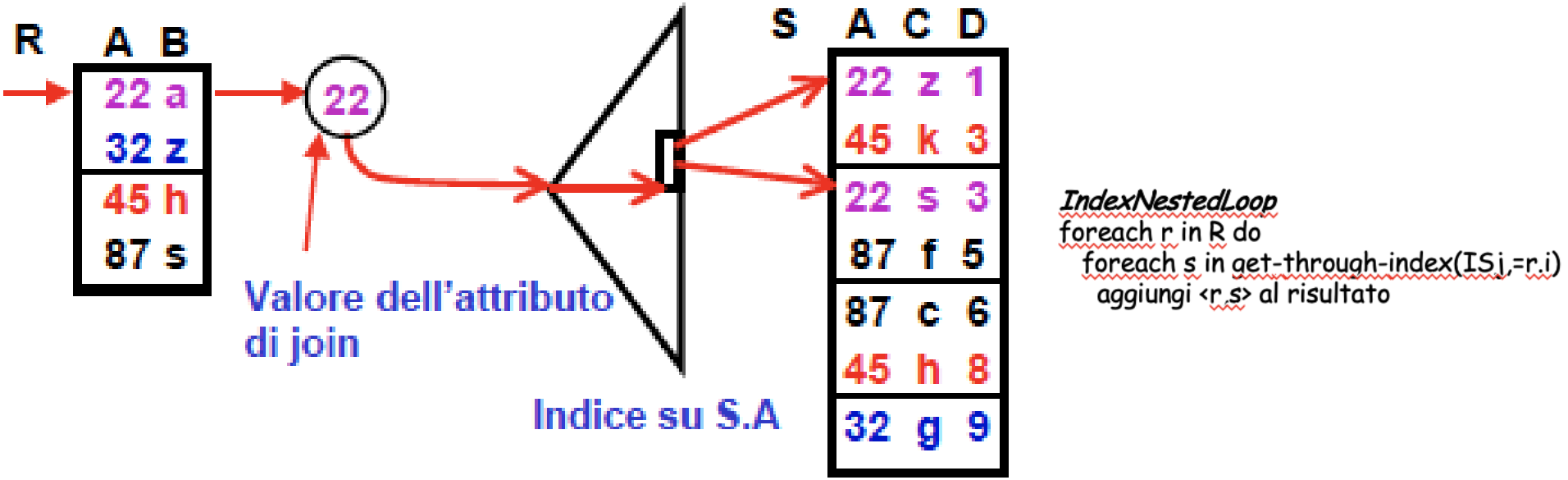
Nested loop con indice (IndexNestedLoop)

- 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, secondo il seguente schema:



- L'accesso alla relazione interna mediante indice porta in generale a ridurre di molto i costi di esecuzione del Nested Loops Join

Sort-merge:

- Il Sort-merge Join è applicabile quando entrambi gli insiemi di tuple in input sono **ordinati** sugli attributi di join
- Per R (S) ciò è possibile se:
  - R (S) è fisicamente ordinata sugli attributi di join
  - Esiste un indice sugli attributi di join di R (S)

La logica dell'algoritmo (**senza** considerare il tempo per il **sort**) sfrutta il fatto che entrambi gli input sono ordinati per evitare di fare inutili confronti, il che fa sì che il numero di letture sia dell'ordine di

Npag(R) + Npag(S)

se si accede sequenzialmente alle due relazioni

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

SortMerge
r = first(R); s = first(S);
while r in R and s in S do
  if r.i = s.i
    avanza r ed s fino a che r.i ed s.i non cambiano entrambe, aggiungendo ciascun <r,s> al risultato
  else if r.i < s.i avanza r dentro R
  else if r.i > s.i avanza s dentro S

ALTRI METODI (riepilogo)

- Nested loop a pagine:
  - Per ogni pagina di R, si visitano le pagine di S, e si trovano i record <r, s> della giunzione, con r in R-pagina e s in S-pagina.
- Nested loop con indice:
  - Si usa quando esiste l'indice ISj sull'attributo di giunzione j della relazione interna S
- Sort-merge:
  - Si usa quando R e S sono ordinate sull'attributo di giunzione: si visitano in R ed S in parallelo

PageNestedLoop

```
foreach r in R do
  foreach s in S where r.i = s.j do
    aggiungi <r, s> al risultato
```

IndexNestedLoop

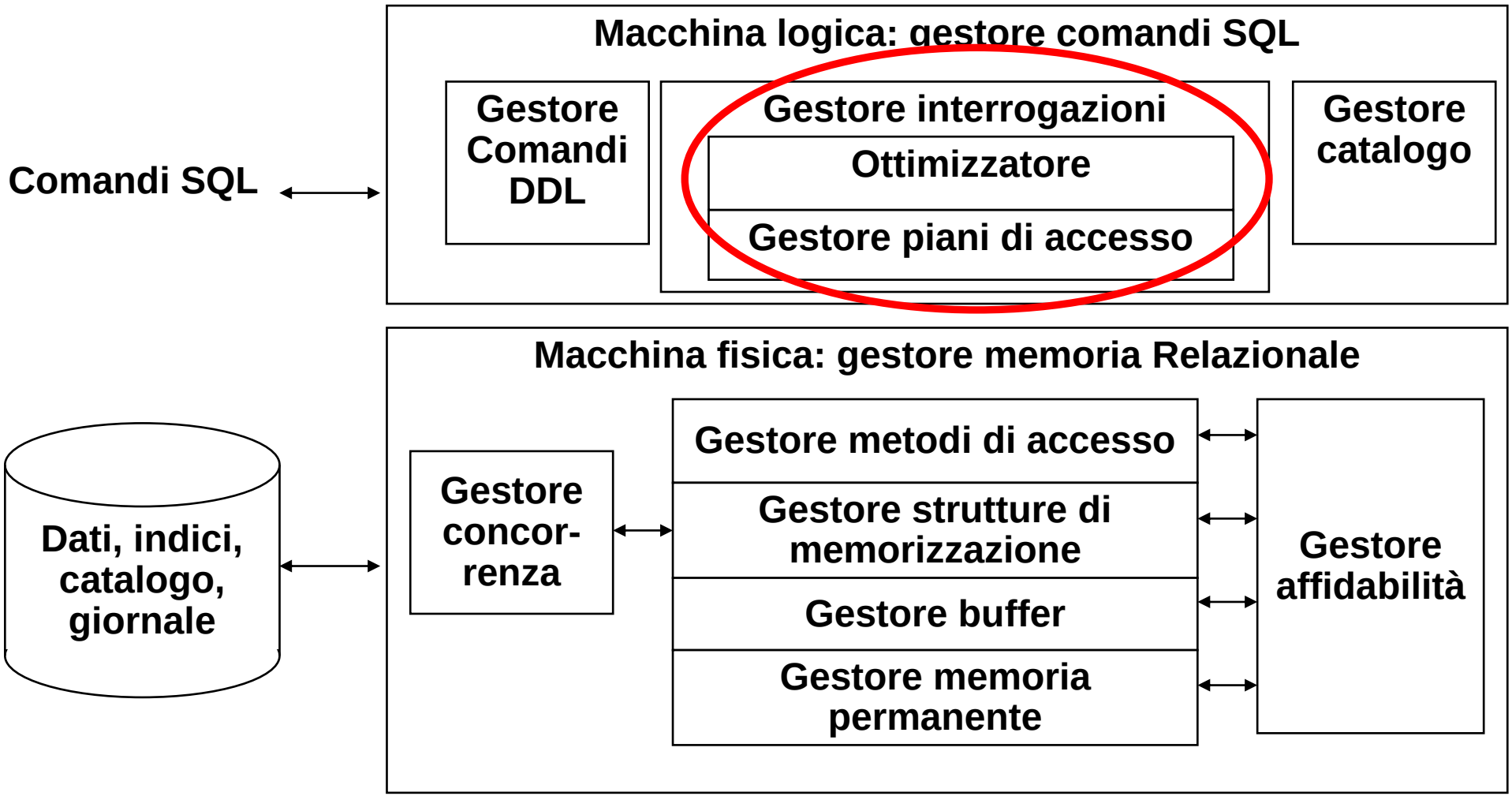
```
foreach r in R do
  foreach s in get-through-index(ISj,=r.i)
    aggiungi <r,s> al risultato
```

