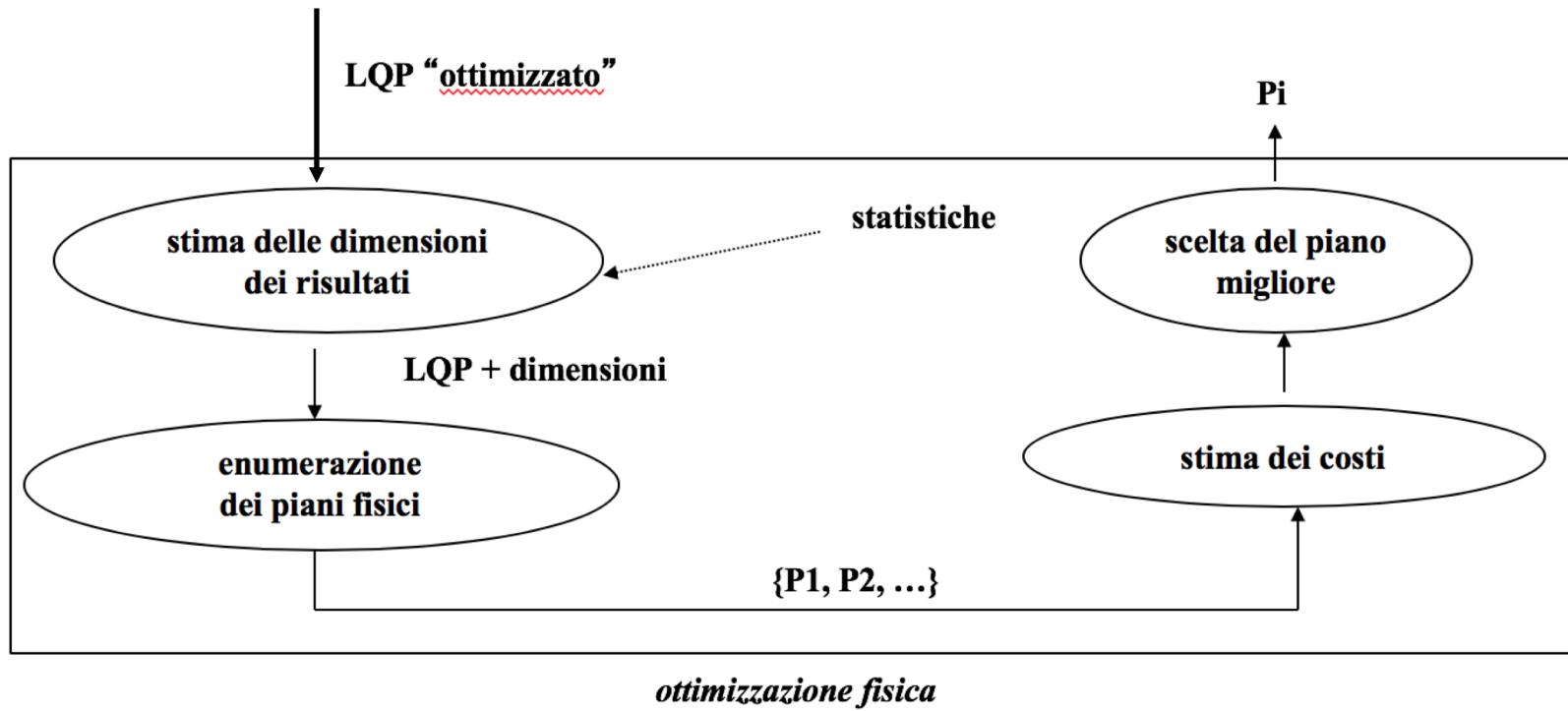


# PASSI – OTTIMIZZAZIONE FISICA

- **input:** piano di esecuzione logica ottimizzato
- **output:** piano di esecuzione fisico ottimale (physical query plan - PQP)
- **Piano di esecuzione fisico:** algoritmo di esecuzione dell'interrogazione, rappresentato come albero, sul livello fisico
- si seleziona il piano di esecuzione fisico più efficiente, che riduce quindi gli accessi a disco
- si determina in modo preciso come la query sarà eseguita (per esempio si determina che indici si useranno)
- *la scelta avviene considerando tutti i possibili piani fisici che realizzano il piano logico scelto, valutando il costo di ognuno di essi e scegliendo il piano fisico di minor costo*



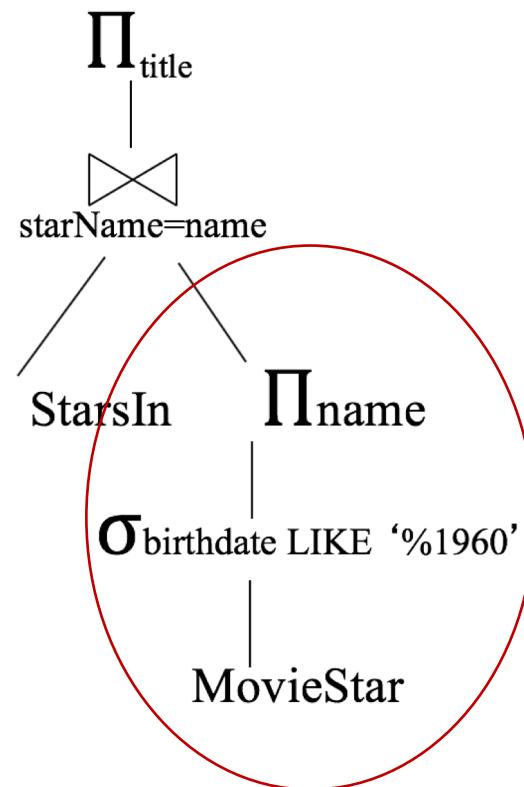
# PASSI – OTTIMIZZAZIONE FISICA



# PASSI – OTTIMIZZAZIONE FISICA

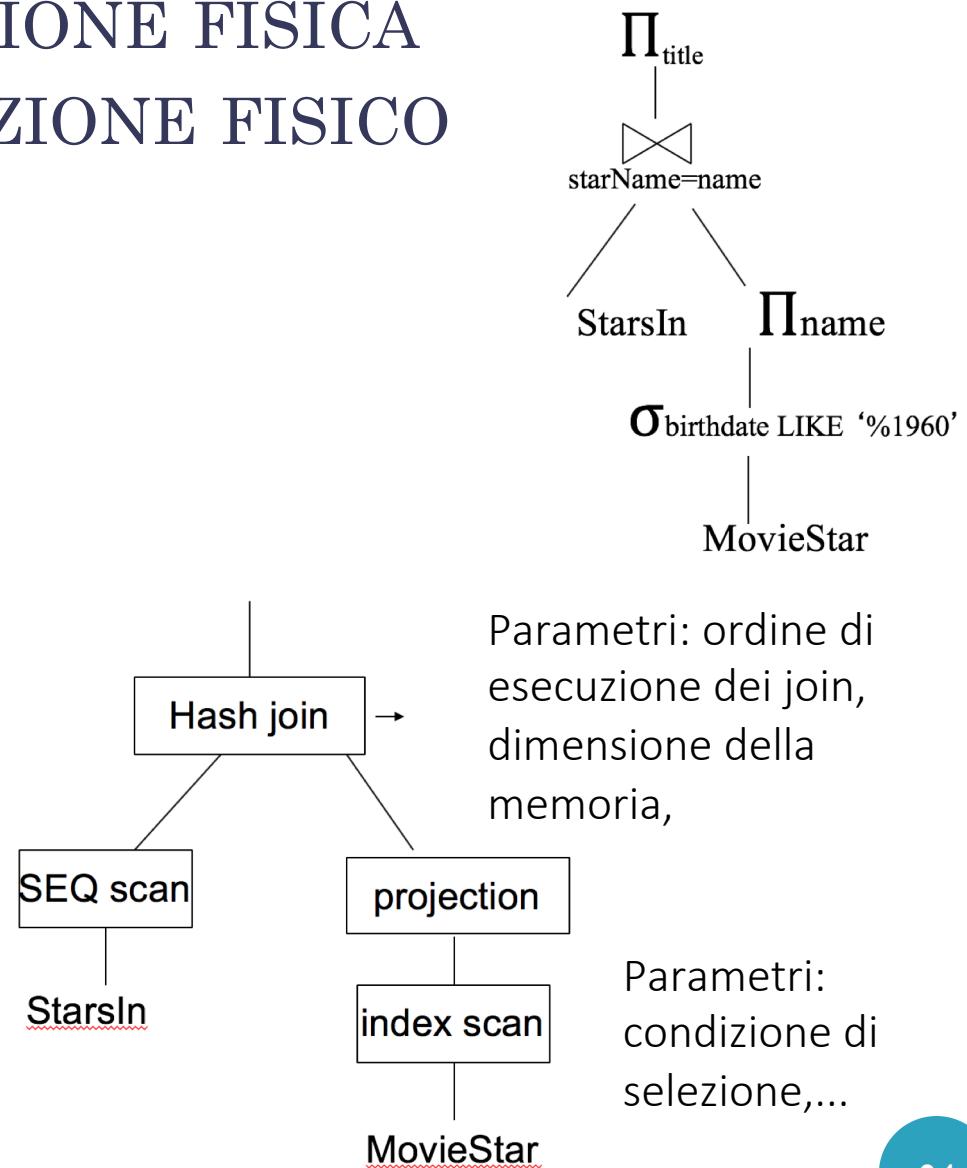
## STIMA DELLA DIMENSIONE DEI RISULTATI

- necessità di determinare la dimensione (fisica!) delle relazioni corrispondenti ai sottoalberi perché diventano input per gli operatori padre
- importanti per calcolare il costo totale del piano, cioè il numero totale di accessi a disco
- stima effettuata utilizzando statistiche mantenute dal sistema nei cataloghi



# PASSI – OTTIMIZZAZIONE FISICA UN PIANO DI ESECUZIONE FISICO

- Ogni nodo dell'albero di esecuzione logico corrisponde a un nodo dell'albero di esecuzione fisico in cui si specifica
  - si sceglie un algoritmo per l'esecuzione dell'operatore
  - si specificano i parametri necessari all'esecuzione dell'operatore con l'algoritmo scelto, a livello fisico



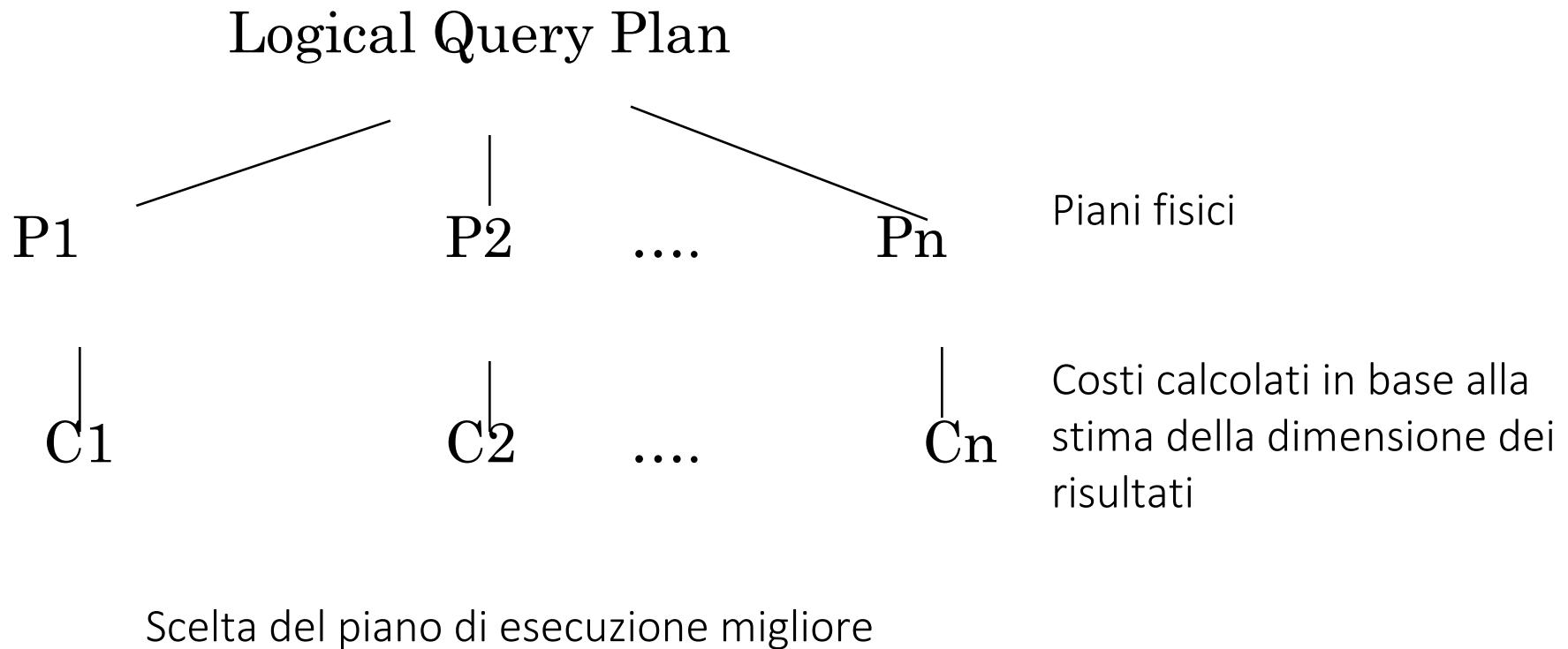
# PASSI – OTTIMIZZAZIONE FISICA

## UN PIANO DI ESECUZIONE FISICO

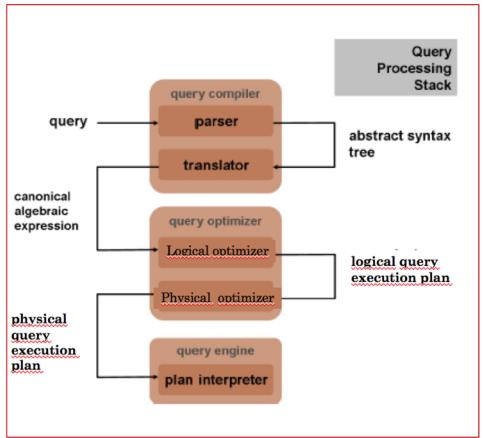
- Il piano fisico può contenere nodi aggiuntivi, corrispondenti a ulteriori operazioni da eseguire per motivi di efficienza (ad esempio ordinamento)
- A livello di piano fisico, si sceglie anche
  - un ordine di esecuzione per le operazioni associative e commutative (join, unione, intersezione)
  - una modalità per passare i risultati intermedi sono passati da un operatore al successivo

# PASSI – OTTIMIZZAZIONE FISICA

## STIMA DEI COSTI E SCELTA DEL PIANO MIGLIORE



# PASSI – ESECUZIONE DEL PIANO



- **input:** piano di esecuzione fisico ottimale
  - **output:** risultato della interrogazione
- 
- Il piano di esecuzione fisico viene eseguito sul livello fisico
  - Il risultato ottenuto viene restituito come risultato dell'interrogazione

# NEL SEGUITO – FORMATO DELLE INTERROGAZIONI

- Per il momento, consideriamo interrogazioni **senza sottointerrogazioni**, cioè della forma

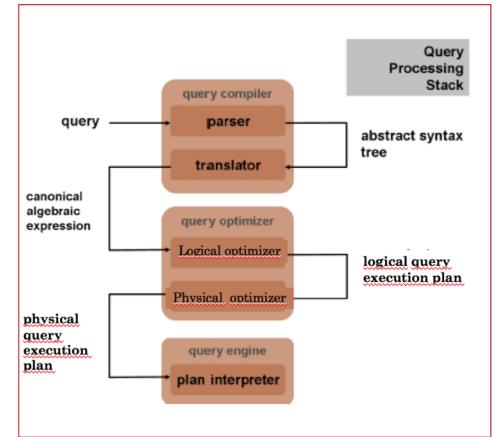
```
SELECT Lista Attributi  
FROM Lista Relazioni  
WHERE condizione in CNF  
[GROUP BY Lista Attributi]  
[HAVING Condizione]  
[ORDER BY]
```

