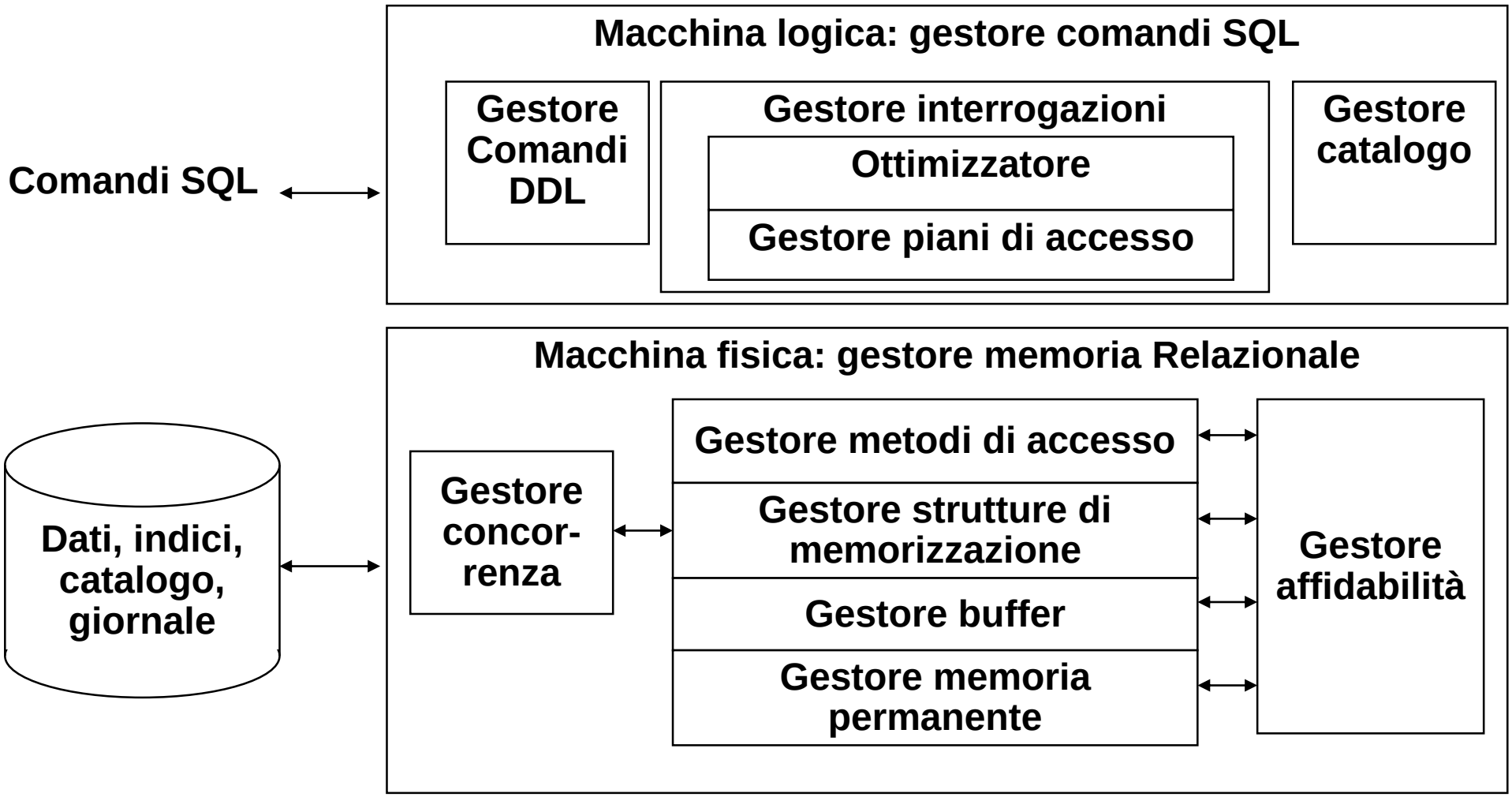
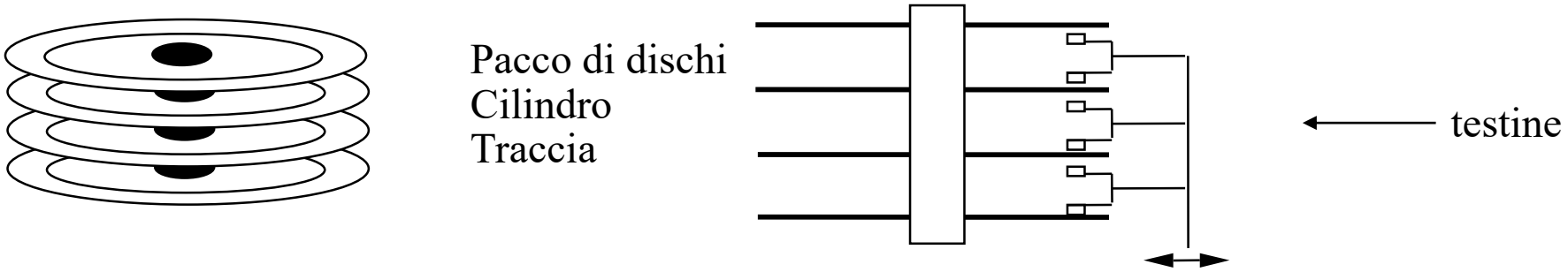
ARCHITETTURA DEI DBMS