SortMerge

```
r = first(R); s = first(S);
while r in R and s in S do
  if r.i = s.j
     avanza r ed s fino a che r.i ed s.j non
     cambiano entrambe, aggiungendo
     ciascun <r,s> al risultato
  else if r.i < s.j avanza r dentro R
  else if r.i > s.j avanza s dentro S
```

PIANI DI ACCESSO

ARCHITETTURA DEI DBMS



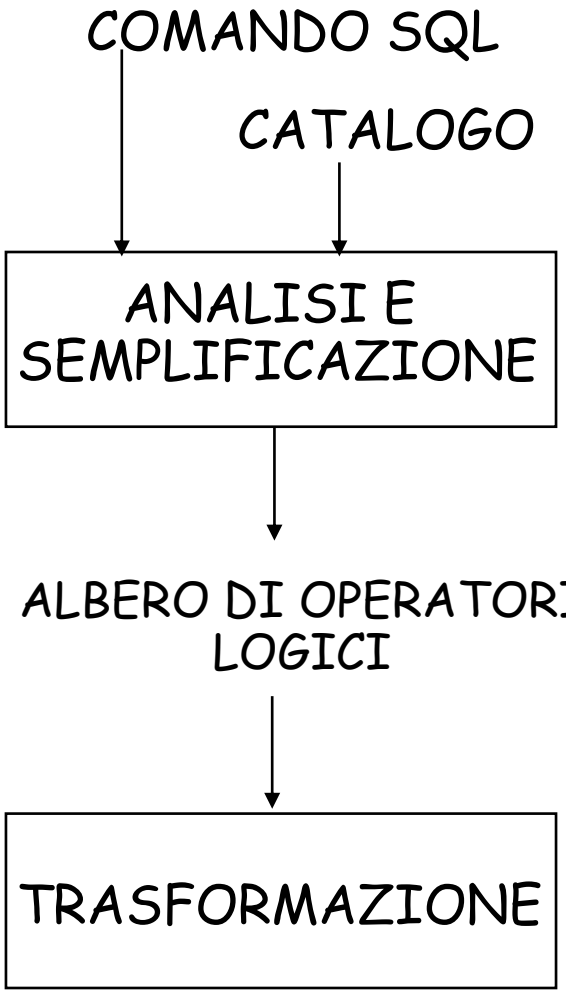
ESECUZIONE DI UN'INTERROGAZIONE

```
// analisi lessicale e sintattica del comando SQL Q
SQLCommand parseTree = Parser.parseStatement(Q);

// analisi semantica del comando
Type type = parseTree.check();

// ottimizzazione dell'interrogazione
Value pianoDiAccesso = parseTree.Optimize();

// esecuzione del piano di accesso
pianoDiAccesso.open();
while !pianoDiAccesso.isDone() do
{ Record rec = pianoDiAccesso.next();
  print(rec);
}
pianoDiAccesso.close();
```

OTTIMIZZATORE DELLE INTERROGAZIONI

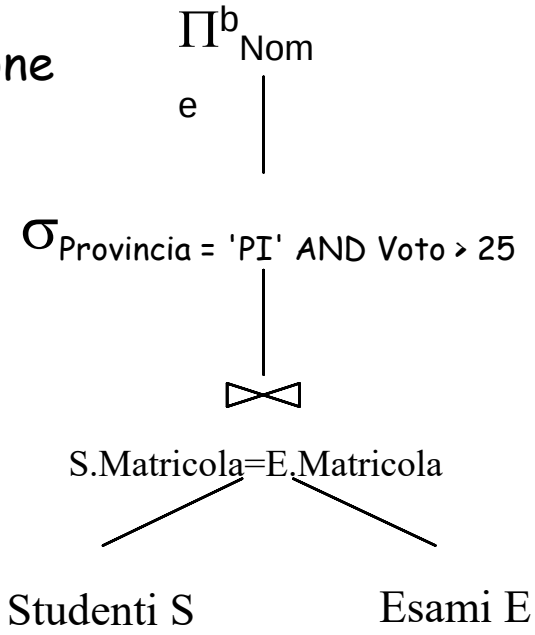
- L'ottimizzazione delle interrogazione è fondamentale nei DBMS.
- E' necessario conoscere il funzionamento dell'ottimizzatore per una buona progettazione fisica.
- Obiettivo dell'ottimizzatore:
  - Scegliere il piano con costo minimo, fra possibili piani alternativi, usando le statistiche presenti nel catalogo.

FASI DEL PROCESSO DI OTTIMIZZAZIONE



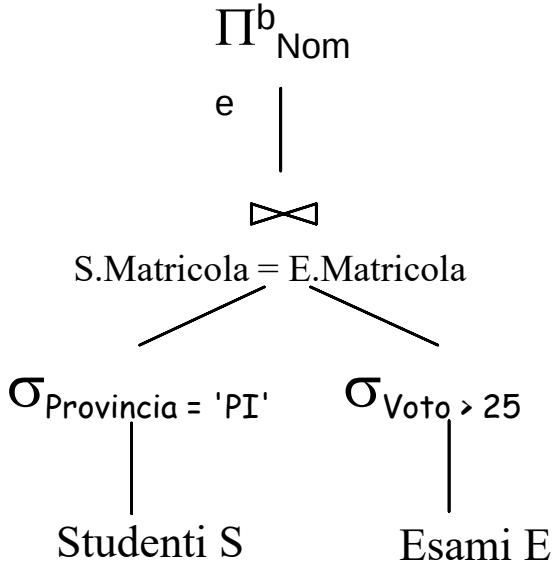

```
SELECT   Nome
FROM     Studenti S, Esami E
WHERE    S.Matricola=E.Matricola AND
         Provincia='PI' AND Voto>25
```

Verifica la correttezza del comando, normalizzazione e semplificazione della condizione



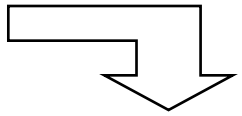
FASI DEL PROCESSO (cont)

Trasformazione dell'albero con regole di equivalenza



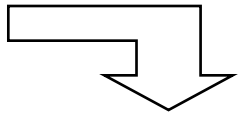
TRASFORMAZIONI INTERESSANTI

```
SELECT   Matricola, Nome
FROM     Studenti
WHERE    Matricola IN (  SELECT Matricola
                         FROM   Esami
                         WHERE  Materia = 'BD');
```



```
SELECT   Matricola, Nome
FROM     Studenti S, Esami E
WHERE    S.Matricola = E.Matricola AND Materia = 'BD';
```

```
SELECT   Matricola, Nome
FROM     VistaStudentiPisani S, VistaEsamiBD E
WHERE    S.Matricola = E.Matricola ;
```



?

FASI DEL PROCESSO (cont.)

OTTIMIZZAZIONE FISICA

↓

PIANO DI ACCESSO: ALBERO DI OPERATORI FISICI

↓

ESECUZIONE PIANO DI ACCESSO

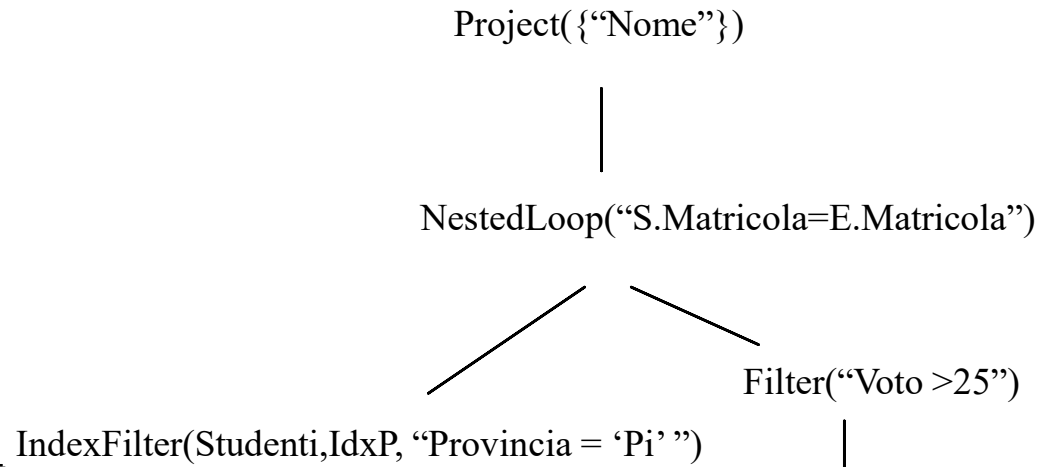
↓

RISULTATO

Piano di accesso: scelta dell'algoritmo per eseguire ogni operatore.