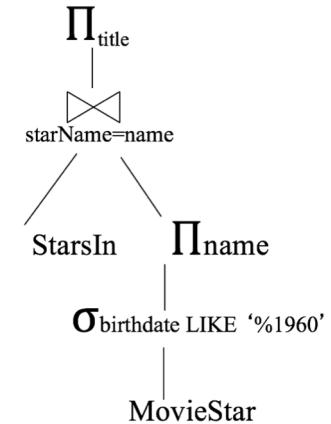
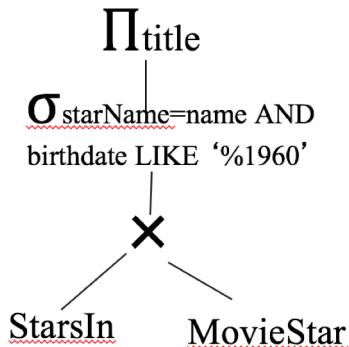
- vedremo poi come si possono trattare le interrogazioni con sottointerrogazioni

# OTTIMIZZAZIONE LOGICA

# OTTIMIZZAZIONE LOGICA

- **input:** espressione algebrica canonica
- **output:** piano di esecuzione logico ottimizzato (logical query plan – LQP)
- **Piano di esecuzione logico:** espressione algebrica (in un'algebra estesa) per l'interrogazione, rappresentata come albero
- L'espressione algebrica canonica viene rappresentata come piano di esecuzione logico
- Il piano di esecuzione logico iniziale viene poi trasformato in un piano equivalente ma più efficiente da eseguire, usando le proprietà dell'algebra relazionale



$$\Pi_{B,D} [\sigma_{\text{starName}=\text{name} \wedge \text{birthdate} \text{ LIKE } \%1960'} \\ (\text{StarsIn} \times \text{MovieStar})]$$


# OPERATORI LOGICI

- Operatori di un'**algebra relazionale estesa**, che include
  - operazioni algebra relazionale
  - altri operatori presenti in SQL
    - unione, intersezione, differenza senza eliminazione dei duplicati
    - proiezione senza eliminazione dei duplicati
    - ordinamento
    - raggruppamento
- In questa parte ci concentriamo sui classici operatori relazionali

# APPROCCIO

- L'ottimizzazione logica si basa su **equivalenze algebriche**
- Tali equivalenze vengono utilizzate come **regole di riscrittura**, guidati da opportune **euristiche** per passare da un'espressione algebrica (quindi da un LQP) ad un'altra, ad essa equivalente ma piu` efficiente

# EQUIVALENZE ALGEBRICHE

- Due espressioni e1 ed e2 dell'algebra relazionale sono dette **equivalenti** se, per ogni possibile base di dati in input D, producono lo stesso risultato in output quando vengono eseguite su D

# EQUIVALENZE ALGEBRICHE

- Selezione

$$\sigma_{P1}(\sigma_{P2}(e)) \equiv \sigma_{P2}(\sigma_{P1}(e)) \equiv \sigma_{P1 \text{ AND } P2}(e)$$

- permette di gestire cascate di selezioni e stabilisce la commutatività della selezione

# EQUIVALENZE ALGEBRICHE

## o Proiezione

$$\Pi_{A_1, \dots, A_n}(\Pi_{B_1, \dots, B_m}(e)) \equiv \Pi_{A_1, \dots, A_n}(e)$$

- o permette di gestire cascate di proiezioni
- o vale se  $\{A_1, \dots, A_n\} \subseteq \{B_1, \dots, B_m\}$

# EQUIVALENZE ALGEBRICHE

## o Commutazione di selezione e proiezione

- se una selezione con predicato P coinvolge solo gli attributi A<sub>1</sub>,...,A<sub>n</sub>, allora

$$\Pi_{A_1, \dots, A_n}(\sigma_P(e)) \equiv \sigma_P(\Pi_{A_1, \dots, A_n}(e))$$

- più in generale, se il predicato P coinvolge anche gli attributi B<sub>1</sub>,...,B<sub>m</sub> che non sono tra gli attributi A<sub>1</sub>,...,A<sub>n</sub> allora

$$\Pi_{A_1, \dots, A_n}(\sigma_P(e)) \equiv \Pi_{A_1, \dots, A_n}(\sigma_P(\Pi_{A_1, \dots, A_n, B_1, \dots, B_m}(e)))$$

# EQUIVALENZE ALGEBRICHE

## ◦ Commutazione di selezione e prodotto Cartesiano

- se una selezione con predicato P coinvolge solo gli attributi di e1, allora

$$\sigma_P(e1 \times e2) \equiv \sigma_P(e1) \times e2$$

- come conseguenza, se  $P=P1 \text{ AND } P2$  dove P1 coinvolge solo gli attributi di e1 e P2 quelli di e2

$$\sigma_{P1 \text{ AND } P2}(e1 \times e2) \equiv \sigma_{P1}(e1) \times \sigma_{P2}(e2)$$

- inoltre se P1 coinvolge solo attributi di e1, mentre P2 coinvolge attributi di e1 e di e2

$$\sigma_{P1 \text{ AND } P2}(e1 \times e2) \equiv \sigma_{P2}(\sigma_{P1}(e1) \times e2)$$

# EQUIVALENZE ALGEBRICHE

- Commutazione di proiezione e prodotto Cartesiano
  - Sia  $A_1, \dots, A_n$  una lista di attributi di cui gli attributi  $B_1, \dots, B_m$  siano attributi di  $e_1$ , e i rimanenti  $C_1, \dots, C_k$  siano attributi di  $e_2$

$$\Pi_{A_1, \dots, A_n}(e_1 \times e_2) \equiv \Pi_{B_1, \dots, B_m}(e_1) \times \Pi_{C_1, \dots, C_k}(e_2)$$

# EQUIVALENZE ALGEBRICHE

- Selezioni, prodotto cartesiano e join
- Si può trasformare una selezione ed un prodotto cartesiano in un join, in accordo alla definizione di join

$$\sigma_P(e1 \times e2) \equiv e1 \triangleright\triangleleft_P e2$$

# EQUIVALENZE ALGEBRICHE

- Prodotto Cartesiano e join
- commutatività
  - $e1 \triangleright\triangleleft_F e2 \equiv e2 \triangleright\triangleleft_F e1$
  - $e1 \triangleright\triangleleft e2 \equiv e2 \triangleright\triangleleft e1$
  - $e1 \times e2 \equiv e2 \times e1$
- associatività
  - $(e1 \triangleright\triangleleft_{F1} e2) \triangleright\triangleleft_{F2} e3 \equiv e1 \triangleright\triangleleft_{F1} (e2 \triangleright\triangleleft_{F2} e3)$
  - $(e1 \triangleright\triangleleft e2) \triangleright\triangleleft e3 \equiv e1 \triangleright\triangleleft (e2 \triangleright\triangleleft e3)$
  - $(e1 \times e2) \times e3 \equiv e1 \times (e2 \times e3)$

# EQUIVALENZE ALGEBRICHE

- Altre equivalenze (unione e differenza)
- commutatività e associatività dell'unione
  - $(e1 \cup e2) \cup e3 \equiv e1 \cup (e2 \cup e3)$
  - $e1 \cup e2 \equiv e2 \cup e1$
- commutazione di selezione e unione
  - $\sigma_P(e1 \cup e2) \equiv \sigma_P(e1) \cup \sigma_P(e2)$
- commutazione di selezione e differenza
  - $\sigma_P(e1 - e2) \equiv \sigma_P(e1) - \sigma_P(e2) \equiv \sigma_P(e1) - e2$
- commutazione di proiezione e unione
  - $\Pi_{A1, \dots, An}(e1 \cup e2) \equiv \Pi_{A1, \dots, An}(e1) \cup \Pi_{A1, \dots, An}(e2)$

# EQUIVALENZE ALGEBRICHE

$$\sigma_{P_1 \wedge P_2}(e) \equiv \sigma_{P_2 \wedge P_1}(e) \equiv \sigma_{P_1}(\sigma_{P_2}(e))$$

$$\Pi_{A_1, \dots, A_n}(\Pi_{B_1, \dots, B_m}(e)) \equiv \Pi_{A_1, \dots, A_n}(e)$$

$$\Pi_{A_1, \dots, A_n}(\sigma_P(e)) \equiv \sigma_P(\Pi_{A_1, \dots, A_n}(e))$$

$$\Pi_{A_1, \dots, A_n}(\sigma_P(e)) \equiv \Pi_{A_1, \dots, A_n}(\sigma_P(\Pi_{D_1, \dots, D_p}(e)))$$

$$a(P) \subseteq \{A_1, \dots, A_n\}$$

$$a(P) \subseteq \{D_1, \dots, D_p\},$$

$$\{A_1, \dots, A_n\} \subseteq \{D_1, \dots, D_p\}$$

$$a(P) \subseteq a(e_1)$$

$$\{B_1, \dots, B_m\} = \{A_1, \dots, A_n\} \cap a(e_1)$$

$$\{C_1, \dots, C_k\} = \{A_1, \dots, A_n\} \cap a(e_2)$$

$$\sigma_P(e_1 \times e_2) \equiv \sigma_P(e_1) \times e_2$$

$$\Pi_{A_1, \dots, A_n}(e_1 \times e_2) \equiv \Pi_{B_1, \dots, B_m}(e_1) \times \Pi_{C_1, \dots, C_k}(e_2)$$

$$\sigma_P(e_1 \cup e_2) \equiv \sigma_P(e_1) \cup \sigma_P(e_2)$$

$$\Pi_{A_1, \dots, A_n}(e_1 \cup e_2) \equiv \Pi_{A_1, \dots, A_n}(e_1) \cup \Pi_{A_1, \dots, A_n}(e_2)$$

- + relazione tra join, prodotto Cartesiano e selezione
- + commutatività e associatività di join e prodotto cartesiano

# OSSERVAZIONE

- Nessuna equivalenza si riferisce all’ordine con cui eseguire un insieme di join
- l’ordine di esecuzione dei join, infatti, viene deciso nella fase successiva (ottimizzazione fisica), sulla base di
  - informazioni relative alla dimensione delle relazioni
  - valutazione del costo dei diversi ordini di esecuzione

# EURISTICHE

- Le **euristiche** permettono di trasformare le equivalenze in regole di riscrittura
- Si basano sull'idea di
  - anticipare il più possibile le operazioni che permettono di ridurre la dimensione dei risultati intermedi
    - selezione e proiezione
  - fattorizzare condizioni di selezione complesse e lunghe liste di attributi in proiezioni, per aumentare la possibilità di applicare regole di riscrittura
- Le altre regole vengono applicate per favorire l'applicazione delle euristiche descritte sopra
- Vediamo solo le principali

# EURISTICA 1

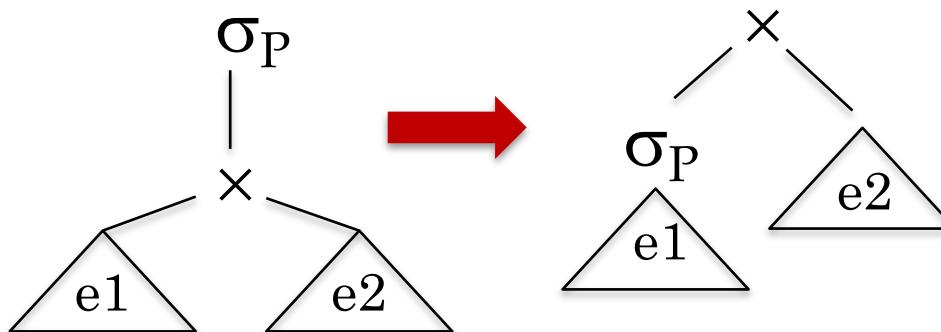
Eseguire le operazioni di selezione ( $\sigma$ ) il più presto possibile

equivalenza

$$\sigma_P(e1 \times e2) \equiv \sigma_P(e1) \times e2$$

regola di riscrittura

$$\sigma_P(e1 \times e2) \rightarrow \sigma_P(e1) \times e2$$



# EURISTICA 2

Eseguire le operazioni di proiezione ( $\pi$ ) il più presto possibile

equivalenze

$$\Pi_{A1, \dots, An}(\sigma_P(e)) \equiv \sigma_P(\Pi_{A1, \dots, An}(e))$$

$$\Pi_{A1, \dots, An}(e1 \times e2) \equiv \Pi_{B1, \dots, Bm}(e1) \times \Pi_{C1, \dots, Ck}(e2)$$

regole di riscrittura

$$\Pi_{A1, \dots, An}(\sigma_P(e)) \rightarrow \sigma_P(\Pi_{A1, \dots, An}(e))$$

$$\Pi_{A1, \dots, An}(e1 \times e2) \rightarrow \Pi_{B1, \dots, Bm}(e1) \times \Pi_{C1, \dots, Ck}(e2)$$

# EURISTICA 3

Introdurre ulteriori proiezioni nell'espressione, gli unici attributi da non eliminare sono quelli che

- appaiono nel risultato della query
- sono necessari in operazioni successive

## equivalenza

$$\Pi_{A1, \dots, An}(\sigma_P(e)) \equiv \Pi_{A1, \dots, An}(\sigma_P(\Pi_{A1, \dots, An, B1, \dots, Bm}(e)))$$

## regola di riscrittura

$$\Pi_{A1, \dots, An}(\sigma_P(e)) \rightarrow \Pi_{A1, \dots, An}(\sigma_P(\Pi_{A1, \dots, An, B1, \dots, Bm}(e)))$$

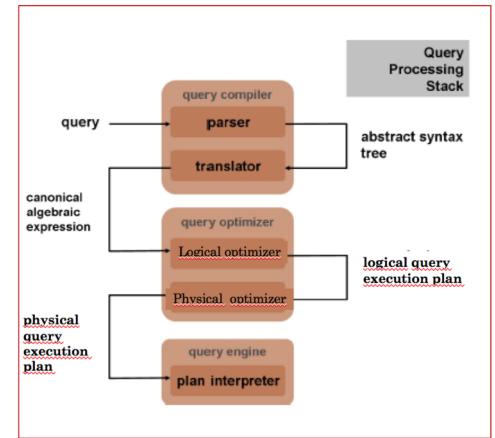
# OUTPUT DELLA FASE DI OTTIMIZZAZIONE LOGICA

- L'output della fase di ottimizzazione logica è un singolo LQP ottimizzato
- un'alternativa sarebbe quella di considerare diversi equivalenti LQP nella fase di ottimizzazione fisica
- in genere questa alternativa non viene utilizzata
  - per ogni piano logico restituito, bisognerebbe individuare il piano fisico ottimale
  - aumento importante del numero di piani di cui stimare il costo