MEMORIE A DISCO

- Un'unità a dischi contiene una pila di dischi metallici che ruota a velocità costante ed alcune testine di lettura che si muovono radialmente al disco



- Una traccia è organizzata in settori di dimensione fissa; i settori sono raggruppati logicamente in blocchi, che sono l'unità di trasferimento.
- Trasferire un blocco richiede un tempo di posizionamento delle testine, un tempo di latenza rotazionale e tempo per il trasferimento (trascurabile)
  - IBM 3380 (1980) :2Gb, 16 ms, 8.3 ms, 0.8 ms/2.4Kb
  - IBM Ultrastar 36Z15 (2001): 36GB, 4.2 ms, 2 ms, 0.02 ms/Kb

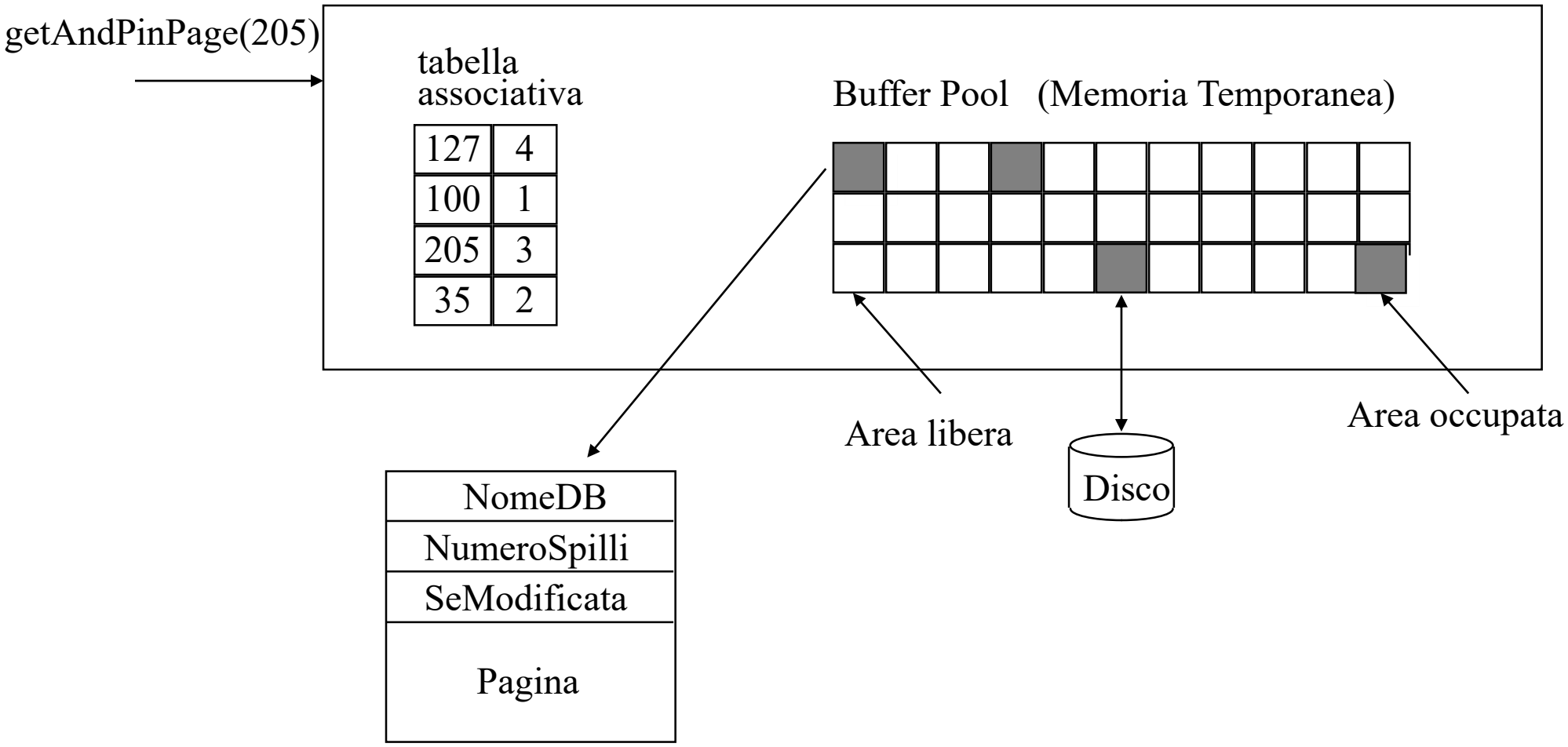
L'IMPORTANZA DELLA LOCALITÀ

- Per leggere un file da un MB servono (IBM Ultrastar 36Z15):
  - 0,027 secondi se memorizzato in settori consecutivi
  - 0,8 secondi se memorizzato in blocchi da 16 settori ciascuno (8KB) distribuiti a caso (128*(4,2+2+0,16))
  - 12,7 secondi se memorizzato in blocchi da 1 settore ciascuno, distribuiti a caso (2048*(4,2+2+0,01))

GESTORE MEMORIA PERMANENTE E GESTORE DEL BUFFER

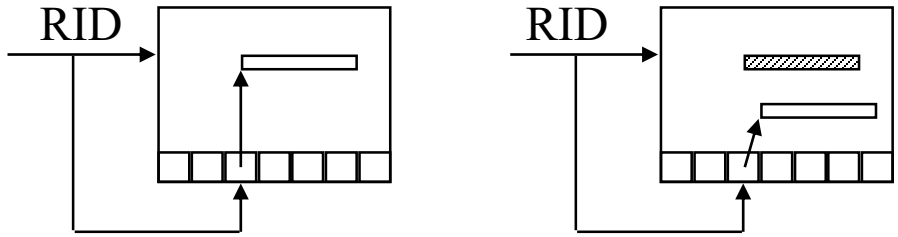
- GESTORE MEMORIA PERMANENTE
  - Fornisce un'astrazione della memoria permanente in termini di insiemi di file logici di pagine fisiche di registrazioni (blocchi), nascondendo le caratteristiche dei dischi e del sistema operativo.
- GESTORE DEL BUFFER
  - Si preoccupa del trasferimento delle pagine tra la memoria temporanea e la memoria permanente, offrendo agli altri livelli una visione della memoria permanente come un insieme di pagine utilizzabili in memoria temporanea, astraendo da quando esse vengano trasferite dalla memoria permanente al buffer e viceversa

GESTORE DEL BUFFER: AREA DELLE PAGINE



STRUTTURA DI UNA PAGINA

- Struttura fisica: un insieme, di dimensione fissa, di caratteri .
- Struttura logica:
  - informazioni di servizio;
  - un'area che contiene le stringhe che rappresentano i record;
- Il problema dei riferimenti ai record: coppia (PID della pagina, posizione nella pagina) (RID).



GESTORE STRUTTURE DI MEMORIZZAZIONE

- Tipi di organizzazioni:
  - Seriali o Sequenziali
  - Per chiave
  - Per attributi non chiave
- Parametri che caratterizzano un'organizzazione:
  - Occupazione di memoria
  - Costo delle operazioni di:
    - Ricerca per valore o intervallo
    - Modifica
    - Inserzione
    - Cancellazione

ORGANIZZAZIONI SERIALE E SEQUENZIALE

- Organizzazione seriale (heap file): i dati sono memorizzati in modo disordinato uno dopo l'altro:
  - Semplice e a basso costo di memoria
  - Poco efficiente
  - Va bene per pochi dati
  - E' l'organizzazione standard di ogni DBMS.
- Organizzazione sequenziale: i dati sono ordinati sul valore di uno o più attributi:
  - Ricerche più veloci
  - Nuove inserzioni fanno perdere l'ordinamento

ORDINAMENTO DI ARCHIVI

- Perché è importante l'ordinamento di archivi
  - Risultato di interrogazioni ordinato (order by)
  - Per eseguire alcune operazioni relazionali (join, select distinct, group by)
- Algoritmo sort-merge: costo N*Log(N)