Ideale: Trovare il piano migliore

Euristica: evitare i piani peggiori!



nome
Tonio
Pino

Le foglie sono le tabelle ed i nodi interni specificano le modalità con cui gli accessi alle tabelle e le operazioni relazionali sono effettuate

REALIZZAZIONE DEGLI OPERATORI RELAZIONALI

- Si considerano i seguenti operatori:
  - Proiezione
  - Selezione
  - Raggruppamento
  - Join

- Un operatore può essere realizzato con algoritmi diversi, codificati in opportuni operatori fisici.

OPERATORI FISICI

- Gli algoritmi per realizzare gli operatori relazionali si codificano in opportuni operatori fisici.
  - Ad esempio ***TableScan* (R),** è l'operatore fisico per la scansione di R.
- Ogni operatore fisico è un **iteratore**, un oggetto con metodi
  - *open*,
  - *next*,
  - *isDone*,
  - *reset*
  - *close*
- realizzati usando gli operatori della macchina fisica, con *next* che ritorna un record.
- Come esempio di operatori fisici prenderemo in considerazione quelli del sistema JRS e poi vedremo come utilizzarli per descrivere un algoritmo per eseguire un'interrogazione SQL (**piano di accesso**).

Interfaccia a iteratore

- I DBMS definiscono gli operatori mediante un'interfaccia a "iteratore", i cui metodi principali sono:
  - open : inizializza lo stato dell'operatore, alloca buffer per gli input e l'output, richiama ricorsivamente open sugli operatori figli; viene anche usato per **passare argomenti** (ad es. la condizione che un operatore Filter deve applicare)
  - next : usato per richiedere un'altra tupla del risultato dell'operatore; l'implementazione di questo metodo include next sugli operatori figli e codice specifico dell'operatore
  - close: usato per terminare l'esecuzione dell'operatore, con conseguente rilascio delle risorse ad esso allocate
  - isDone: indica se vi sono ancora valori da leggere, in genera è booleano.

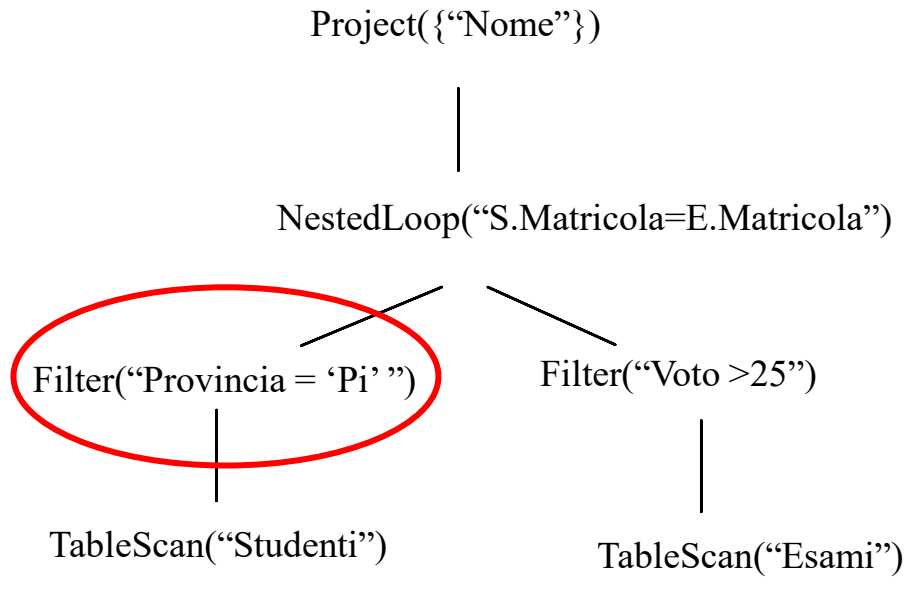
PIANI DI ACCESSO

Un piano di accesso è un algoritmo per eseguire un'interrogazione usando gli operatori fisici disponibili.

Interrogazione:

```
SELECT  Nome
FROM    Studenti S, Esami E
WHERE   S.Matricola=E.Matricola AND
        Provincia='PI' AND Voto>25
```

Piano di accesso:



Le foglie sono le tabelle ed i nodi interni specificano le modalità con cui gli accessi alle tabelle e le operazioni relazionali sono effettuate

OPERATORI LOGICI E FISICI

Operatore logico	Operatore fisico
R	***TableScan*** **(R)** per la scansione di R;
	IndexScan **(R, Idx)** per la scansione di R con l'indice Idx;
	SortScan **(R, {A_i})** per la scansione di R ordinata sugli {A_i};
$\pi_{\{Ai\}}$	***Project*** **(O, {A_i})** per la proiezione dei record di O senza l'eliminazione dei duplicati;
	Distinct **(O)** per eliminare i duplicati dei record ordinati di O;

Quale è la differenza fra gli operatori che prendono R e quelli che prendono O?

OPERATORI LOGICI E FISICI (cont.)

Quale è la differenza fra gli operatori che prendono R e quelli che prendono O?

Operatore logico	Operatore fisico
σ_{ψ}	***Filter*** **(O, ψ)** per la restrizione senza indici dei record di O;
	IndexFilter **(R, Idx, ψ)** per la restrizione con indice dei record di R;
$\tau_{\{Ai\}}$	***Sort*** **(O, {A_i})** per ordinare i record di O sugli {A_i}, per valori crescenti;

OPERATORI LOGICI E FISICI (cont.)

Operatore logico	Operatore fisico
$\{Ai\}\ \gamma\ \{fi\}$	***GroupBy*** **(O, {A_i}, {f_i})** per raggruppare i record di O sugli {A_i} usando le funzioni di aggregazione in {f_i} . • Nell'insieme {f_i} vi sono le funzioni di aggregazione presenti nella SELECT e nella HAVING. • L'operatore restituisce record con attributi gli {A_i} e le funzioni in {f_i}. • I record di O sono ordinati sugli {A_i};

OPERATORI LOGICI E FISICI (cont.)

Operatore logico	Operatore fisico
$\bowtie_{\psi_J}$	***NestedLoop*** (O_E, O_I, ψ_J) per la giunzione con il *nested loop* e ψ_J la condizione di giunzione;
	PageNestedLoop (O_E, O_I, ψ_J) per la giunzione con il *page nested loop*;
	IndexNestedLoop (O_E, O_I, ψ_J) per la giunzione con il *index nested loop*. L'operando interno O_I è un *IndexFilter*(R, Idx, ψ_J) oppure *Filter* (O,ψ'): *con O un IndexFilter*(R, Idx, ψ_J); per ogni record *r* di O_E, la condizione ψ_J dell'*IndexFilter* è quella di giunzione con gli attributi di O_E sostituiti dai valori in *r*.
	SortMerge (O_E, O_I, ψ_J) per la giunzione con il *sort-merge*, con i record di O_E e O_I ordinati sugli attributi di giunzione.