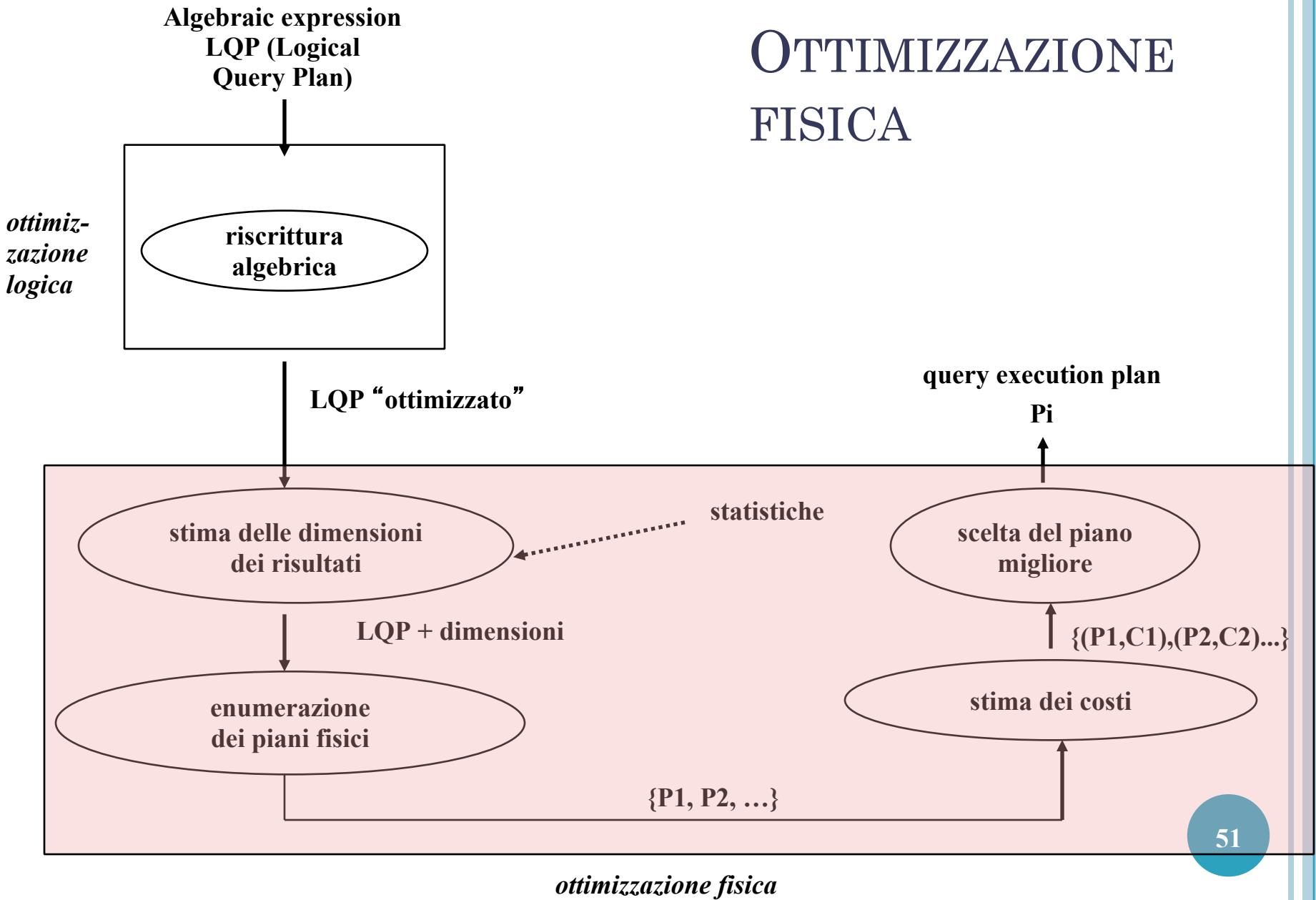
# OTTIMIZZAZIONE FISICA

# OTTIMIZZAZIONE FISICA

- **input:** piano di esecuzione logica ottimizzato
- **output:** piano di esecuzione fisico ottimale (physical query plan - PQP)
- **Piano di esecuzione fisico:** algoritmo di esecuzione dell'interrogazione, rappresentato come albero, sul livello fisico
- si seleziona il piano di esecuzione fisico più efficiente, che riduce quindi gli accessi a disco
- si determina in modo preciso come la query sarà eseguita (per esempio si determina che indici si useranno)
- *la scelta avviene considerando tutti i possibili piani fisici che realizzano il piano logico scelto, valutando il costo di ognuno di essi e scegliendo il piano fisico di minor costo*



# OTTIMIZZAZIONE FISICA



# OTTIMIZZAZIONE FISICA

- Algoritmi
  - per l'elaborazione dei singoli operatori logici
  - per l'elaborazione complessiva del piano fisico
- Scelta del piano fisico
  - stima della dimensione dei risultati e del costo di un piano di esecuzione
  - identificazione dello spazio di ricerca dei piani
  - trattamento delle sottointerrogazioni

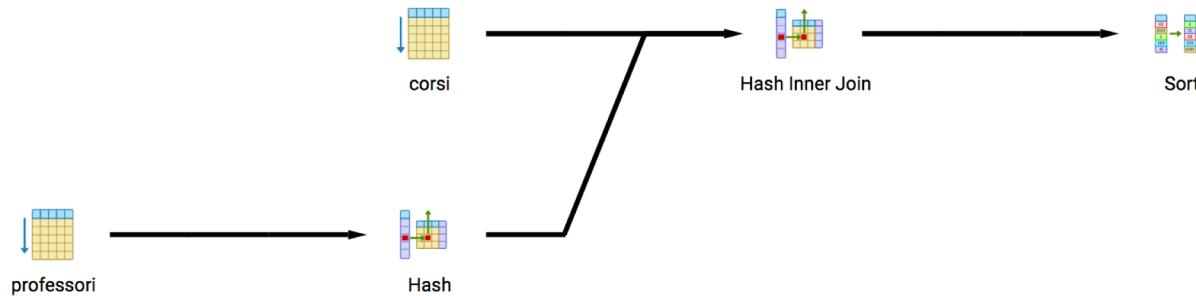
# OTTIMIZZAZIONE FISICA

Algoritmi per l'elaborazione degli operatori logici

# PIANI DI ESECUZIONE FISICI

- Ogni piano di esecuzione fisico si compone di una serie di **operatori fisici** connessi ad albero
- Le foglie del piano di accesso sono le relazioni di base presenti nel piano logico ottimizzato
- Gli altri nodi sono operatori che agiscono su 1 o 2 insiemi di tuple in input e producono 1 insieme di tuple in output

```
select denominazione, corsodilaurea, cognome  
from corsi join professori on corsi.professore= professori.id  
where attivato = TRUE  
order by corsodilaurea, denominazione, cognome;
```



# OPERATORI FISICI

- Sono **implementazioni specifiche di un operatore logico** (ad esempio del join)
- in funzione dei valori delle statistiche, dei parametri di sistema (es. dimensione del buffer pool) e della logica dell'algoritmo
- per ogni operatore logico, esistono diversi **algoritmi di realizzazione**
- algoritmi diversi possono utilizzare diverse informazioni contenute a livello fisico

# OPERATORI FISICI

- Si basano su tre principali tecniche:
  - **iterazione**: si esaminano le tuple della relazione di input sequenzialmente (scansione sequenziale)
  - **indici**: se è specificata una selezione o una condizione di join su una relazione di base, si usa un indice per esaminare solo le tuple che soddisfano la condizione
  - **partizionamento**: si partizionano le tuple in base ad una chiave (ad esempio usando una funzione hash) si decompone il problema eseguendo l'operazione prima sulle partizioni (più piccolo, meno costoso) e poi si integrano i risultati

# OPERATORI FISICI

- Diversi algoritmi possono portare a costi differenti
- **Costo** = numero di operazioni di I/O (non consideriamo il tempo di CPU)
- nel determinare il costo delle varie operazioni in genere non si tiene in considerazione il costo di scrittura dell'output finale
- questo perché
  - tale costo non dipende dalla strategia utilizzata, ma solo dalla dimensione dell'output, che è uguale per tutte le strategie
  - non è detto che si voglia effettivamente scrivere su disco l'output
- non vedremo i costi nel dettaglio ma forniremo solo intuizioni

# ELABORAZIONE DELLE FOGLIE DEL PIANO – ACCESSO ALLE RELAZIONI DI BASE

- Le foglie dei piani di esecuzione fisici rappresentano sempre tabelle di base
- L'accesso alle relazioni di base nell'ambito dell'esecuzione di una interrogazione avviene tramite **cammini di accesso**
- Un **cammino di accesso** descrive come accedere al file dei **dati** di una relazione per ritrovare le tuple di interesse e tiene conto di eventuali operazioni di selezione definite sulle relazioni di base, per le quali esista un indice utilizzabile per la loro esecuzione

# ACCESSO ALLE RELAZIONI DI BASE – CAMMINO DI ACCESSO

- Un **cammino di accesso** per una relazione di base è
  - una **scansione sequenziale**, oppure
  - un **indice più una condizione di selezione** (detta **predicato di ricerca**) per la quale esiste un indice utilizzabile per individuare le tuple che soddisfano la condizione
- Diversi piani fisici per lo stesso piano logico possono utilizzare diversi cammini di accesso alle relazioni di base

```
SELECT *
FROM R
WHERE R.A = 5
```



Livello fisico:  
qualsiasi

Livello fisico:  
indice su  $I_R(A)$

*Cammini di accesso*

- scansione sequenziale

*Cammini di accesso:*

- Scansione sequenziale
- $(I_R(A), A=5)$

```
SEQ SCAN R
Filter (A=5)
```

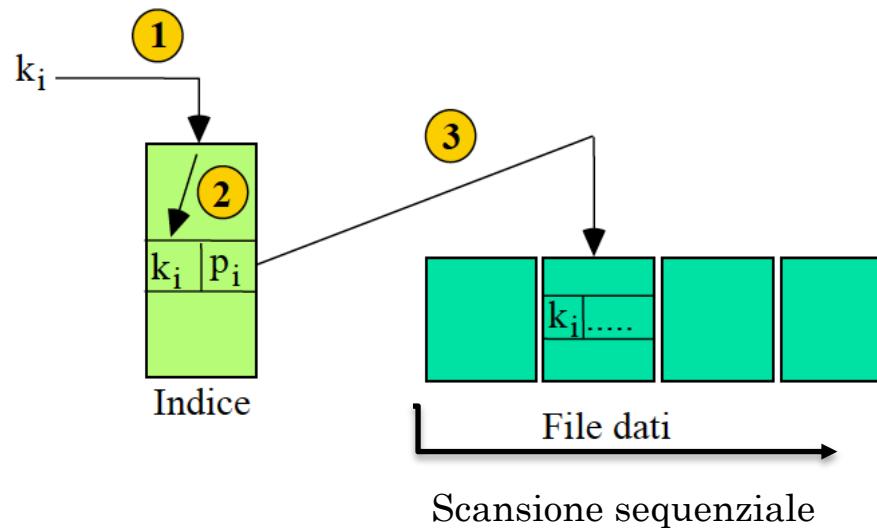
```
INDEX SCAN
(I_R(A), A = 5)
```

# ACCESSO ALLE RELAZIONI DI BASE - SCANSIONE SEQUENZIALE E USO DI INDICI

$$\sigma_A = k_i (R)$$

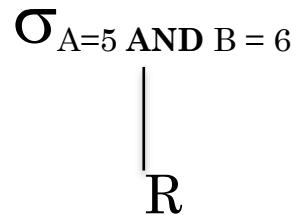
1. Accedere all'indice
2. Ricercare la coppia  $(k_i, p_i)$
3. Accedere alla pagina dati relativa

```
SELECT *  
FROM R  
WHERE A = k_i
```



# ESEMPIO

```
SELECT *
FROM R
WHERE R.A = 5 AND R.B = 6
```



Livello fisico:  
qualsiasi

```
SEQ SCAN R
Filter (A=5 AND B=6)
```

Livello fisico:  
indice su R.A ( $I_R(A)$ )  
indice su R.B ( $I_R(B)$ )

```
INDEX SCAN
(IR(A) A = 5)
Filter (B=6)
```

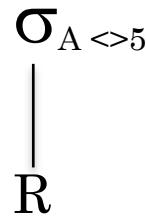
```
INDEX SCAN
(IB(A, B = 6)
Filter (A=5)
```

Tutti possibili cammini di accesso e relativi piani fisici

# ATTENZIONE

- Un indice può essere utilizzato come cammino di accesso a una relazione solo se **esiste una condizione di selezione che può essere eseguita utilizzando l'indice**

```
SELECT *  
FROM R  
WHERE R.A <> 5
```



Livello fisico:  
qualsiasi

```
SEQ SCAN R  
Filter (A<>5)
```

Livello fisico:  
indice su R.A ( $I_R(A)$ )

```
INDEX SCAN  
( $I_R(A)$ , A <> 5)
```

Questo non è un piano di accesso  
perché  $I_R(A)$  non può essere  
utilizzato per eseguire la selezione

# ACCESSO ALLE RELAZIONI DI BASE – SELEZIONE CON CONDIZIONE COMPOSTA

- In presenza di condizioni composte il sistema preferisce scegliere un PQP che contenga un cammino di accesso basato su una **condizione con indice che, se falsa, rende falsa tutta l'interrogazione (fattore booleano)**

```
SELECT *
FROM R
WHERE (R.A = 5 OR R.B = 6)
      AND R.C >10
```

$$\sigma_{(A=5 \text{ OR } B=6) \text{ AND } C > 10}$$

R

Livello fisico:  
qualsiasi

```
SEQ SCAN R
Filter
((R.A = 5 OR R.B = 6)
 AND R.C >10)
```

Livello fisico:  
Indice su R.A ( $I_R(A)$ )  
Indice su R.B ( $I_R(B)$ )  
Indice (ad albero) su R.C ( $I_R(C)$ )

INDEX SCAN  
( $I_R(A), A = 5$ )  
Filter ?

INDEX SCAN  
( $I_R(B), B = 6$ )  
Filter ?

R.C>10 fattore booleano  
**(il sistema sceglierà questo cammino di accesso per il PQP)**

```
INDEX SCAN
( $I_R(C), C > 10$ )
Filter (A=5 OR B=6)
```

# ACCESSO ALLE RELAZIONI DI BASE – SELEZIONE CON CONDIZIONE COMPOSTA

- Può anche essere scelto più di un cammino di accesso, combinando poi i risultati con un ulteriore operatore fisico

```
SELECT *  
FROM R  
WHERE R.A = 5 AND R.B = 6
```

$$\sigma_{A=5 \text{ AND } B=6}$$

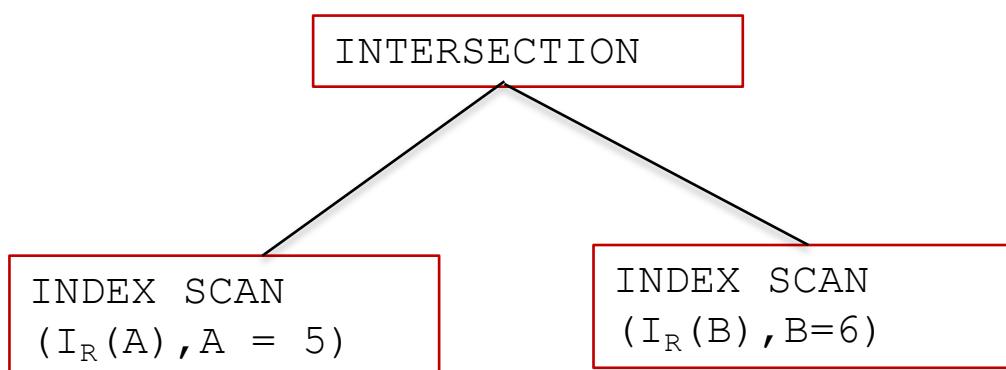
R

Livello fisico:

Indice su R.A ( $I_R(A)$ )

Indice su R.B ( $I_R(B)$ )

*Si scelgono solo cammini di accesso con indice su fattori booleani*



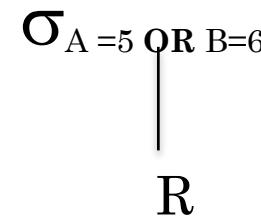
Un possibile piano fisico, che utilizza due cammini di accesso alla stessa relazione

## ACCESSO ALLE RELAZIONI DI BASE – SELEZIONE CON CONDIZIONE COMPOSTA

- I sistemi privilegiano cammini di accesso su fattori booleani
- Alcuni sistemi sono però prendono anche in considerazione piani di esecuzione basati su cammini di accesso con indice che non coinvolgono fattori booleani
- La maggior parte dei sistemi esistenti non gestisce in modo efficiente questo caso

# ACCESSO ALLE RELAZIONI DI BASE – SELEZIONE CON CONDIZIONE COMPOSTA

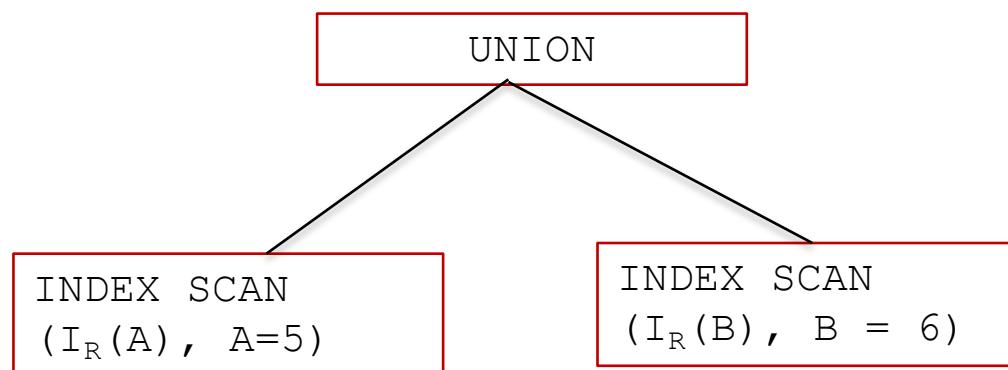
```
SELECT *  
FROM R  
WHERE R.A = 5 OR R.B = 6
```