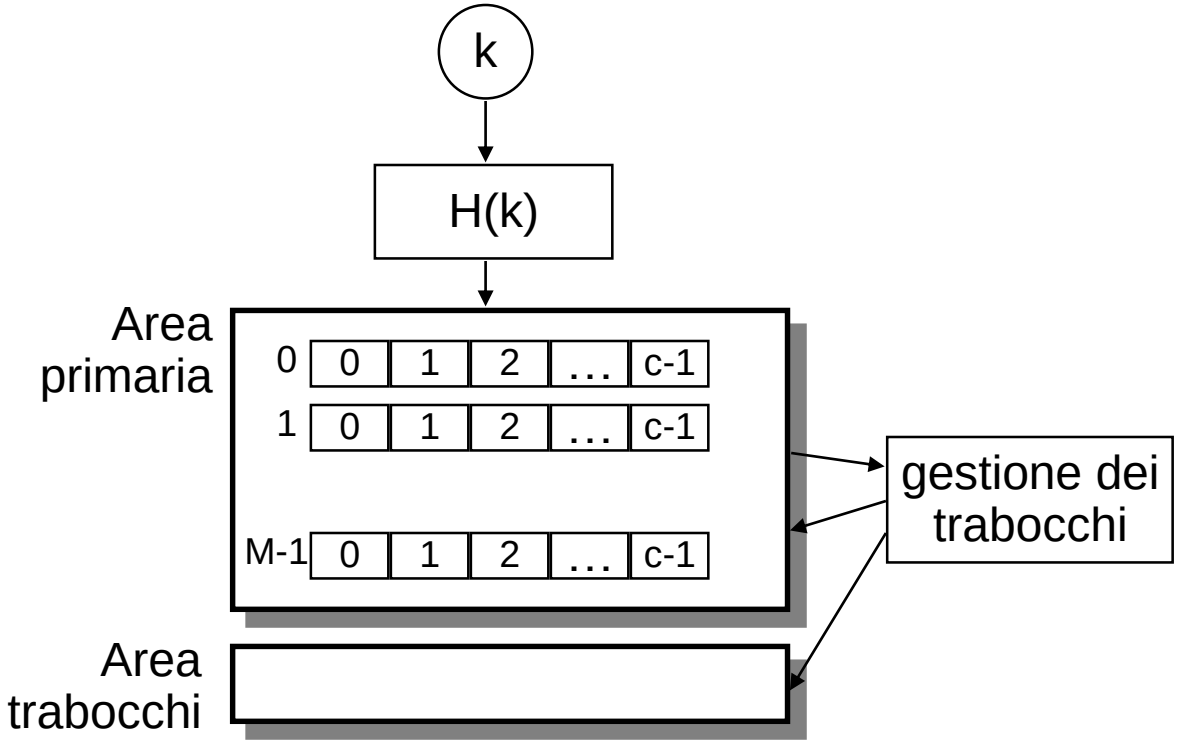
ORGANIZZAZIONI PER CHIAVE

- **Obiettivo**: noto il valore di una chiave, trovare il record di una tabella con qualche accesso al disco (ideale: 1 accesso).
- **Alternative**:
  - Metodo procedurale (hash) o tabellare (indice)
  - Organizzazione statica o dinamica.

ORGANIZZAZIONE PER CHIAVE: METODO PROCEDURALE STATICO

Parametri di progetto:

- La funzione per la trasformazione della chiave
- Il fattore di caricamento d=N/(M*c)
- La capacità c delle pagine
- Il metodo per la gestione dei trabocchi



ORGANIZZAZIONE PER CHIAVE: METODO TABELLARE

- Il metodo procedurale è utile per ricerche per chiave ma non per intervallo. Per entrambi i tipi di ricerche è utile invece il **metodo tabellare**:
  - si usa un *indice*, ovvero di un insieme **ordinato** di coppie (k, r(k)), dove *k* è un valore della chiave ed *r(k)* è un riferimento al record con chiave *k*.
- L'indice è gestito di solito con un'opportuna struttura albero detta **B^+-albero**, la struttura più usata e ottimizzata dai DBMS.
- Gli indici possono essere multi-attributi.

ESEMPI DI INDICI PER ATTRIBUTO CHIAVE O NON CHIAVE

- Tabella:

RID	Matr	Prov	An
1	106	MI	1972
2	102	PI	1970
3	107	PI	1971
4	104	FI	1968
5	100	MI	1970
6	103	PI	1972

- Indici

Matr	RID
100	5
102	2
103	6
104	4
106	1
107	3

Indice su Matr

An	RID
1968	4
1970	2
1970	5
1971	3
1972	1
1972	6

Indice su An

REALIZZAZIONE DEGLI OPERATORI RELAZIONALI

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

PROIEZIONE

```
SELECT   DISTINCT
         Provincia
FROM     Studenti R
```

- Approccio basato sull'ordinamento (non è l'unico!):
  - Si legge R e si scrive T che contiene solo gli attributi della SELECT
  - Si ordina T su tutti gli attributi
  - Si eliminano i duplicati

RESTRIZIONE CON CONDIZIONE SEMPLICE

```
SELECT  *
FROM    Studenti R
WHERE   R.Provincia = 'PI'
```

- Senza indice e dati disordinati: Npag(R).