Esempi: tableScan e SortScan

```
SELECT   *
FROM     R ;
```

Albero logico

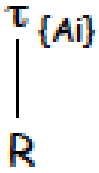
TableScan
(R)

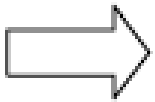
Piano di accesso

per la scansione di R

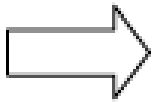
Non considerando la proiezione

```
SELECT   *
FROM     R
ORDER BY A ;
```





SortScan
(R, {A})



Sort
({A})

TableScan
(R)

per la scansione di R ordinata su {A};

OPERATORI FISICI PER LA PROIEZIONE

Operatori per $\pi^{b}_{\{Ai\}}$ e $\pi_{\{Ai\}}$

- Project (O, {Ai}): per la proiezione dei record di O senza l'eliminazione dei duplicati;

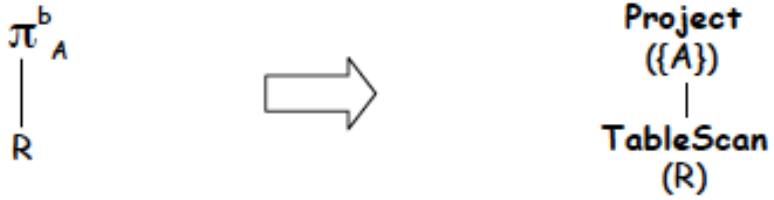
```
SELECT   A
FROM     R ;
```

OPERATORI FISICI PER LA PROIEZIONE

Operatori per $\pi^{b}_{\{Ai\}}$ e $\pi_{\{Ai\}}$

- Project (O, {Ai}): per la proiezione dei record di O senza l'eliminazione dei duplicati;

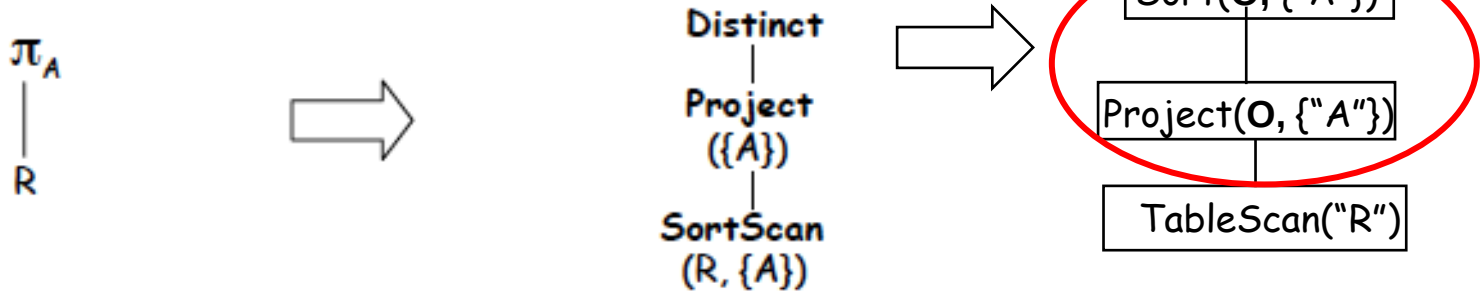
```
SELECT   A
FROM     R ;
```



Albero logico — Piano di acces

- Distinct (O): per eliminare i duplicati dei record ordinati di O;

```
SELECT   DISTINCT A
FROM     R ;
```



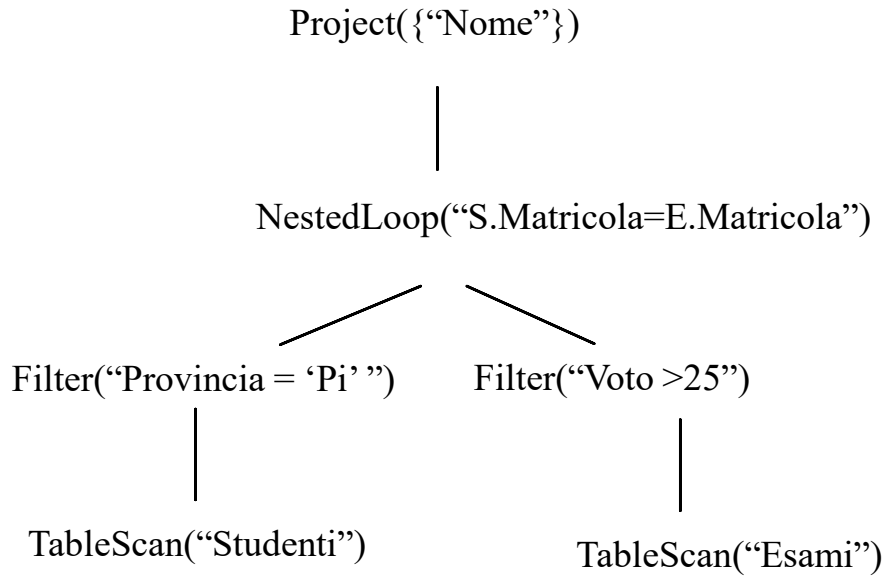
PIANI DI ACCESSO

Un piano di accesso è un algoritmo per eseguire un'interrogazione usando gli operatori fisici disponibili.

Interrogazione:

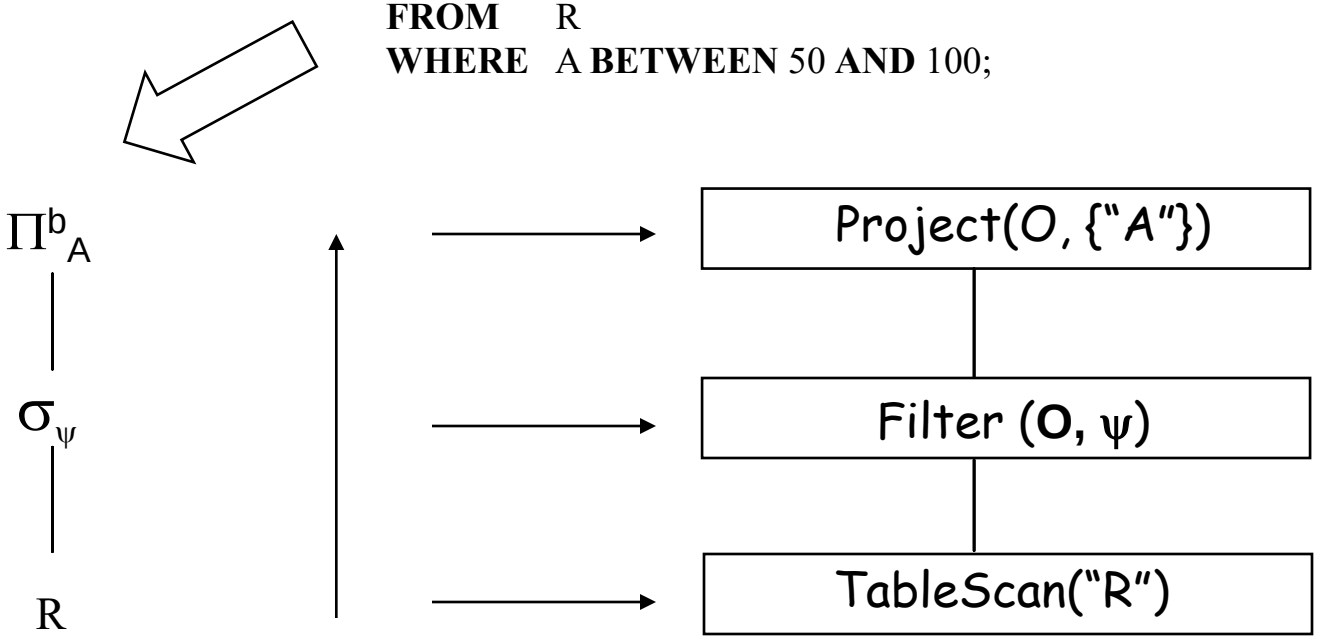
```
SELECT  Nome
FROM    Studenti S, Esami E
WHERE   S.Matricola=E.Matricola AND
        Provincia='PI' AND Voto>25
```

Piano di accesso:



1) ESEMPIO DI PIANO DI ACCESSO: SFW

```
SELECT   A
FROM     R
WHERE    A BETWEEN 50 AND 100;
```



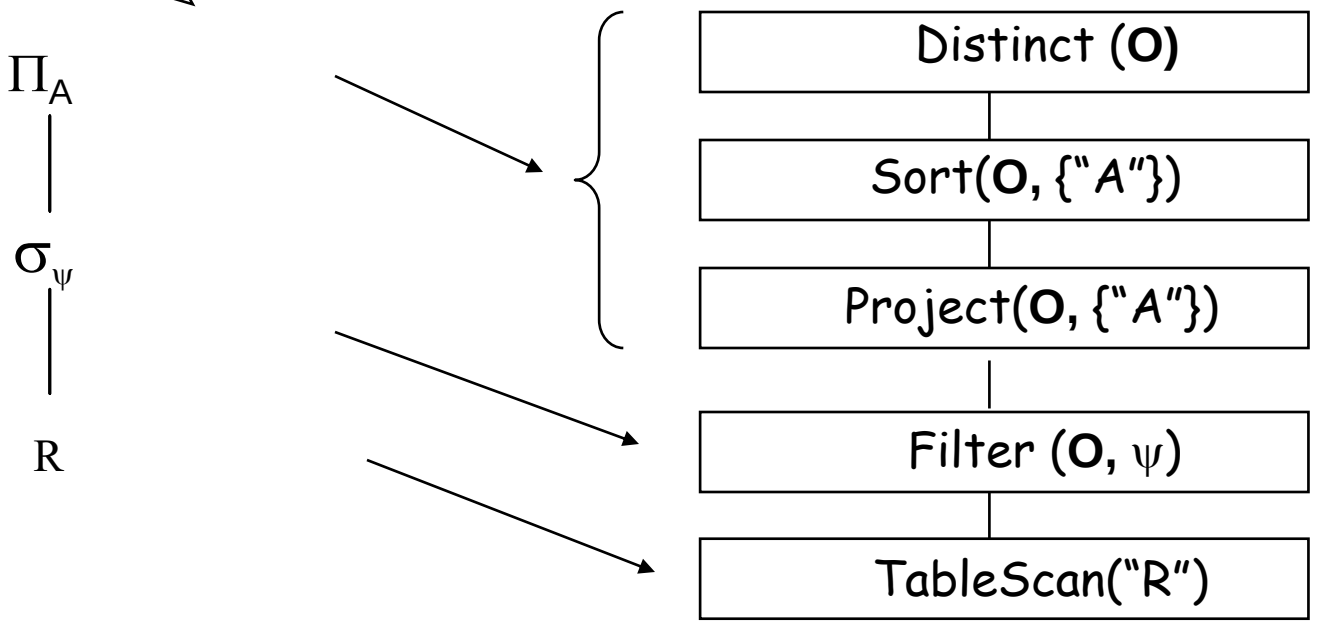
Albero logico

Albero Fisico
(Piano di accesso)

ψ = A BETWEEN 50 AND 100

2) ESEMPIO DI PIANO DI ACCESSO: DISTINCT

```
SELECT   DISTINCT A
FROM     R
WHERE    A BETWEEN 50 AND 100;
```

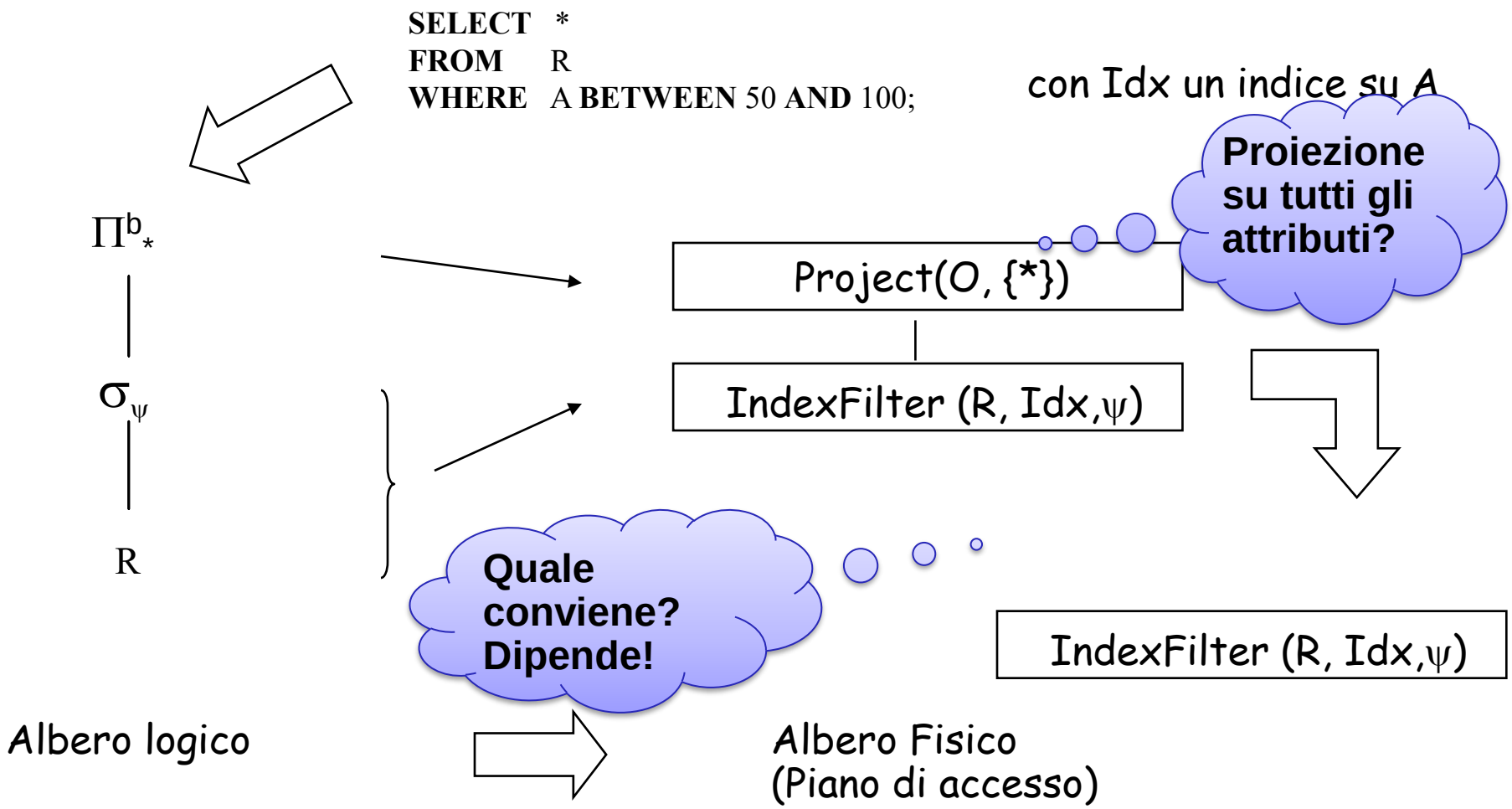


Albero logico

Albero Fisico
(Piano di accesso)

ψ = A **BETWEEN** 50 **AND** 100

3) ESEMPIO DI PIANO DI ACCESSO CON INDICE

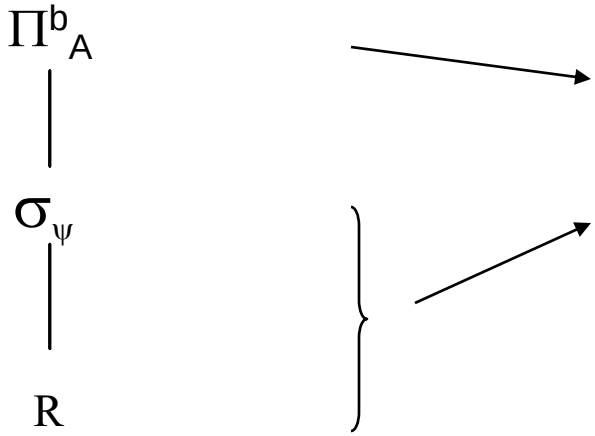


ψ = *A* **BETWEEN** 50 **AND** 100

4) ESEMPIO DI PIANO DI ACCESSO CON INDICE

```
SELECT   A
FROM     R
WHERE    A BETWEEN 50 AND 100;
```

con Idx un indice su A



Project(O, {"A"})

IndexFilter (R, Idx,ψ)