Livello fisico:

Indice su R.A ( $I_R(A)$ )  
Indice su R.B ( $I_R(B)$ )

*In questo caso i cammini di accesso con indice non sono definiti su fattori booleani ma alcuni sistemi ammettono piani di questo tipo*



Un possibile piano fisico, che utilizza due cammini di accesso alla stessa relazione

# ACCESSO ALLE RELAZIONI DI BASE - COSTI

- Scansione sequenziale
  - Unica opzione in assenza di indici
  - $NB(R)$  – numero di blocchi del file per  $R$
- Accesso con indice: il costo dipende da
  - *tipo di indice* (ordinato, hash)
  - *numero di tuple che soddisfano la condizione* di selezione
  - *costo totale* =  
costo di determinare i riferimenti ai dati che soddisfano la condizione
    - 1 per hash
    - $h (\log N)$  per indice ad albero+  
costo per accedere ai corrispondenti blocchi dei dati (tipicamente maggiore)
  - clusterizzazione influisce

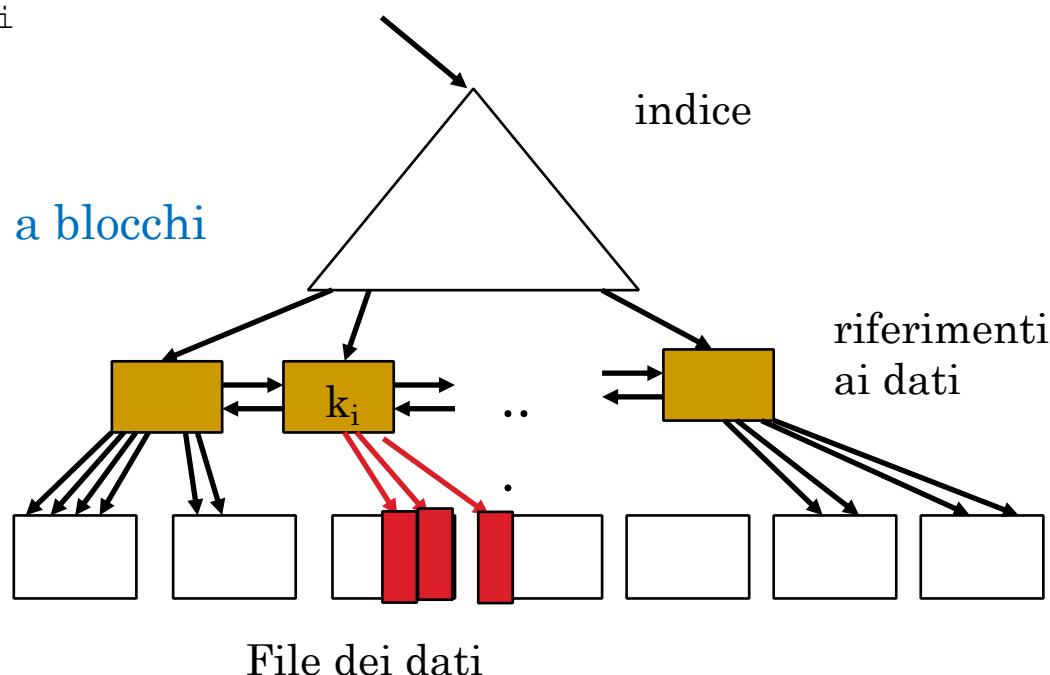
# ESEMPIO – INDICE ORDINATO CLUSTERIZZATO

$$\sigma_{A=k_i}(R)$$

```
SELECT *
FROM R
WHERE A = ki
```

Ogni blocco dati viene visitato al più una volta  
(sia per condizione di uguaglianza che per intervallo)

Costo:  $h + 2$  accessi a blocchi dati



# ESEMPIO – INDICE ORDINATO NON CLUSTERIZZATO

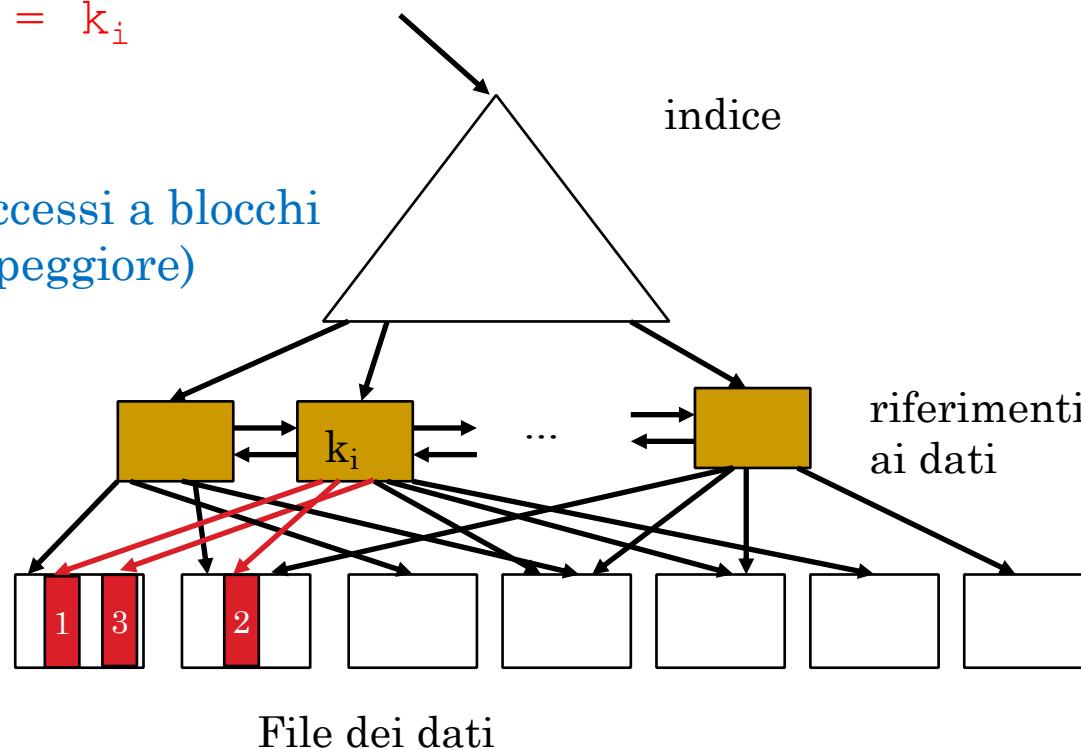
$$\sigma_{A=k_i}(R)$$

```
SELECT *  
FROM R  
WHERE A = ki
```

Lo stesso blocco può essere acceduto più volte  
(sia per uguaglianza che per intervallo)

Costo:  $h + 3$  accessi a blocchi  
dati (nel caso peggiore)

I numeri indicano  
l'ordine di accesso  
alle tuple (rosse) nei  
blocchi



# ESEMPIO – INDICE ORDINATO NON CLUSTERIZZATO

$$\sigma_{A=k_i}(R)$$

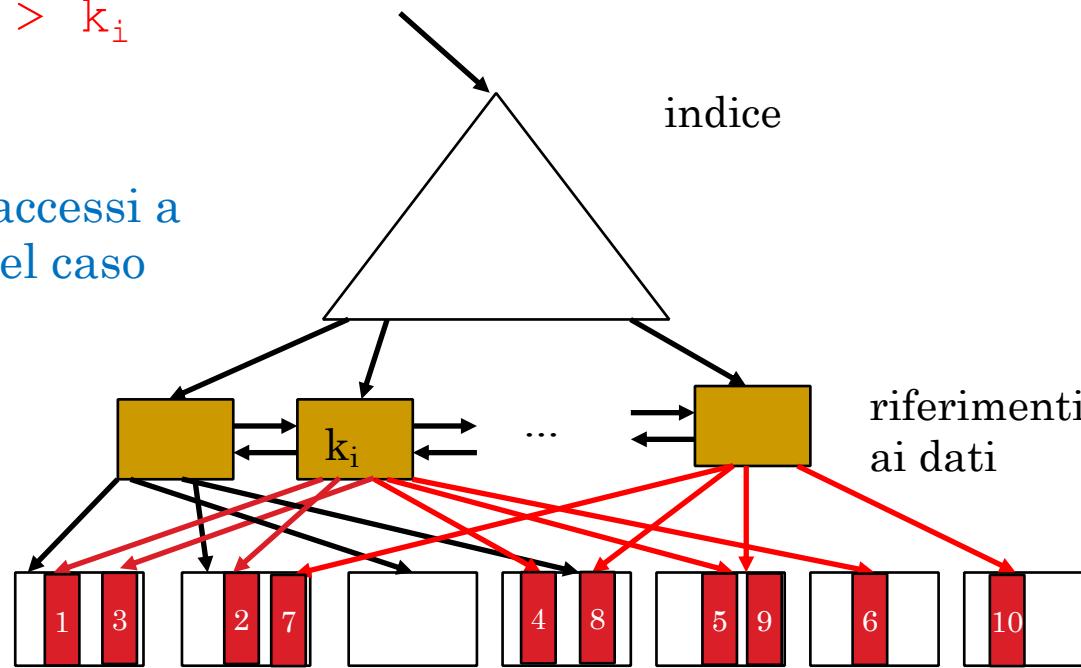
Lo stesso blocco può essere acceduto più volte  
(sia per uguaglianza che per intervallo)

```
SELECT *  
FROM R  
WHERE A > ki
```

Al più tanti accessi quante sono le tuple risultato

Costo:  $h + 10$  accessi a  
blocchi dati (nel caso  
peggiore)

I numeri indicano  
l'ordine di accesso  
alle tuple (rosse) nei  
blocchi



## ACCESSO ALLE RELAZIONI DI BASE - COSTI

- Se viene utilizzato un cammino di accesso basato su **indice non clusterizzato**, è possibile (e importante!) **evitare di visitare lo stesso blocco più volte per lo stesso valore della chiave di ricerca**
  - si individuano le foglie dell'indice che soddisfano la condizione
  - si ordinano i rid dei record dei dati da reperire, in modo che i rid di record nello stesso blocco siano vicini
  - si accede ai record corrispondenti in ordine
- in tal modo si accede ad ogni blocco un'unica volta

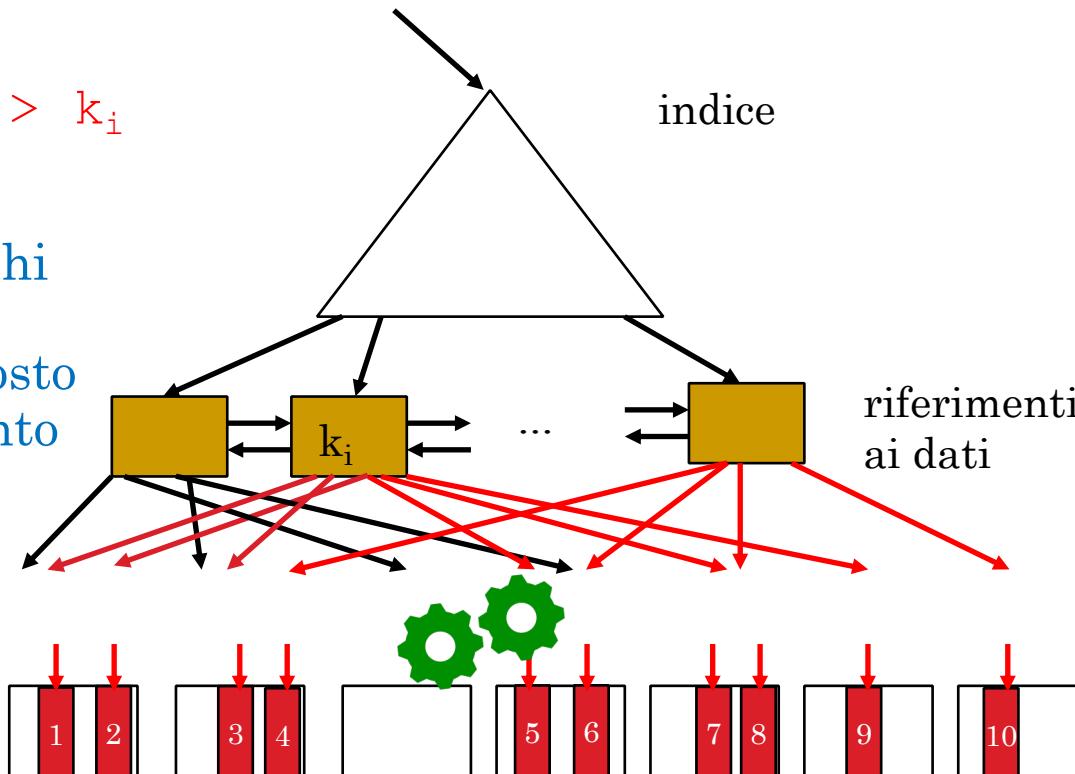
# ESEMPIO – INDICE ORDINATO NON CLUSTERIZZATO

$$\sigma_{A=k_i}(R)$$

Lo stesso blocco può essere acceduto più volte  
(sia per uguaglianza che per intervallo)

```
SELECT *  
FROM R  
WHERE A > ki
```

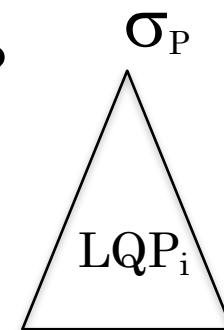
Costo:  $h + 6$   
accessi a blocchi  
dati (nel caso  
peggiore) (+ costo  
per ordinamento  
RID)



I numeri indicano  
l'ordine di accesso  
alle tuple (rosse) nei  
blocchi

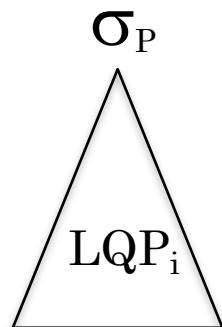
# ELABORAZIONE DEI NODI INTERNI DEL PIANO - SELEZIONE

- Finora abbiamo discusso la selezione integrata all'accesso ai dati
- Come eseguire la selezione se viene applicata a un risultato intermedio?

$$\sigma_p \\ \downarrow \\ R$$


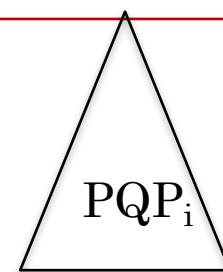
# ELABORAZIONE DEI NODI INTERNI DEL PIANO - SELEZIONE

- In questo caso la selezione può essere implementata solo con una scansione sequenziale della relazione in input
- Non è mai possibile utilizzare un indice su un risultato intermedio
  - Indice sempre costruito su relazione di base



Livello fisico:  
qualsiasi

SEQ SCAN R  
Filter P

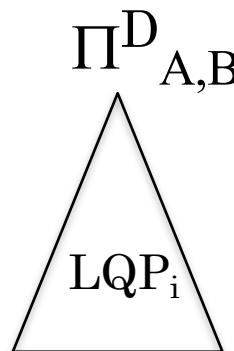


# ELABORAZIONE DEI NODI INTERNI DEL PIANO - PROIEZIONE

- La proiezione in SQL può prevedere
  - L'eliminazione di duplicati (proiezione algebrica)  
SELECT DISTINCT  
 $\Pi$  operatore logico
  - Il mantenimento dei duplicati  
SELECT  
 $\Pi^D$  operatore logico

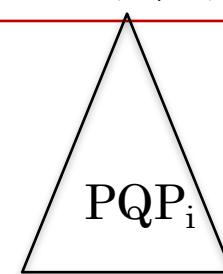
# PROIEZIONE SENZA ELIMINAZIONE DEI DUPLICATI