- Con indice (B^+-albero): CI + CD

OPERAZIONI DI AGGREGAZIONE

- Senza GROUP BY
  - Si visitano i dati e si calcolano le funzioni di aggregazione.
- Con GROUP BY
  - Approccio basato sull'ordinamento (non è l'unico!):
    - Si ordinano i dati sugli attributi del GROUP BY, poi si visitano i dati e si calcolano le funzioni di aggregazione per ogni gruppo.

GIUNZIONE

```
SELECT  *
FROM    Studenti S, Esami E
WHERE   S.Matricola=E.Matricola
```

- R x S è grande; pertanto, R x S seguito da una restrizione è inefficiente.

NESTED LOOPS

- Per ogni record della relazione esterna R, si visita tutta la relazione interna S.
  - Costo: Npag(R) + Nreg(R) * Npag(S)
    ≈ Npag(R) * $\frac{\text{Nreg(R)}}{\text{Npag(R)}}$ * Npag(S)

  con S esterna:
  - Costo: Npag(S) + Nreg(S) * Npag(R)
    ≈ Npag(S) * $\frac{\text{Nreg(S)}}{\text{Npag(S)}}$ * Npag(R)

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

- Come esterna conviene la relazione con record più lunghi

ALTRI METODI

- 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
```

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* e *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**).

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};
$\pi^{b}_{\{Ai\}}$	***Project* (O, {A_i})** per la proiezione dei record di O senza l'eliminazione dei duplicati;
$\pi_{\{Ai\}}$	***Distinct* (O)** per eliminare i duplicati dei record ordinati di O;

OPERATORI LOGICI E FISICI (cont.)

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 ritorna 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, $\underline{\psi}_J$) oppure *Filter* (O,ψ'): *con O un IndexFilter*(R, Idx, $\underline{\psi}_J$); per ogni record *r* di O_E, la condizione $\underline{\psi}_J$ 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.

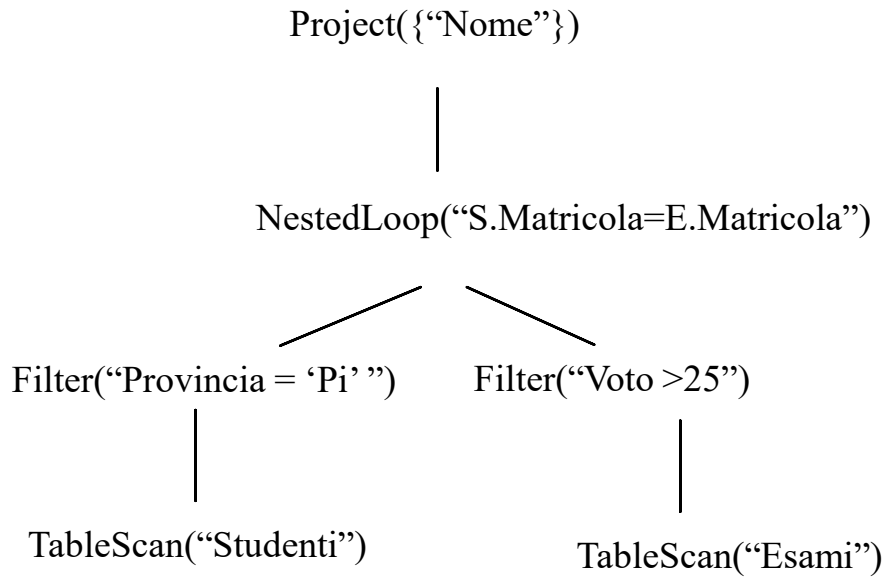
PIANI DI ACCESSO

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:



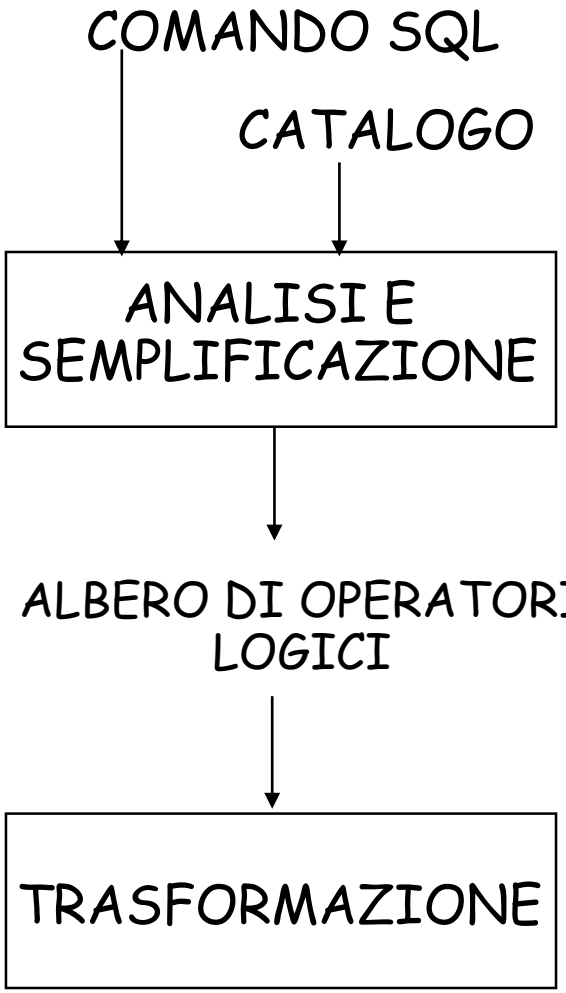
ESECUZIONE DI UN'INTERROGAZIONE