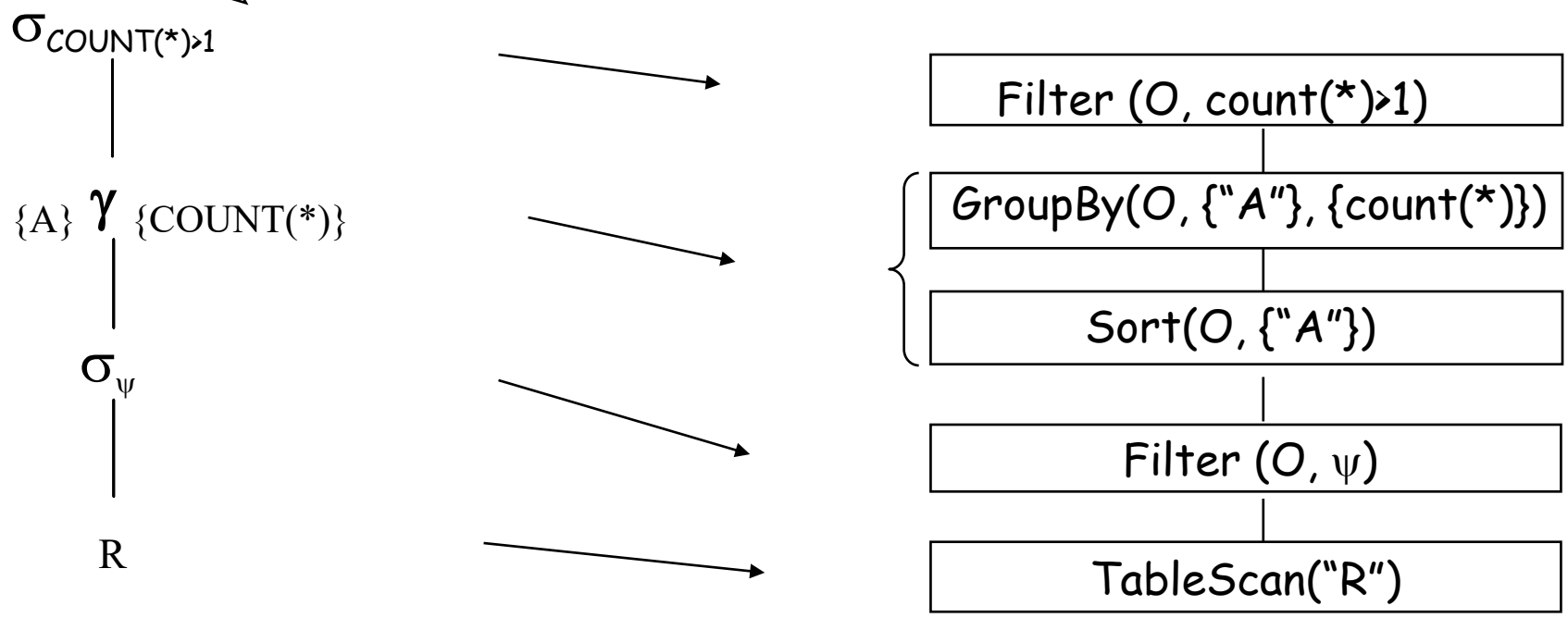
IndexOnlyFilter (R, Idx,ψ)

Albero logico ⇒ Albero Fisico (Piano di accesso)

ψ = A **BETWEEN** 50 **AND** 100

5) ESEMPIO DI PIANO DI ACCESSO: GROUP BY

```
SELECT  A,  COUNT(*)
FROM    R
WHERE   A BETWEEN 50 AND 100
GROUP BY A HAVING COUNT(*) > 1;
```

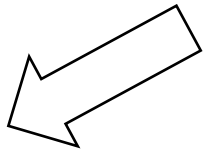


Albero logico

Albero Fisico
(Piano di accesso)

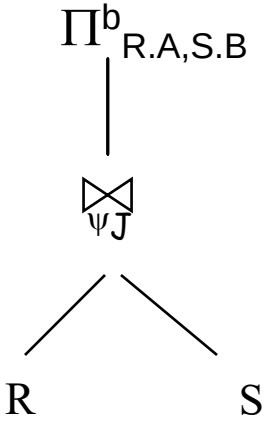
ψ = A **BETWEEN** 50 **AND** 100

6) ESEMPIO DI PIANO DI ACCESSO: GIUNZIONE SENZA INDICE

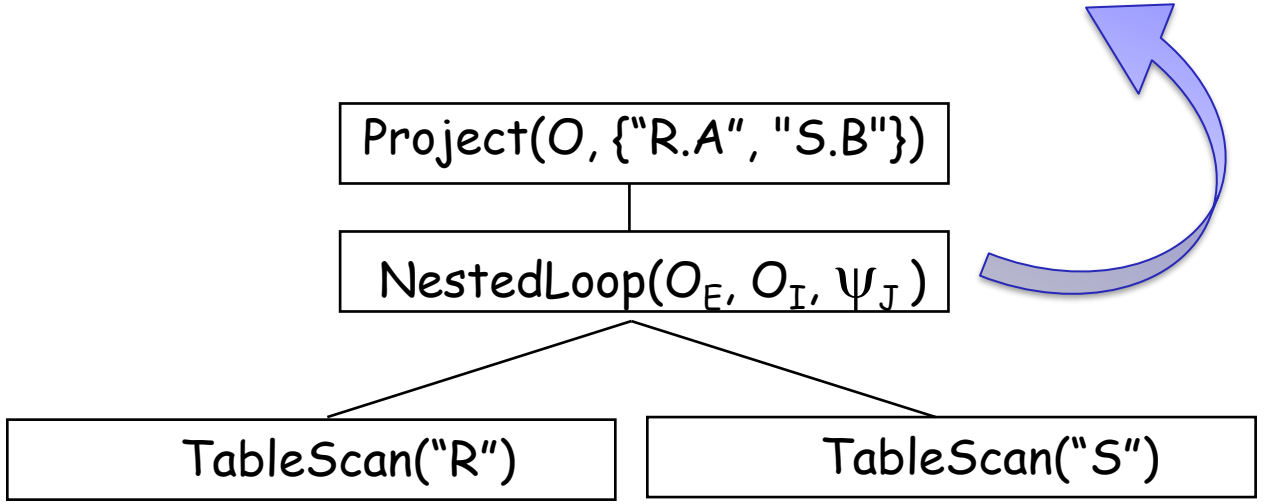


```
SELECT  R.A,  S.B
FROM    R ,  S
WHERE   R.A = S.B;
```

```
foreach record r in R do
  foreach record s in S do
    if r_i = s_j then
       aggiungi <r, s> al risultato
```



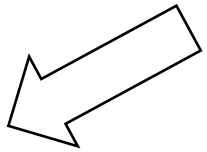
Albero logico



Albero Fisico
(Piano di accesso)

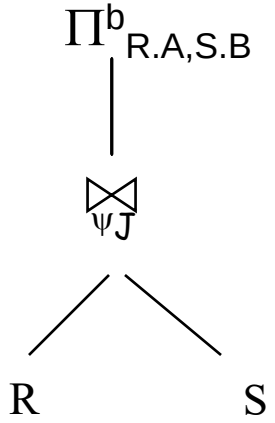
ψ_J = R.A = S.B

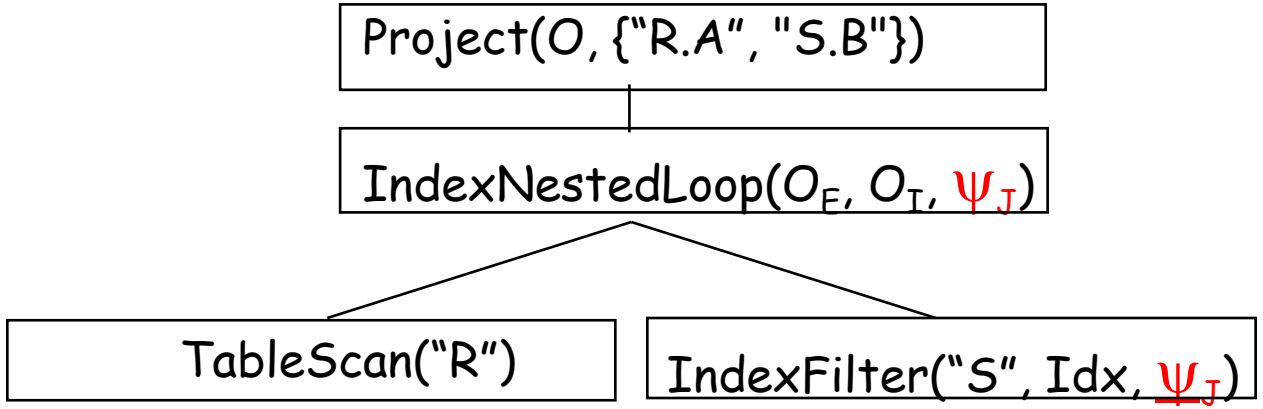
6) ESEMPIO DI PIANO DI ACCESSO: GIUNZIONE CON INDICE



```
SELECT  R.A,  S.B
FROM    R ,  S
WHERE   R.A = S.B;
```

con Idx un indice su S.B

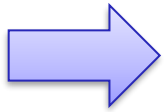




Albero logico

Albero Fisico
(Piano di accesso)