- Per implementare la **proiezione senza eliminazione dei duplicati**, per ogni tupla in input, è necessario rimuovere gli attributi che non compaiono nella lista di proiezione
- Approccio **iterativo** (scansione sequenziale della relazione in input), non servono gli indici
- Costo pari al numero di blocchi della relazione in input



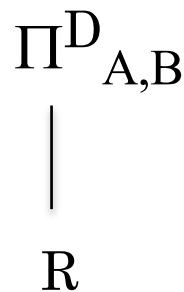
Livello fisico:  
qualsiasi

PROJECTION  
List(A, B)



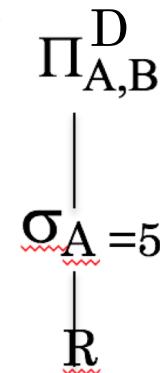
# PROIEZIONE SENZA ELIMINAZIONE DEI DUPLICATI

- Può essere integrata con accesso alla relazione di base (o in generale con l'operatore figlio), se la query lo ammette



Livello fisico:  
qualunque

SEQ SCAN R  
Proj (A, B)



Livello fisico:  
qualunque

SEQ SCAN R  
Filter (A=5)  
Proj (A, B)

Livello fisico:  
 $I_R(A)$

INDEX SCAN  
( $I_R(A)$ , A = 5)  
Proj (A, B)

# PROIEZIONE CON ELIMINAZIONE DEI DUPLICATI

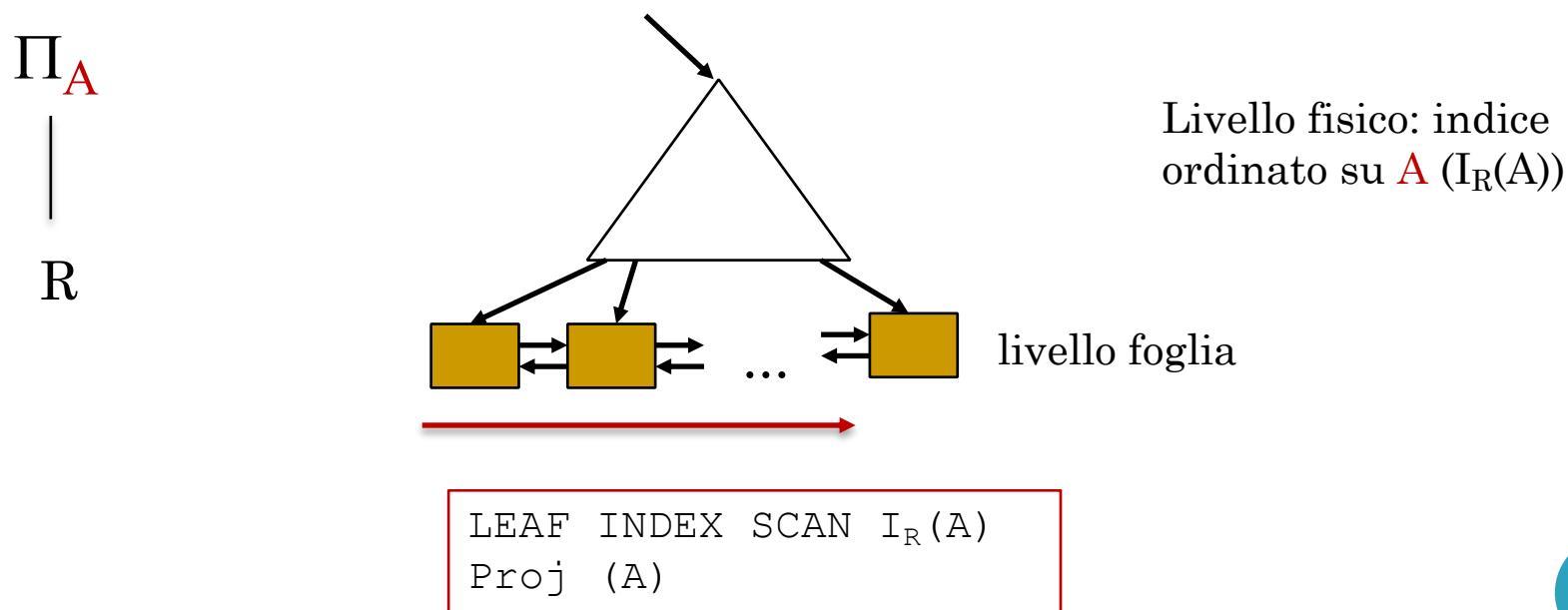
- Per implementare la proiezione con eliminazione di duplicati è necessario:
  - rimuovere gli attributi che non compaiono nella proiezione
  - eliminare i duplicati
- il primo passo si può eseguire come per proiezione senza eliminazione dei duplicati
- il secondo passo è quello più difficile/costoso
- tre algoritmi possibili approcci, che sfruttano
  - ordinamento (approccio di partizionamento)
  - indici (approccio index-based)
  - hashing (approccio di partizionamento)

# PROIEZIONE CON ELIMINAZIONE DEI DUPLICATI - ORDINAMENTO

1. Si **accede sequenzialmente** a R per ottenere un insieme di tuple che contengono solo gli attributi desiderati
2. Si **ordina** questo insieme di tuple rispetto a tutti gli attributi di proiezione (vedremo in seguito come si implementa l'ordinamento)
3. Si **scandisce** il risultato ordinato, eliminando le tuple duplicate (che sono adiacenti)
  - Si può ottimizzare integrando 3 in 2

# PROIEZIONE CON ELIMINAZIONE DEI DUPLICATI – APPROCCIO INDEX-BASED

- Per proiezioni su **relazioni di base**, se esiste un indice ordinato con chiave di ricerca uguale alla lista di proiezione, si può accedere a **tutte le foglie dell'indice** invece che al file dei dati



# ELABORAZIONE DEI NODI INTERNI DEL PIANO - ORDINAMENTO

- È spesso necessario ordinare i dati
  - perché l'interrogazione SQL da eseguire contiene una clausola ORDER BY
  - perché avere i dati ordinati permette di eseguire altre operazioni in modo più efficiente
    - come abbiamo visto , proiezione con eliminazione dei duplicati
    - come vedremo, join, raggruppamento

# ORDINAMENTO

- Per implementare l'ordinamento, non si possono usare algoritmi di ordinamento classici perché i dati da ordinare sono troppi per stare in memoria principale
- Due approcci principali
  - merge sort esterno a due fasi
  - uso di B+ tree

# ORDINAMENTO – MERGE SORT ESTERNO A DUE FASI

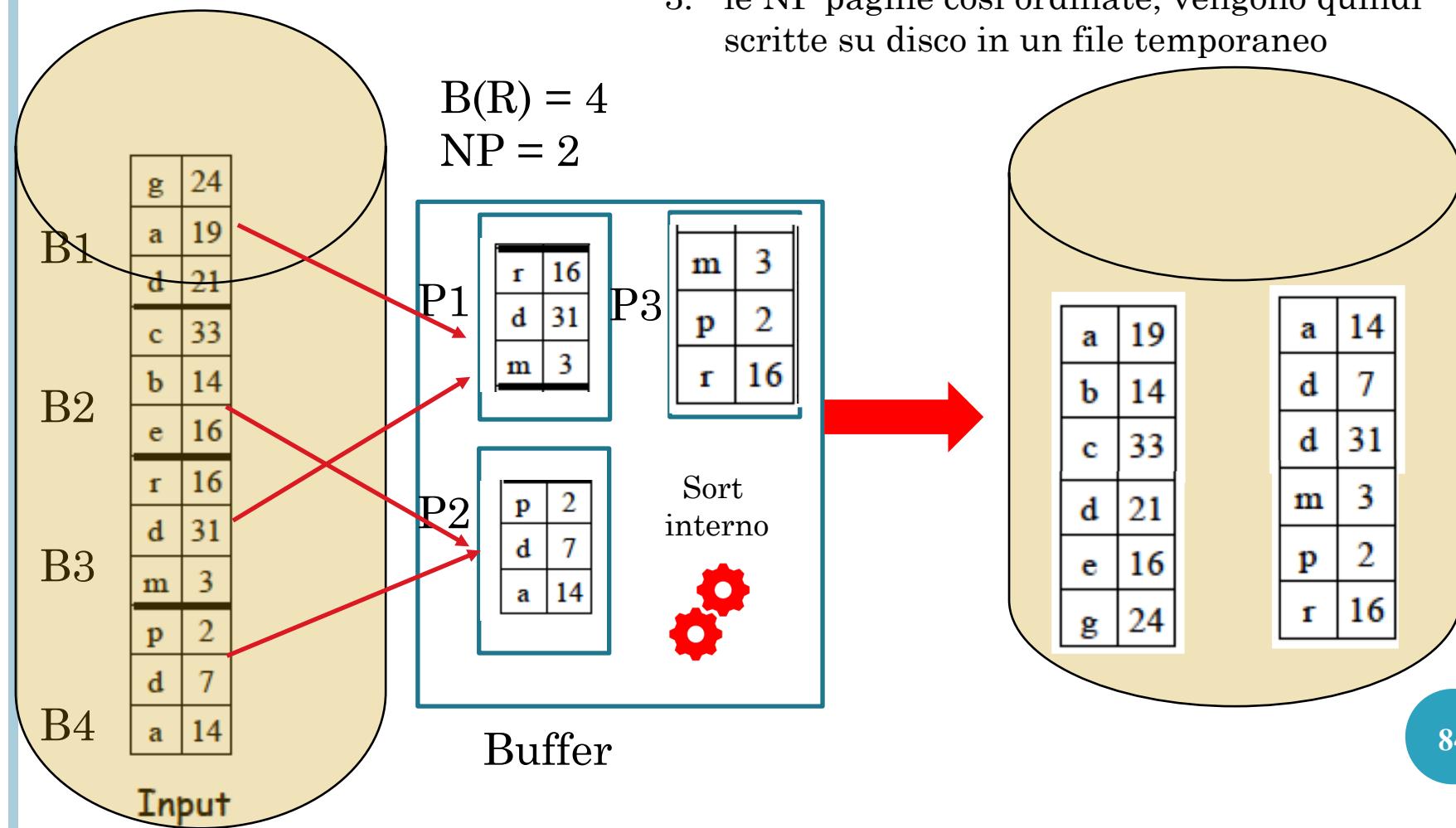
- L'algoritmo più comunemente utilizzato dai DBMS è quello detto di **merge sort esterno a 2 fasi**
- Supponiamo di dover ordinare una relazione R in input che consiste di un file di **B(R)** blocchi
- Supponiamo di avere a disposizione solo **NP+1 < B(R)** pagine di buffer in memoria centrale
- Si ordinano separatamente porzioni di dati che stanno in memoria e poi se ne effettua il merge
-

# ORDINAMENTO – MERGE SORT ESTERNO A DUE

## FASI, FASE 1

Fase 1 (sort interno):

1. si leggono NP pagine alla volta dal file dei dati al buffer
2. i record delle NP pagine vengono ordinati facendo uso di un algoritmo di sort interno (es. Quicksort)
3. le NP pagine così ordinate, vengono quindi scritte su disco in un file temporaneo

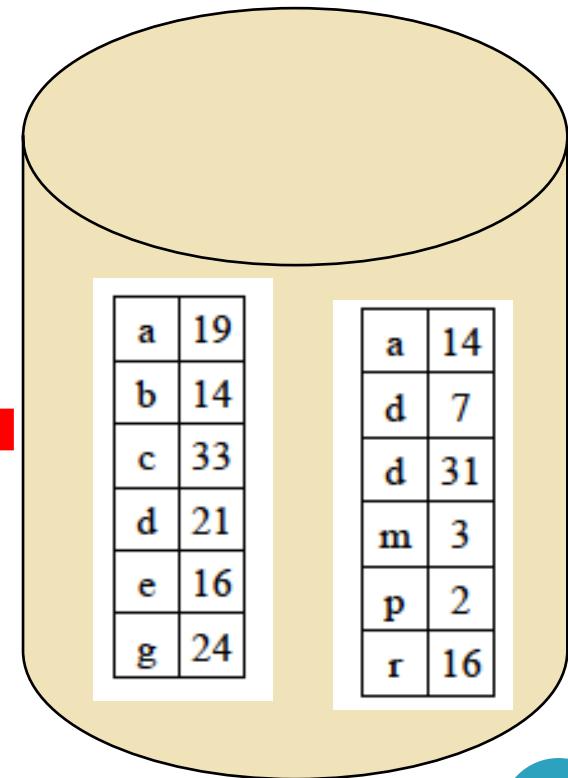
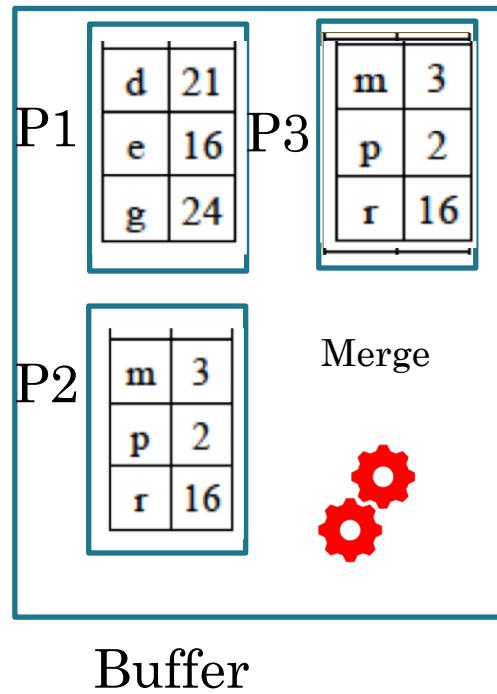
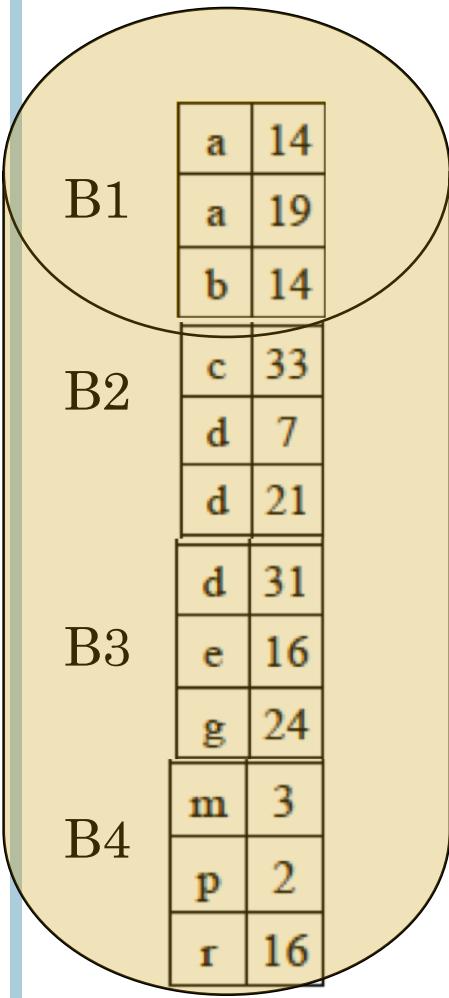


# ORDINAMENTO – MERGE SORT ESTERNO A DUE FASI

- **Fase 2:**
  - si effettua il merge delle sottoliste ordinate
  - in memoria si carica il primo blocco di ogni sottolista, per un totale di  $NP$  pagine di buffer
  - Si effettua quindi un ciclo fino a che non si sono esaurite tutte le sottoliste ordinate in cui:
    - trovo il valore più piccolo tra quelli contenuti nelle  $NP$  pagine in memoria
    - lo sposto nel blocco di output (il buffer contiene  $NP+1$  pagine)
    - se il blocco di output è pieno lo scarico su disco e lo reinizializzo
    - se il blocco da cui ho estratto l'elemento è vuoto, leggo il prossimo blocco di quella sottolista
    - se quella sottolista è terminata, lascio vuoto quel blocco di buffer

# ORDINAMENTO – MERGE SORT ESTERNO A DUE FASI, FASE 2

Fase 2 (merge): le liste di pagine ordinate generate dalla Fase 1 vengono fuse, fino a produrre un'unica lista di pagine ordinate