```
// 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

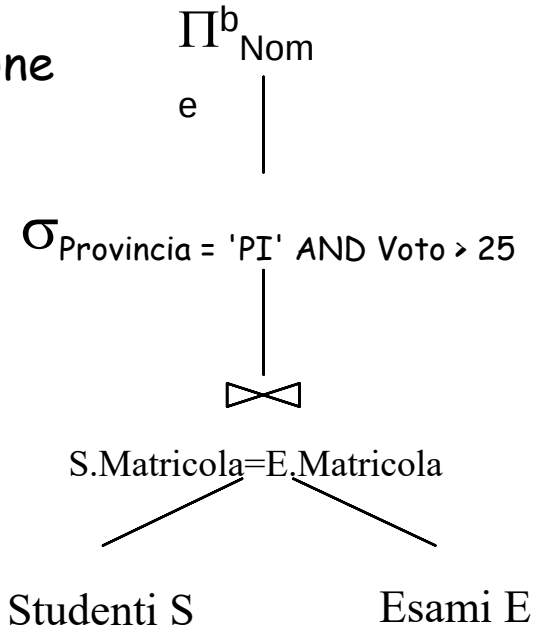
- L'ottimizzazione delle interrogazione è fondamentale nei DBMS.
- E' necessario conoscere il funzionamento dell'ottimizzatore per una buona progettazione fisica.
- Obiettivo dell'ottimizzatore:
  - Scegliere il piano con costo minimo, fra possibili piani alternativi, usando le statistiche presenti nel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

FASI DEL PROCESSO DI OTTIMIZZAZIONE



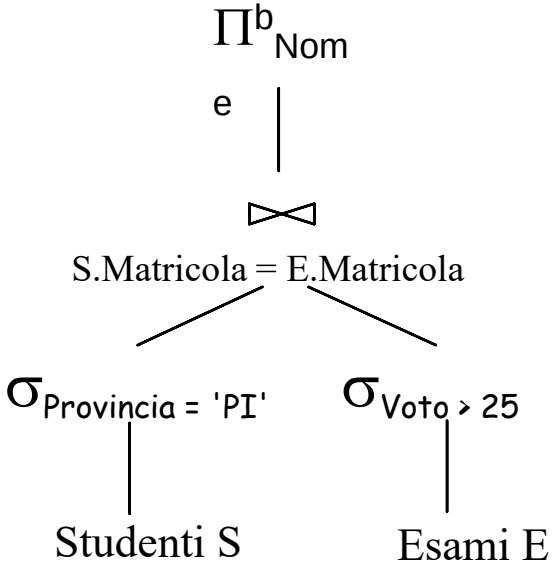

```
SELECT   Nome
FROM     Studenti S, Esami E
WHERE    S.Matricola=E.Matricola AND
         Provincia='PI' AND Voto>25
```

Verifica correttezza del comando, normalizzazione e semplificazione della condizione



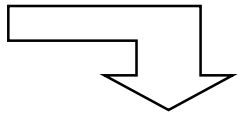
FASI DEL PROCESSO (cont)

Trasformazione dell'albero con regole di equivalenza



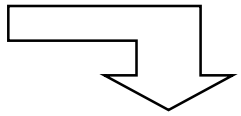
TRASFORMAZIONI INTERESSANTI

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



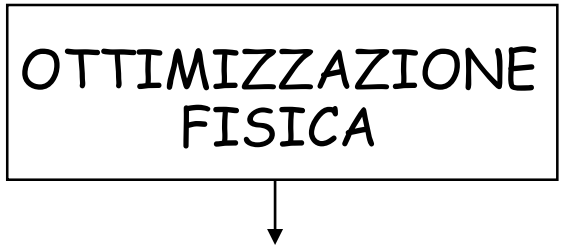
```
SELECT   Matricola, Nome
FROM     Studenti S, Esami E
WHERE    S.Matricola = E.Matricola AND Materia = 'BD';
```