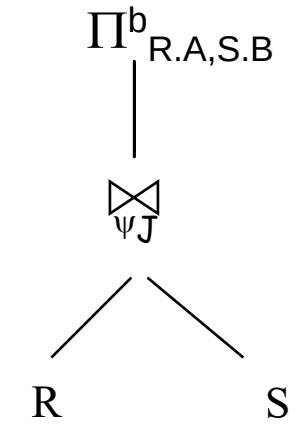
ψ_J = R.A = S.B



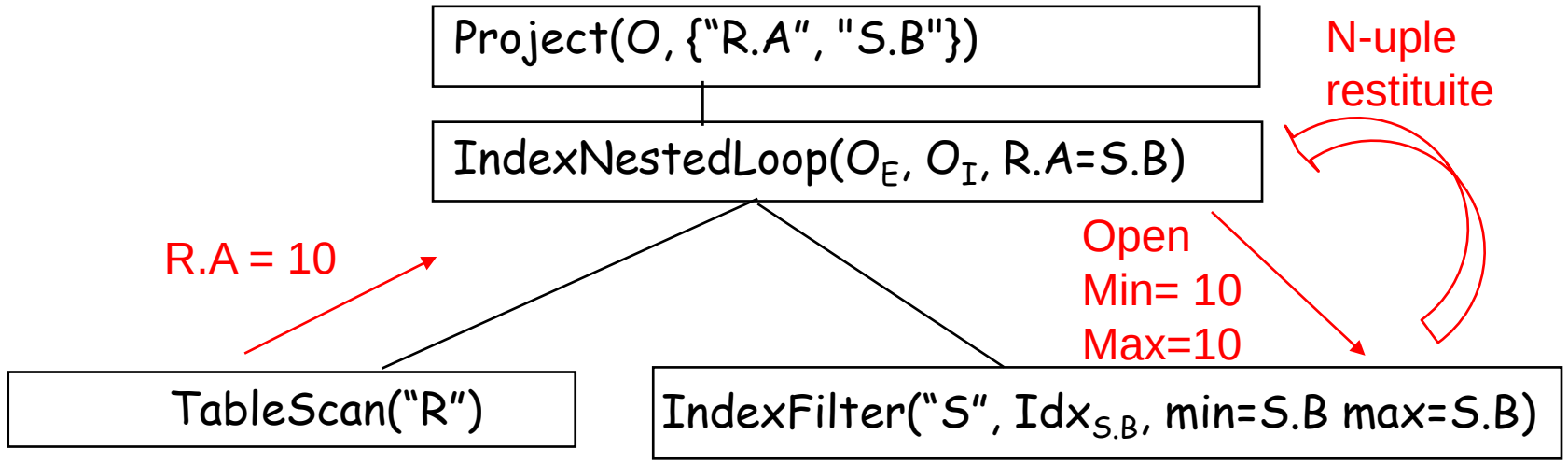
6) ESEMPIO DI PIANO DI ACCESSO: GIUNZIONE CON INDICE

```
SELECT  R.A,  S.B
FROM    R ,  S
WHERE   R.A = S.B;
```

con Idx un indice su S.B



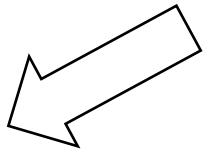
Albero logico



Albero Fisico
(Piano di accesso)

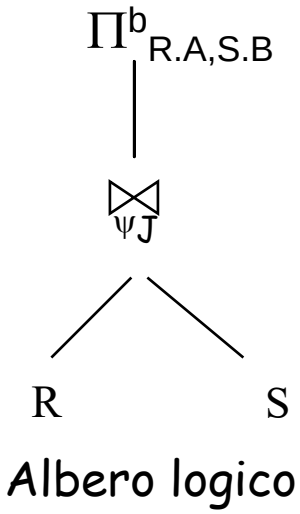
ψ_J = R.A = S.B

6) ESEMPIO DI PIANO DI ACCESSO: GIUNZIONE CON INDICE



```
SELECT  R.A,  S.B
FROM    R ,  S
WHERE   R.A = S.B;
```

con Idx un indice su S.B



Albero logico

Attributi

Project(O, {"R.A", "S.B"})

N-uple restituite

IndexNestedLoop(O_E, O_I, R.A=S.B)

R.A = 10

Open
Min= 10
Max=10

TableScan("R")

IndexFilter("S", $Idx_{S.B}$, R.A=S.B)

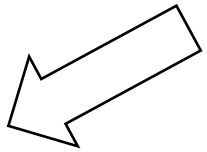
Possiamo invertire i due sottoalberi?

Albero Fisico
(Piano di accesso)

ψ_J = R.A = S.B

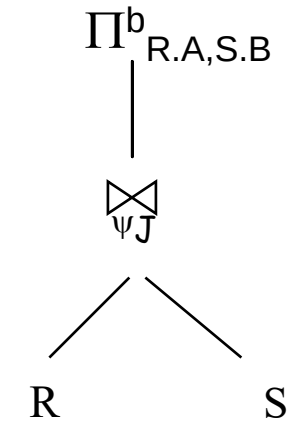
Valore in run-time

6) ESEMPIO DI PIANO DI ACCESSO: GIUNZIONE CON INDICE

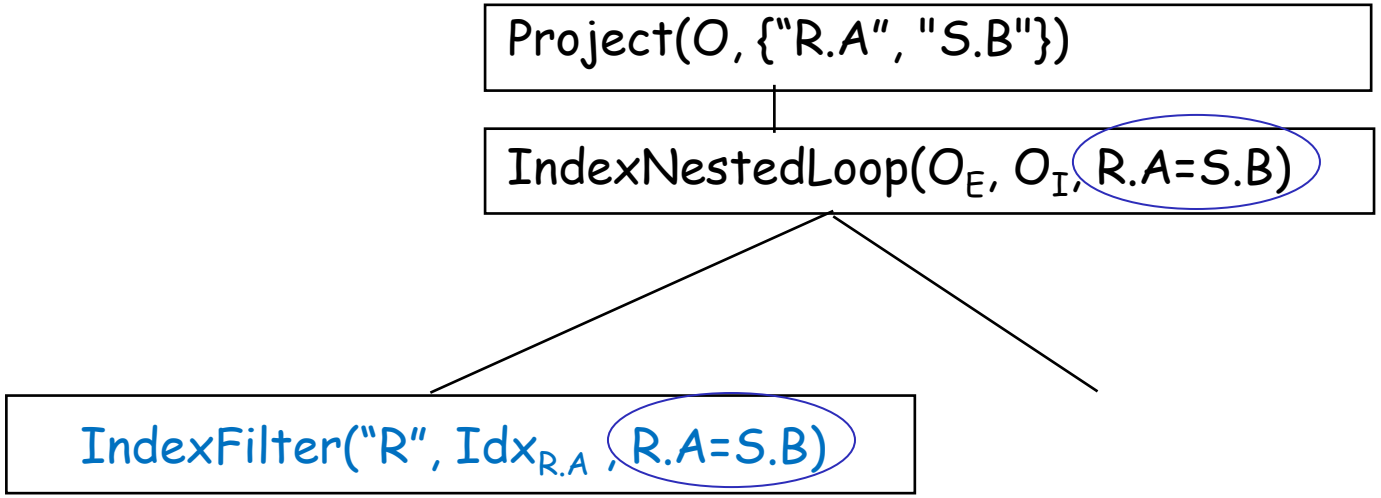


```
SELECT  R.A,  S.B
FROM    R ,  S
WHERE   R.A = S.B;
```

con Idx un indice su S.B



Albero logico



Albero Fisico
(Piano di accesso)

ALTRI ESERCIZI

1)
```
SELECT   A
FROM     R
WHERE    A BETWEEN 50 AND 100
ORDER BY A;
```

2)
```
SELECT   DISTINCT A
FROM     R
WHERE    A BETWEEN 50 AND 100
ORDER BY A;
```

3)
```
SELECT   DISTINCT A
FROM     R
WHERE    A BETWEEN 50 AND 100
ORDER BY A;
```

ed esiste un indice su A

4)
```
SELECT   DISTINCT A
FROM     R
WHERE    A = 100
ORDER BY A;
```

1) A e' una chiave
2) A non è una chiave

5)
```
SELECT   A, COUNT(*)
FROM     R
WHERE    A > 100
GROUP BY A;
```

6)
```
SELECT   DISTINCT A, COUNT(*)
FROM     R
WHERE    A > 100
GROUP BY A;
```

Soluzione Esercizio 6 (senza indici)

```
SELECT   DISTINCT A, COUNT(*)
FROM     R
WHERE    A > 100
GROUP    BY A;
```