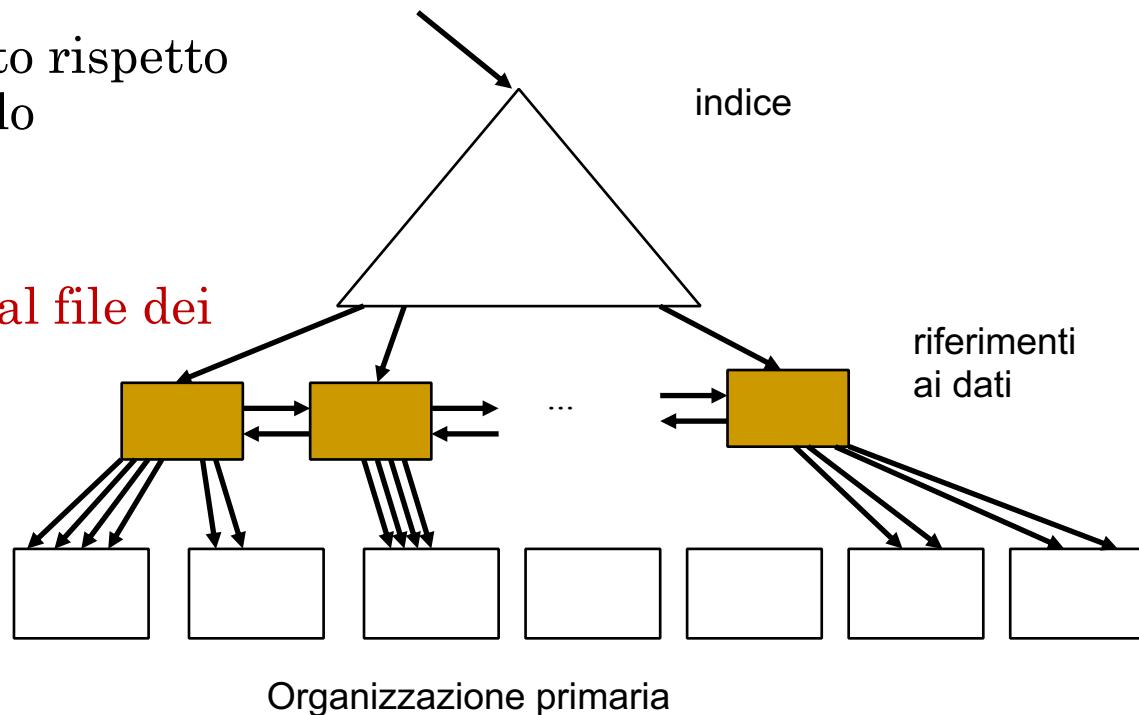


## ORDINAMENTO – USO DI B+ TREE

- Se la relazione da ordinare ha un indice di tipo B+ tree sugli attributi rispetto ai quali vogliamo ordinare si può pensare di effettuare l’ordinamento attraversando le pagine foglia dell’indice
  - se l’indice è clusterizzato è una buona idea
  - se l’indice non è clusterizzato può essere una pessima idea

# ORDINAMENTO – USO DI B+ TREE

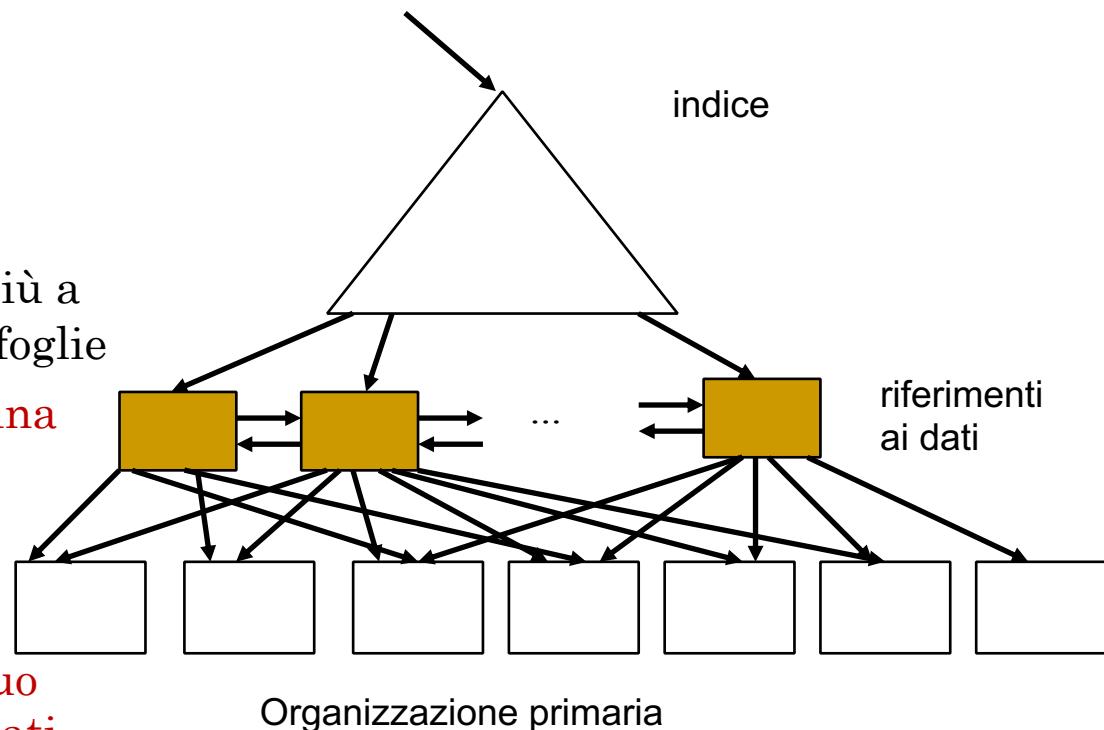
- ORDER BY A
- B+ tree clusterizzato su A
- File dei dati ordinato rispetto ad A, basta accederlo sequenzialmente
- Costo: B(R) accessi al file dei dati



# ORDINAMENTO – USO DI B+ TREE

- ORDER BY A
- B+ tree non clusterizzato su A
- File dei dati non ordinato rispetto ad A

- Usiamo indice:
  - dalla radice alla foglia più a sinistra (h), poi tutte le foglie
  - **torno su una stessa pagina dei dati più volte**



- Costo
  - **nel caso peggiore, effettuo tanti accessi al file dei dati quanti sono i record!**
  - **Usando ordinamento dei RID, come discusso per la selezione si reduce il costo**

## ORDINAMENTO – USO DI B+-TREE

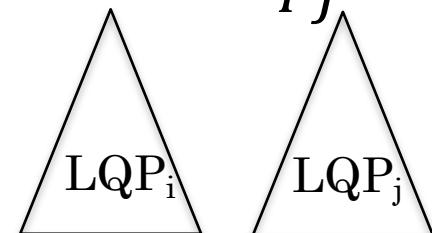
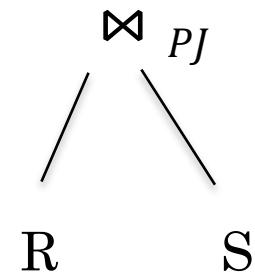
- È però possibile (e importante!) evitare di visitare lo stesso blocco più volte per lo stesso valore della chiave di ricerca
  - si trovano le foglie dell'indice che soddisfano la condizione
  - si ordinano i rid dei record dei dati da reperire, in modo che i rid di record nello stesso blocco siano vicini
  - si accede ai record corrispondenti in ordine
- in tal modo si accede ad ogni blocco un'unica volta

# ELABORAZIONE DEI NODI INTERNI DEL PIANO - JOIN

- L'operazione di join è una operazione molto costosa: date due relazioni **R** e **S**, contenenti  $T(R)$  e  $T(S)$  tuple, rispettivamente, richiede  $T(R) * T(S)$  confronti!
- Esistono moltissime implementazioni del Join, che mirano a sfruttare al meglio le risorse del sistema e le (eventuali) proprietà degli insiemi di tuple in ingresso per **evitare di eseguire tutti i possibili  $T(R) * T(S)$  confronti**
  - Nested Loop Join (semplice e a blocchi)
  - Index Nested Join
  - Merge Join
  - Hash Join

# JOIN

- $R \bowtie_{PJ} S$ 
  - R: **relazione esterna** (outer relation, per convenzione a sinistra)
  - S: **relazione interna** (inner relation)
  - $PJ = R.A \theta S.B$
- Nel seguito, quando non specificato altrimenti, assumiamo che R ed S siamo relazioni di base
- Ricordiamo però che in generale l'input può deri~~re~~<sup>PJ</sup> dall'applicazione di altri operatori



# JOIN – ITERAZIONE SEMPLICE (NESTED LOOP JOIN)

- Si accede ad una tupla di R (outer relation) e si confronta con ogni tupla di S (inner relation)
- Costo:  $B(R) + T(R) * B(S)$

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 } }

R	A	B
22	a	
87	s	
45	h	
32	b	

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

PJ: R.A = S.A



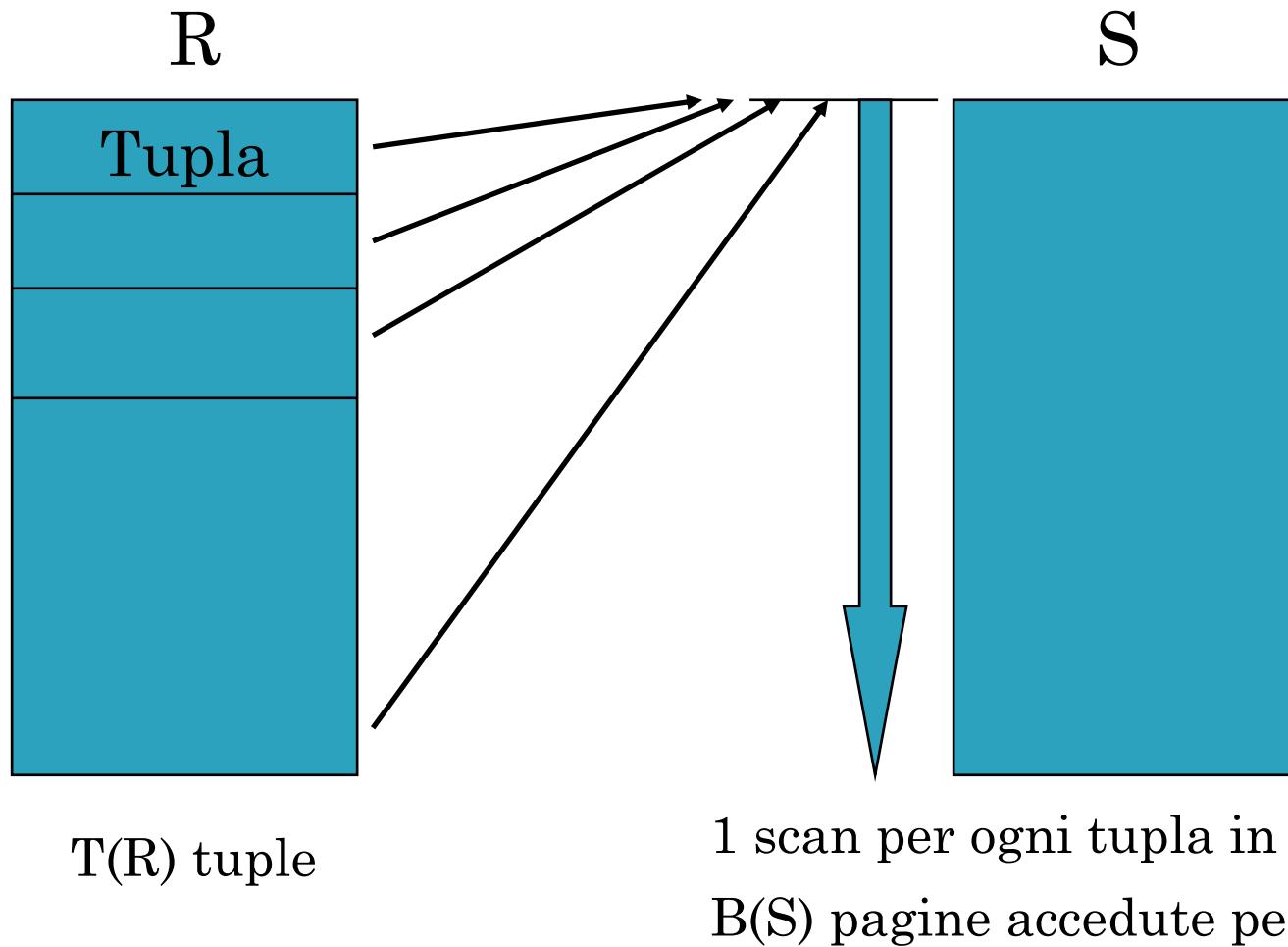
	A	C	D	B
22	z	8		a
22	s	7		a
87	s	9		s
45	k	4		h
45	h	5		h
32	c	3		b
32	g	6		b

L'ordine con cui vengono generate le tuple del risultato coincide con l'ordine delle tuple nella relazione esterna

## JOIN – ITERAZIONE SEMPLICE

- Conveniente se la relazione inner può essere mantenuta in MM, in questo caso
- Costo:  $B(R) + B(S)$
- Quindi: *conviene considerare come relazione outer la relazione più grande*

## JOIN – ITERAZIONE SEMPLICE



# JOIN – ITERAZIONE ORIENTATA AI BLOCCHI (BLOCK NESTED LOOP)

- Variante del Nested Loop molto usata
- Rinunciando a preservare l'ordine della relazione esterna, risulta più efficiente in quanto esegue il join di tutte le tuple in memoria prima di richiedere nuove pagine della relazione interna
- Costo:  $B(R) + B(R)*B(S)$

Per ogni pagina  $p_R$  di R:

{ Per ogni pagina  $p_S$  in S:

{ esegui il join di tutte le tuple in  $p_R$  e  $p_S$  }

R	A	B
22	a	
87	s	
87	h	
92	b	

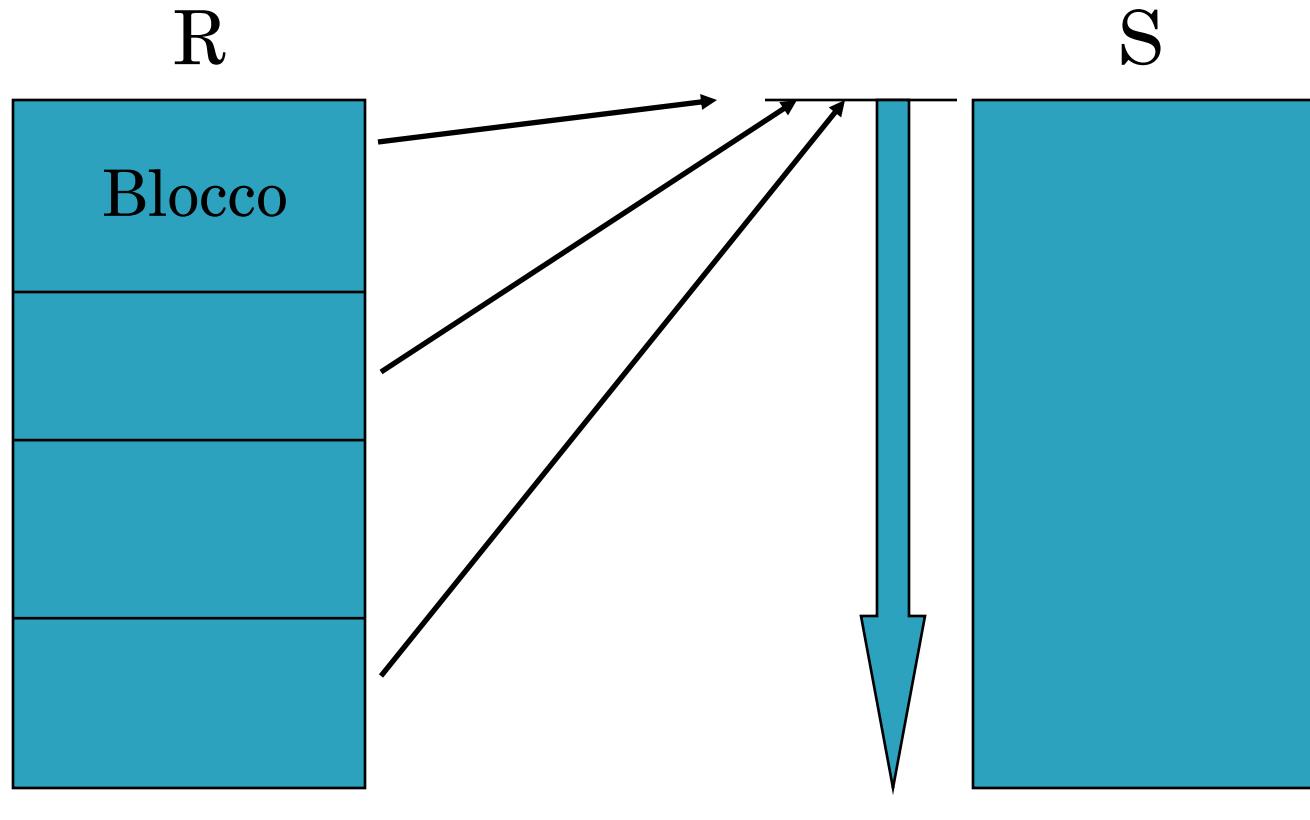
S	A	C	D
22	z	8	
87	c	8	
22	s	7	
87	f	9	



	A	C	D	B
22	z	8	a	
87	c	8	s	
22	s	7	a	
87	f	9	s	
87	c	8	h	
87	f	9	h	

L'ordine con cui vengono generate le tuple del risultato **non** coincide con l'ordine delle tuple nella relazione esterna