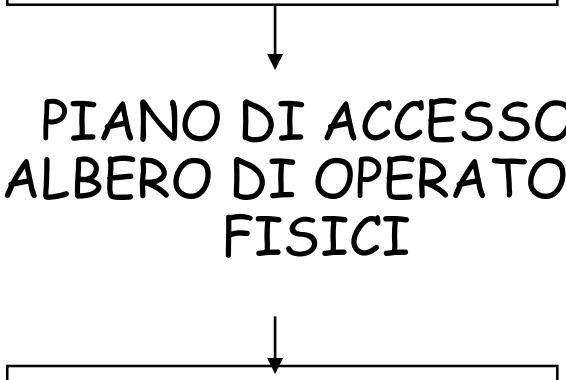
```
SELECT   Matricola, Nome
FROM     VistaStudentiPisani S, VistaEsamiBD E
WHERE    S.Matricola = E.Matricola ;
```

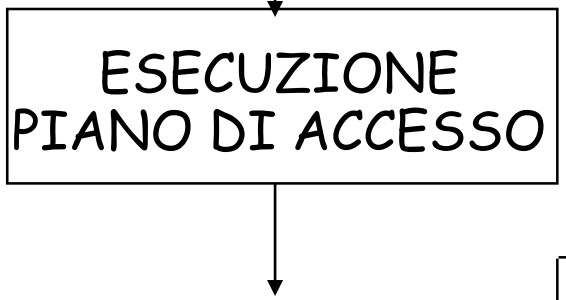


?

FASI DEL PROCESSO (cont.)