Non serve perché la group by lascia in output solo A e count(*)

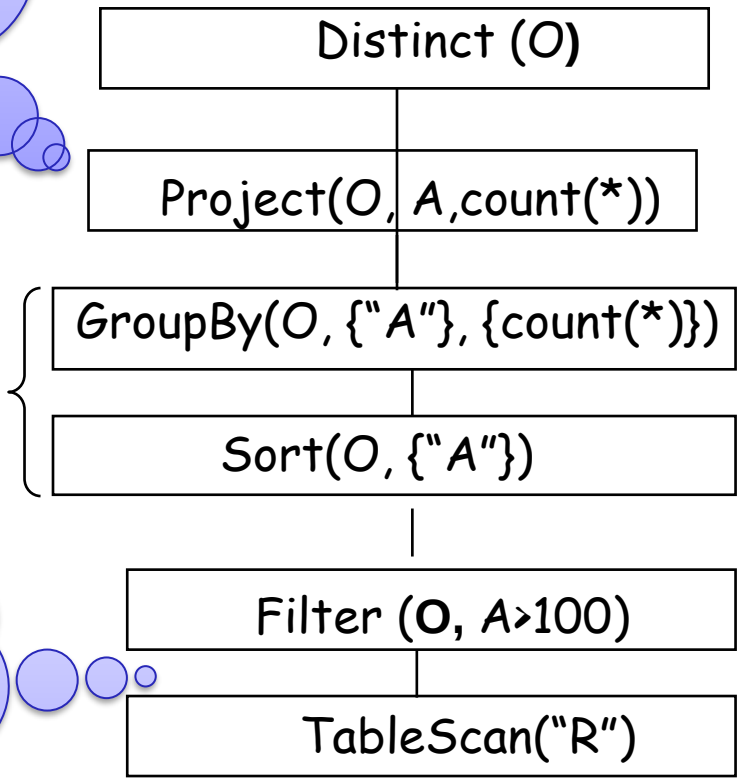
$\Pi_{A,count(*)}$

$\{A\}\ \gamma\ \{COUNT(*)\}$

$\sigma_{A>100}$

R

Albero logico

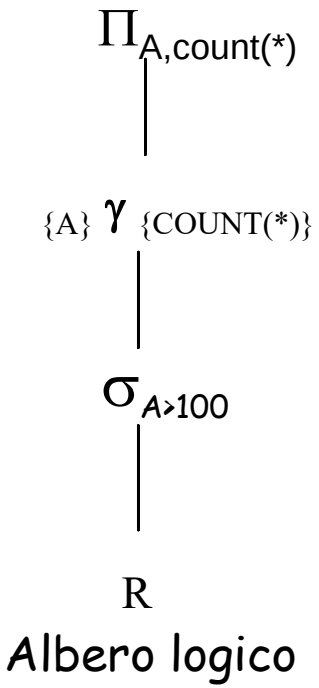


Albero Fisico (Piano di accesso)

Avrei potuto fare il sort subito dopo tableScan?

Soluzione Esercizio 6 (senza indici) Cont.

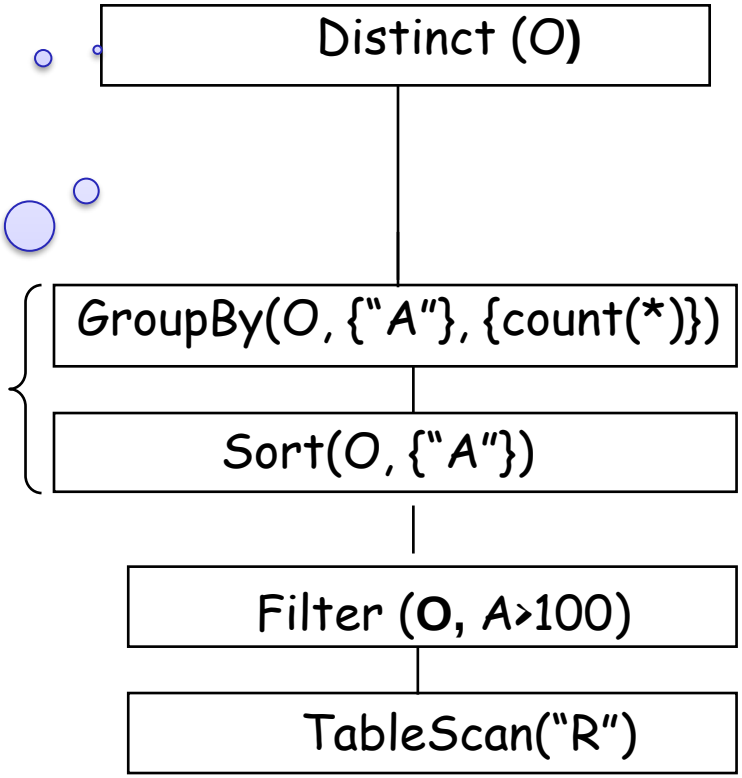
```
SELECT   DISTINCT A, COUNT(*)
FROM     R
WHERE    A > 100
GROUP    BY A;
```



Albero logico

Serve?

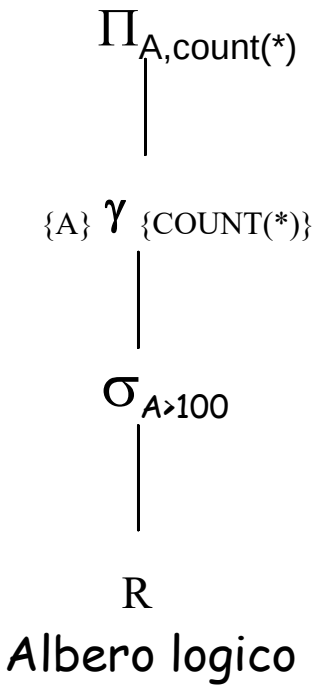
Bisogna fare un sort su A e su Count per eliminare i duplicati?



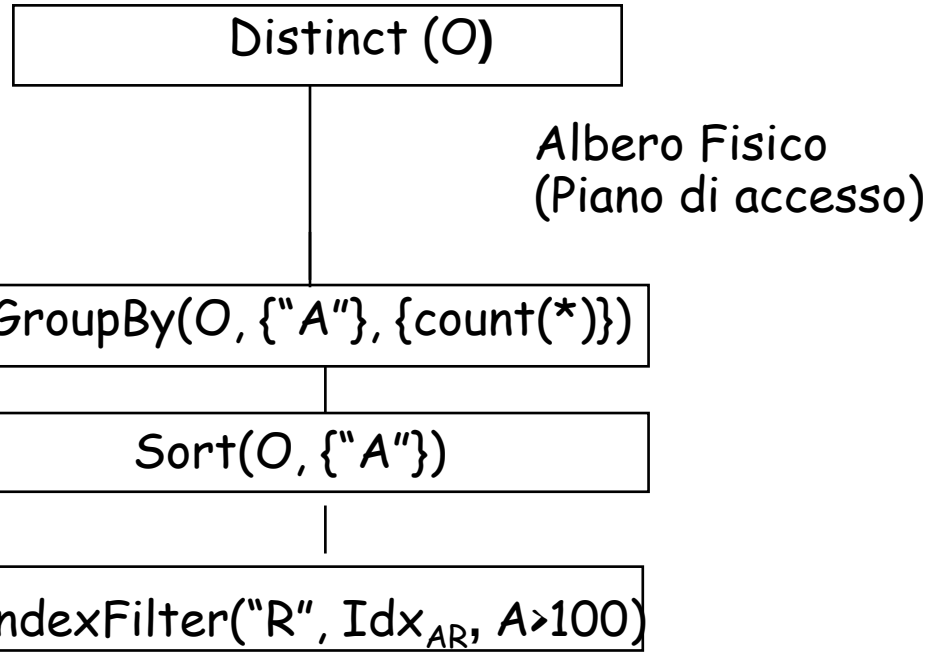
Albero Fisico (Piano di accesso)

Soluzione Esercizio 6 (con indici) Cont.

```
SELECT   DISTINCT A, COUNT(*)
FROM     R
WHERE    A > 100
GROUP    BY A;
```



Albero logico



Albero Fisico
(Piano di accesso)

Serve?

Bisogna specificare gli indici!!!