# JOIN – ITERAZIONE ORIENTATA AI BLOCCHI



$B(R)$  blocchi

1 scan per ogni blocco di R  
 $B(S)$  pagine accedute  
per scan

## JOIN – INDEX NESTED LOOP

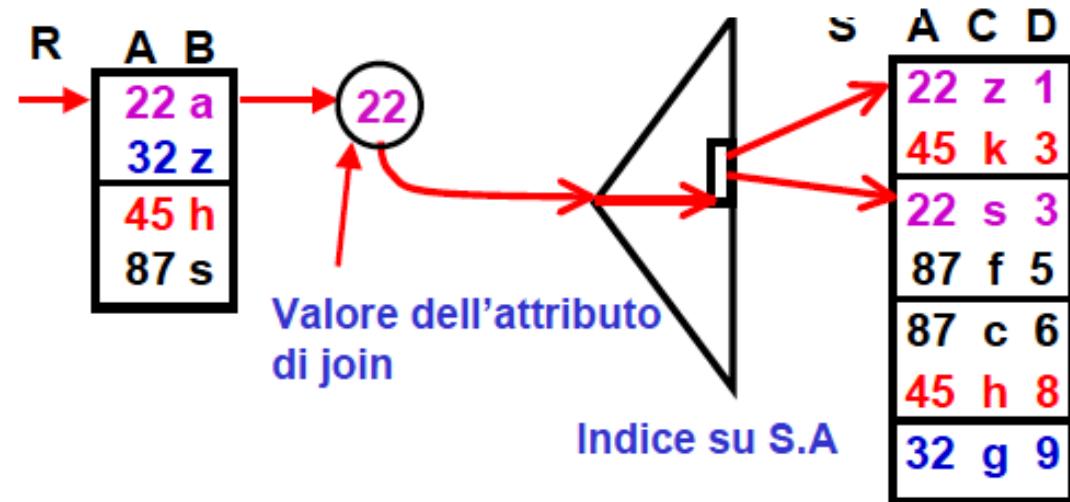
- Supponiamo che la relazione interna sia una relazione di base ed esista un indice sull'attributo A della relazione S che supporta ricerche rispetto a una condizione di join  $PJ = R.A \theta S.B$
- Consideriamo la strategia di iterazione semplice
- Data una tupla tr di R non è più necessario scandire l'intera relazione S, ma è sufficiente eseguire una ricerca sull'indice di S con condizione  $A \theta tr.A$
- Costo:  $B(R) + T(R)^*$  Costo di ricerca con accesso  $(I_S(A), A \theta tr.A)$

# JOIN – INDEX NESTED LOOP

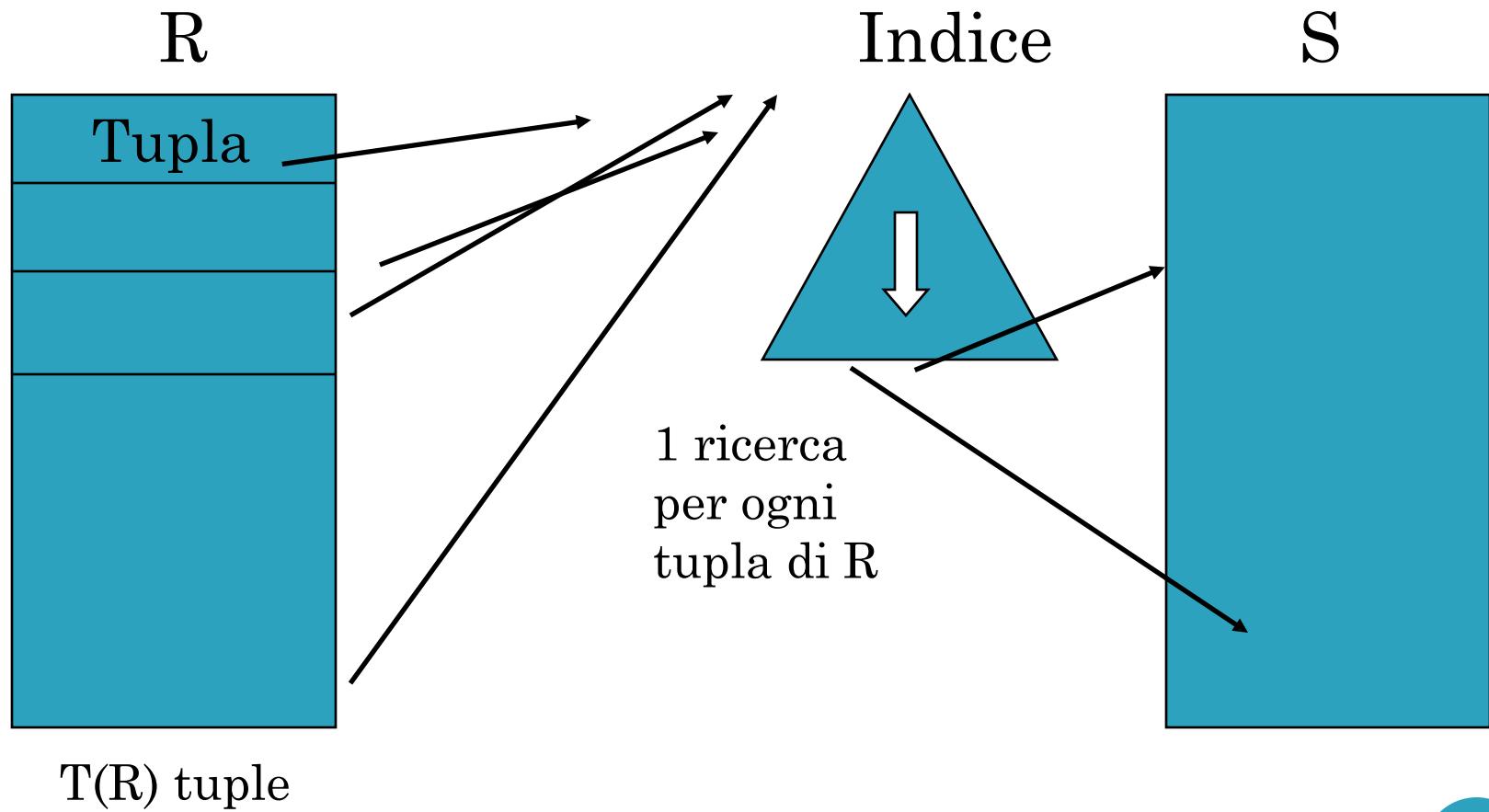
per ogni tupla  $t_R$  di  $R$ :

{ accedi  $S$  con cammino di accesso  $(I_S(A), A = t_R.A)$ ;

per ogni tupla  $t_S$  restituita dall'indice,  
aggiungi  $(t_R, t_S)$  al risultato }



## JOIN – USO DI INDICI



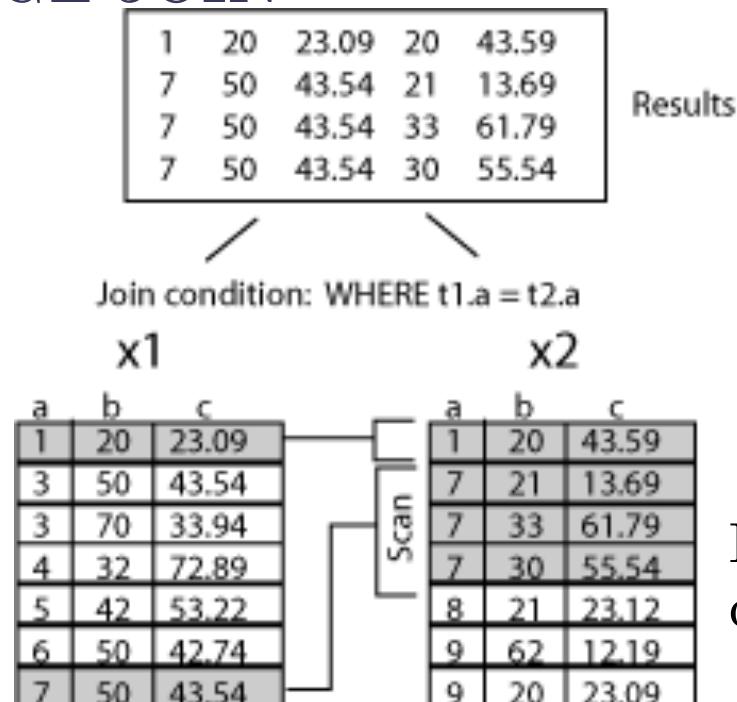
## JOIN – MERGE JOIN

- Il Merge Join è applicabile quando le relazioni in input sono ordinate rispetto all’attributo di join
- L’algoritmo sfrutta il fatto che entrambi gli input sono ordinati per evitare di fare inutili confronti
  - Idea simile all’**algoritmo di merge sort**
- Questo porta il numero di letture nell’ordine di **B(R) + B(S)** se si accede sequenzialmente alle due relazioni
- Il Merge Join è in generale usato solo per prediciati di **join di uguaglianza** (equi-join), perché negli altri casi i suoi vantaggi si riducono considerevolmente

# JOIN – MERGE JOIN

- Algoritmo di merge analogo a quanto visto ad ASD per il merge sort
  - l'operazione di merge-join richiede di associare inizialmente un puntatore ad ogni relazione
  - i puntatori inizialmente puntano alla prima tupla di ogni relazione
  - i puntatori vengono fatti avanzare in relazione all'esito dei controlli tra tuple delle due relazioni in join
  - poiché le tuple sono ordinate in base all'attributo di join ogni tupla (e quindi ogni blocco) viene letta esattamente una volta

# JOIN – MERGE JOIN



Relazioni dopo ordinamento

t1

a	b	c
4	32	72.89
7	50	43.54
5	42	53.22
3	70	33.94
6	50	42.74
3	50	43.54
1	20	23.09

Outer table

t2

a	b	c
7	21	13.69
9	62	12.19
9	20	23.09
8	21	23.12
1	20	43.59
7	33	61.79
7	30	55.54

Inner table

Relazioni iniziali

## JOIN – HASH JOIN

- Il vantaggio del Merge Join è che viene sostanzialmente ridotto il numero di confronti tra i record delle due relazioni, grazie all'**ordinamento**
- Un altro modo per ridurre il numero di confronti è partizionare i record **usando una funzione hash**
- L'algoritmo di Hash Join, **applicabile solo in caso di equi-join**, non richiede né la presenza di indici né input ordinati

# JOIN- HASH JOIN

- Funzione hash  $H$  definita sugli attributi di join delle due relazioni
  - $R.A \text{ e } S.B \text{ se } PJ = R.A \theta S.B$
- Se  $t_R$  è tupla di  $R$  e  $t_S$  è tupla di  $S$ , allora
  - se  $t_R.A = t_S.B$  allora  $H(t_R.A) = H(t_S.B)$
  - viceversa, se  $H(t_R.A) \neq H(t_S.B)$ , allora sicuramente vale  $t_R.A \neq t_S.B$
- Quindi se due tuple sono associate a due valori diversi della funzione hash (applicata agli attributi di join) allora sicuramente le tuple non possono essere in join
- Idea di base algoritmo
  - $R$  e  $S$  vengono partizionate sulla base dei valori di  $H$
  - la ricerca delle tuple in join avviene solo tra partizioni relative allo stesso valore di  $H$
- Il costo risulta essere dell'ordine di  $B(R) + B(S)$

# JOIN – CONDIZIONI PIÙ GENERALI

- **Uguaglianza su più di un attributo**
  - si può usare un indice su tutti gli attributi o uno qualsiasi degli attributi e poi filtrare il risultato rispetto alle condizioni rimanenti (analogo a selezioni composte)
  - in merge e hash join, si deve ordinare/partizionare su tutti gli attributi di join
- **Theta-join con disuguaglianza**
  - si può usare l'index nested loop se l'indice è ordinato
  - merge e hash join non sono applicabili

## ALTRÉ OPERAZIONI

- L'intersezione e il prodotto Cartesiano sono casi speciali del join
- Per operazioni insiemistiche con eliminazione dei duplicati e operazioni di raggruppamento si usano tecniche analoghe a quelle viste per proiezione con eliminazione dei duplicati
- *Provare per esercizio a definire degli algoritmi di realizzazione degli operatori sopra citati, partendo dagli algoritmi discussi in precedenza*

# OTTIMIZZAZIONE FISICA

Algoritmi per l'elaborazione complessiva del piano fisico

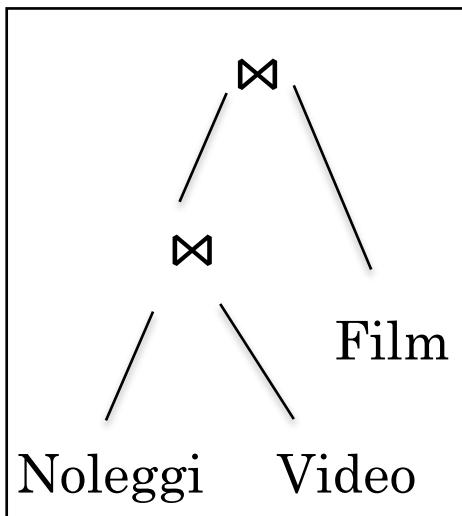
# ELABORAZIONE DEL PIANO FISICO - MATERIALIZZAZIONE

- Un piano di esecuzione fisico è un albero
- Abbiamo visto algoritmi per implementare i singoli nodi
  - Accesso alle relazioni di base
  - Selezione
  - Proiezione
  - Ordinamento
  - Join
- Come si combinano insieme i risultati delle varie operazioni?