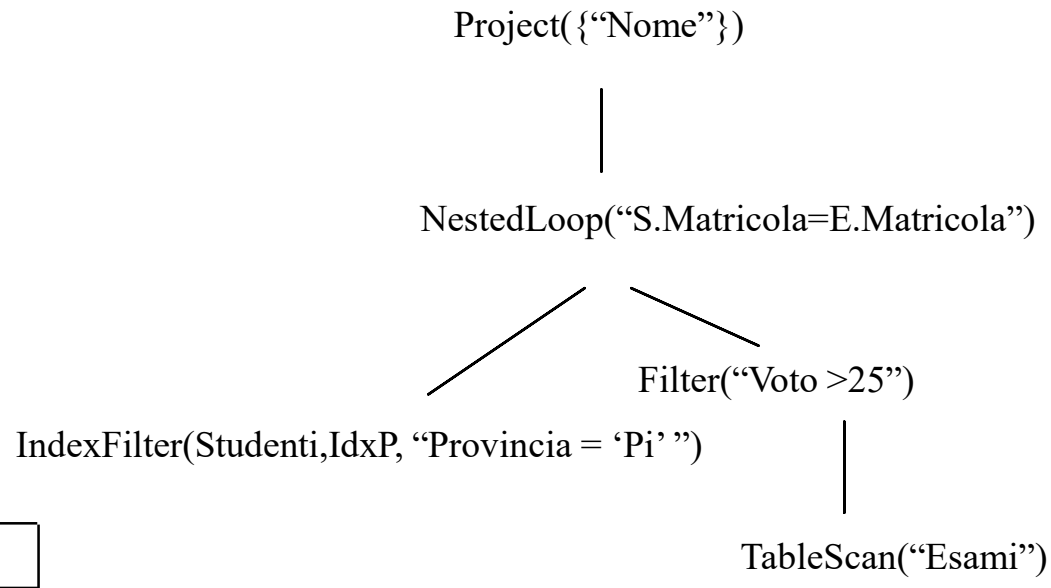






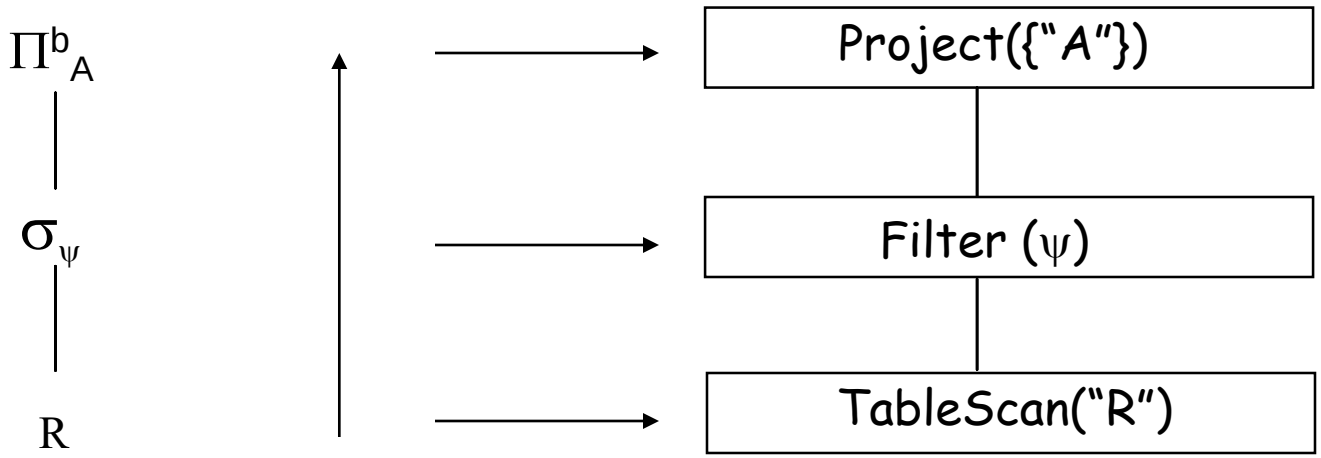
nome
Tonio
Pino

Piano di accesso: scelta dell'algoritmo per eseguire ogni operatore. **Ideale**: Trovare il piano migliore **Euristica**: evitare i piani peggiori!