# ELABORAZIONE DEL PIANO FISICO - MATERIALIZZAZIONE

- Un semplice modo di eseguire un piano di accesso composto da diversi operatori consiste nel procedere **bottom-up**, secondo il seguente schema:
  - Si accede alle relazioni di base, eventualmente integrando l'accesso con le operazioni di selezione e proiezione, e si memorizzano tali risultati in relazioni temporanee
  - Si procede quindi in modo analogo per gli operatori del livello sovrastante, fino ad arrivare alla radice
- Tale modo di procedere è detto “**valutazione per materializzazione**” ed è altamente inefficiente
  - comporta la creazione, scrittura e lettura di molte relazioni temporanee
  - se la dimensione dei risultati intermedi eccede lo spazio disponibile in memoria centrale, i risultati devono essere scritti/ letti su/da disco
  - costoso

# ESEMPIO



```
SELECT *  
FROM Noleggi N, Video V, Film F  
WHERE N.colloc = V.colloc AND  
V.titolo = F.titolo AND  
V.regista = F.regista
```

valutazione per  
materializzazione - costosa

NESTED LOOP JOIN  
Filter Video.titolo = Film.titolo  
AND Video.regista = Film.regista

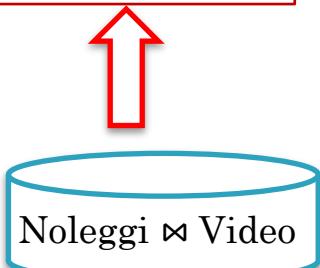
NESTED LOOP JOIN  
Filter Noleggi.colloc =  
Video.colloc

Noleggi  $\bowtie$  Video  
deve essere  
memorizzato nel  
buffer se ci sta,  
altrimenti su  
disco

SEQ SCAN Noleggi

SEQ SCAN Video

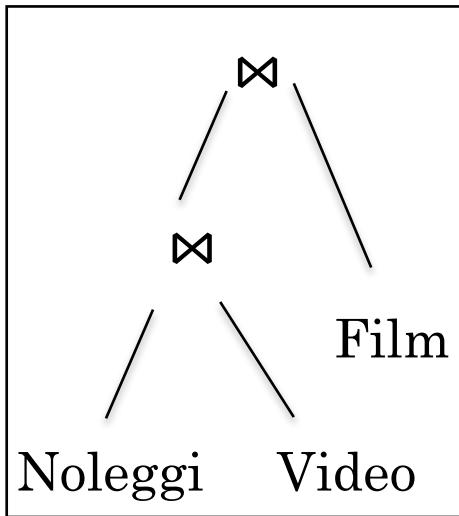
SEQ SCAN Film



# ELABORAZIONE DEL PIANO FISICO - PIPELINE

- Un modo alternativo di eseguire un piano di accesso è quello di eseguire più **operatori in pipeline**, ovvero non aspettare che termini l'esecuzione di un operatore per iniziare l'esecuzione di un altro

# ESEMPIO



```
SELECT *
FROM Noleggi N, Video V, Film F
WHERE N.colloc = V.colloc AND
      V.titolo = F.titolo AND
      V.regista = F.regista
```

valutazione in pipeline –  
quando possibile, più efficiente

$t_1 \bowtie$  Film  
 $t_2 \bowtie$  Film  
 $t_3 \bowtie$  Film

NESTED LOOP JOIN  
Filter Video.titolo = Film.titolo  
AND Video.regista = Film.regista

NESTED LOOP JOIN  
Filter Noleggi.colloc =  
Video.colloc

$t_1$   
 $t_2$   
 $t_3$

Noleggi  $\bowtie$  Video  
non viene  
memorizzato

SEQ SCAN Noleggi

SEQ SCAN Video

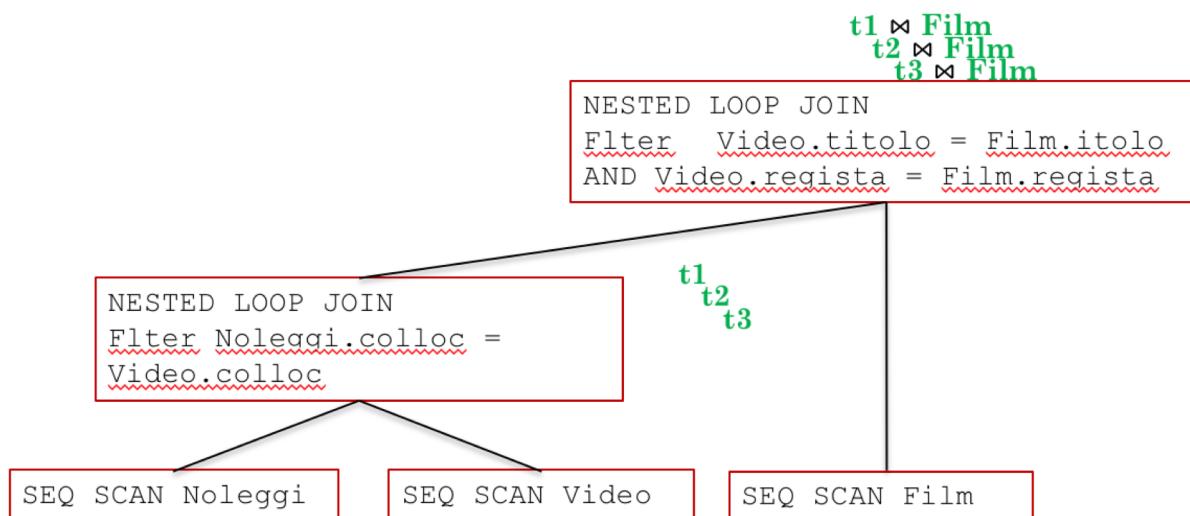
SEQ SCAN Film

# ELABORAZIONE DEL PIANO FISICO - PIPELINE

- Si inizia a eseguire il primo Join (**nodo 3**)
- Appena viene prodotta **la prima tupla t** dell'output, questa viene passata in input al secondo Join (**nodo 5**), che può quindi iniziare la ricerca di matching su Film (**che deve essere materializzata**) e produrre la prima tupla del risultato finale della query
- La valutazione prosegue cercando eventuali altri match per la tupla prodotta dal primo Join
- Quando è terminata la scansione della relazione interna (Film), il secondo Join richiede al primo Join di produrre un'altra tupla

# ELABORAZIONE DEL PIANO FISICO - PIPELINE

- Osserviamo che, nel caso del join, il pipeline si può applicare solo se l'operatore fisico esegue un ciclo sulla relazione outer
  - Nested loop semplice
  - Nested loop a blocchi
  - Index nested loop
- In questo caso, il pipeline viene applicato alla relazione outer mentre la relazione inner, utilizzata ad ogni ciclo per i confronti, deve essere materializzata



# ELABORAZIONE DEL PIANO FISICO - PIPELINE

- Il pipelining permette di risparmiare il costo di scrivere il risultato intermedio e di rileggerlo successivamente
- Poiché tale costo può essere significativo si preferisce il pipelining alla materializzazione se l'algoritmo per l'operatore lo permette
- Un piano può contenere alcuni operatori valutati per materializzazione (input materializzato) e altri eseguiti in pipeline (input una tupla alla volta)

# SINTESI ELABORAZIONE DEGLI OPERATORI RELAZIONALI TRAMITE OPERATORI FISICI

\_\_\_\_\_ = ammette elaborazione in pipeline (nel join su relazione outer)

Operatore fisico			
Operatore logico	Iterazione	Uso di indici	Partizionamento
ordinamento		X	merge sort a due fasi
$\sigma$	X	X	
$\pi^D$	X		
$\pi$ (con eliminazione duplicati)		X	Ordinamento (uso di funzione hash, non presentato)
$\bowtie$	<u>nested loop</u> <u>semplice e a blocchi</u>	<u>index join</u>	merge join hash join

# OTTIMIZZAZIONE FISICA

Stima del costo di un piano di esecuzione

# STIMA DEL COSTO DI UN PIANO DI ESECUZIONE

- Approccio bottom-up
- Per stimare il costo di un piano di esecuzione
  - Si stima il **costo di accesso** alle relazioni di base
  - Per ogni nodo nell'albero:
    - Si stima il **costo** di effettuare l'operazione corrispondente
    - Si stima la **dimensione del risultato** (che sarà l'input di operatori successivi) e si determina se è ordinato
  - Si sommano poi i costi parziali ottenuti

# STATISTICHE

- Per la determinazione della stima dei costi delle varie operazioni e della dimensione del risultato, si utilizzano **dati contenuti nei cataloghi di sistema**
- Per ogni relazione R:
  - $T(R)$  numero di tuple nella relazione R
  - $B(R)$  numero di blocchi della relazione R
  - $S(R)$  dimensione di una tupla della relazione R in bytes (per tuple a lunghezza fissa, altrimenti si usano valori medi)
  - $S(A,R)$  dimensione dell'attributo A nella relazione R
  - $V(A,R)$  numero di valori distinti per l'attributo A nella relazione R
  - $\text{Max}(A,R)$  e  $\text{Min}(A,R)$  valori minimo e massimo dell'attributo A nella relazione R

# STATISTICHE

- Per ogni indice I:
  - $K(I)$  numero di entrate (valori di chiave) dell'indice I
  - $L(I)$  numero di pagine foglia dell'indice I (se ordinato)
  - $H(I)$  altezza dell'indice I (se ordinato)
- Tali statistiche sono aggiornate alla creazione di un indice e **in seguito solo periodicamente**
  - aggiornarle dopo ogni modifica ai dati è troppo costoso
  - Poiché le stime dei costi sono approssimate comunque si accetta che tali valori non siano completamente accurati
- Molti DBMS prevedono un comando (**ANALYZE** in PostgreSQL) per richiedere esplicitamente il loro aggiornamento

# STIMA DELLA DIMENSIONE DEL RISULTATO DEGLI OPERATORI FISICI

- Nel seguito a titolo di esempio vedremo il caso della selezione

# STIMA DELLA DIMENSIONE DEL RISULTATO DELLA SELEZIONE

- Il numero di tuple restituito da una selezione  $\sigma_P(R)$  dipende da quante tuple di  $R$  soddisfano il predicato  $P$
- Fattore di selettività  $F(P)$ 
  - probabilità che una tupla di  $R$  soddisfi il predicato  $P$
- Tanto più il fattore di selettività è minore di uno, tanto più è selettivo il predicato a cui si riferisce (minore è il numero di tuple che soddisfano  $P$ )
- Si stima poi che  $\sigma_P(R)$  selezioni un numero di tuple pari a  $T(R) * F(P)$

# STIMA DELLA DIMENSIONE DEL RISULTATO DELLA SELEZIONE

- Come si stima il fattore di selettività?
- Assunzione: uniformità di distribuzione dei valori di ogni attributo, cioè si assume che ogni valore appaia con la stessa probabilità
- Stima dipende dalla condizione di selezione

- ESEMPIO
- Condizione di selezione  $A = v$
- $F(A=v)$ : probabilità che una tupla in input alla selezione soddisfi la condizione  $A = v$

- *Casi possibili*: tutti i possibili valori per A in R  $V(A,R)$
- *Casi favorevoli*: 1 (solo v)
- Quindi  $F(A=v) = 1 / V(A,R)$

$$\sigma_{A=v} \mid R$$

# STIMA DELLA DIMENSIONE DEL RISULTATO DELLA SELEZIONE

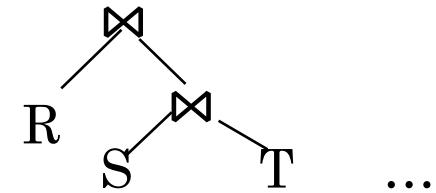
- $F(A=v) = 1/V(A,R)$
- $F(A \text{ IN } (v_1, v_2, \dots, v_N)) = N * F(A=v)$
- $F(A>v) = (\text{Max}(A,R) - v)/(\text{Max}(A,R) - \text{Min}(A,R))$
- $F(A<v) = (v - \text{Min}(A,R))/(\text{Max}(A,R) - \text{Min}(A,R))$
- $F(A_1 = A_2) = 1 / \text{MAX } (V(A_1,R), V(A_2,R))$
- $F(C_1 \text{ AND } C_2) = F(C_1) * F(C_2)$
- $F(C_1 \text{ OR } C_2) = F(C_1) + F(C_2) - F(C_1) * F(C_2)$
- $F(\text{NOT } C) = 1 - F(C)$

# OTTIMIZZAZIONE FISICA

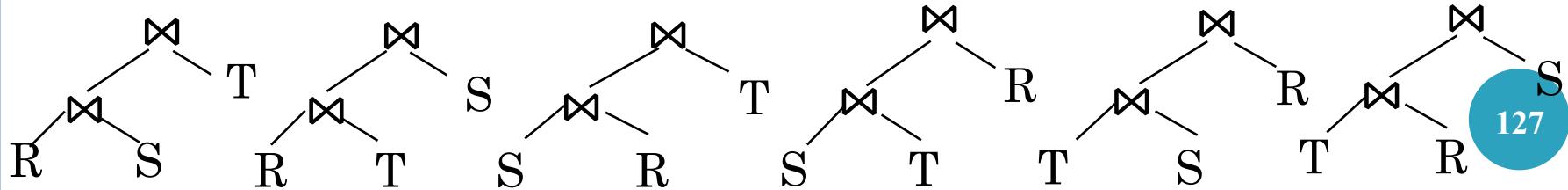
Identificazione dello spazio di ricerca dei piani

# SPAZIO DI RICERCA DEI PIANI

- Data un'interrogazione un ottimizzatore dovrebbe enumerare tutti i possibili piani per eseguirla e valutarne il costo, per poi selezionare quello di costo minimo
- Problema complesso e costoso
- Esempio
  - $R \bowtie S \bowtie T$
  - Per  $N$  join, esistono almeno  $N!$  ordinamenti del join
  - Per ogni ordinamento, diversi piani di esecuzione in relazione a: scelta relazione outer e inner, algoritmo di join



...



# SPAZIO DI RICERCA DEI PIANI

- Soluzione
  - Utilizzare euristiche (analogamente a ottimizzazione logica) per evitare di considerare piani di accesso che sicuramente non possono risultare ottimali
  - Pruning dello spazio di ricerca

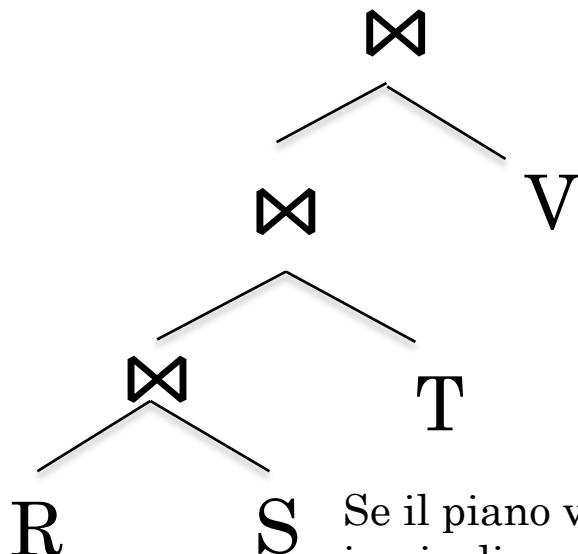
# EURISTICA PER JOIN

- Per restringere lo spazio della ricerca
  - non vengono considerate permutazioni di join che implicano dei prodotti cartesiani
  - si considerano solo **piani left-deep**, in cui il figlio destro di ogni nodo etichettato con un join è una relazione di base
- Piani left-deep permettono di generare piani di esecuzione **fully-pipelined**, in cui tutti gli operatori vengono eseguiti in pipeline
  - le relazioni inner sono già materializzate
  - se il piano non è left-deep, un piano in cui la relazione inner sia il risultato di un join forza a materializzare il risultato del join

## ESEMPIO – PIANO LEFT DEEP

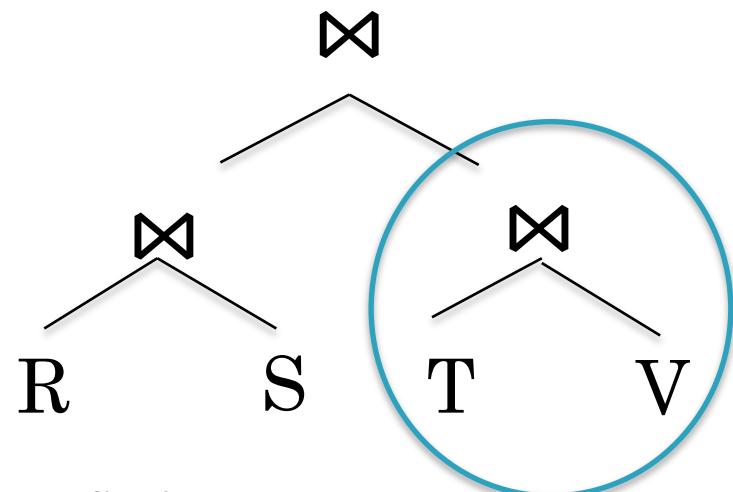
$R \bowtie S \bowtie T \bowtie V$

Piano left deep



Se il piano viene eseguito  
in pipeline,  
*la parte destra è già  
materializzata in quanto è  
una relazione di base*

Piano non left deep



Se il piano viene  
eseguito in pipeline,  
*la parte destra deve  
essere materializzata*