1) ESEMPIO DI PIANO DI ACCESSO: SFW

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



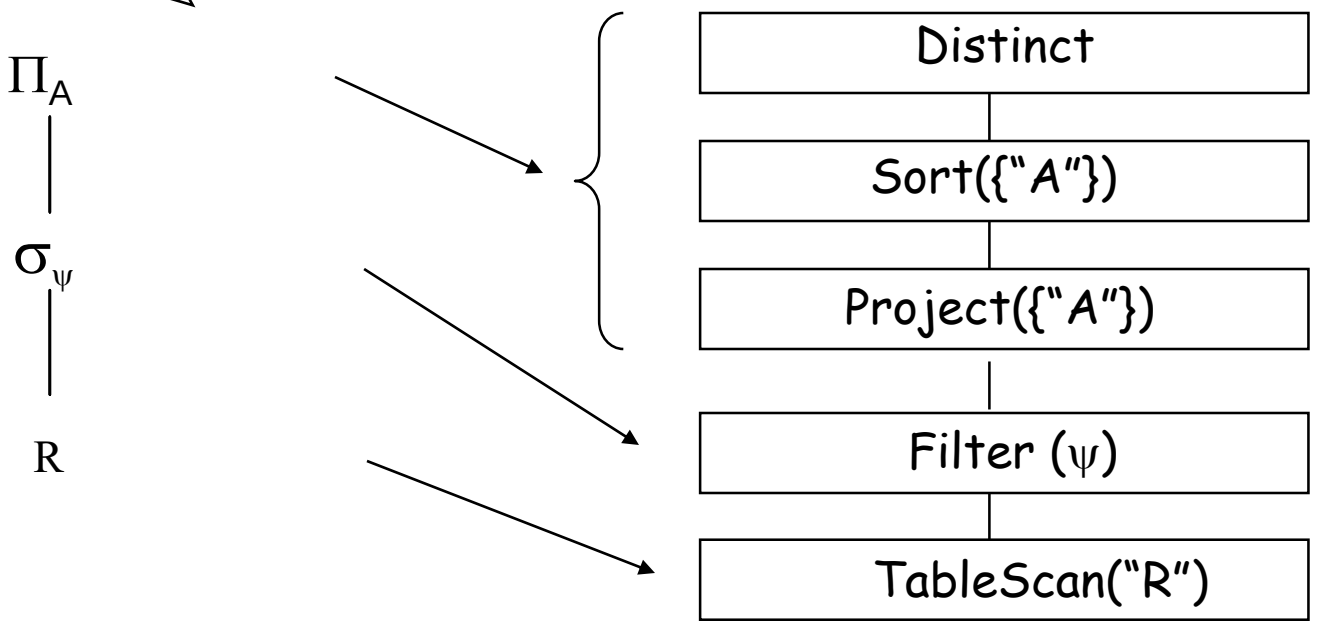
Albero logico

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

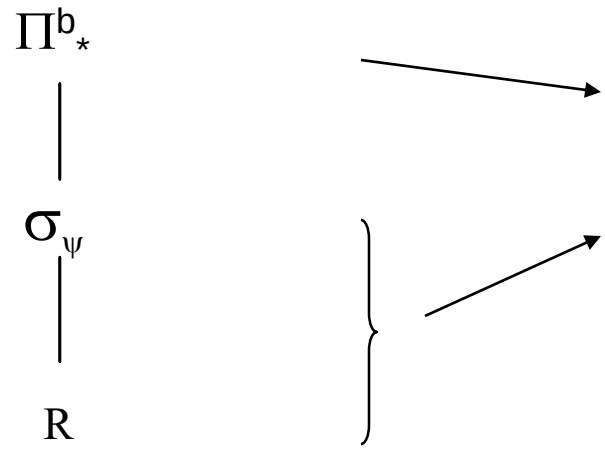
Albero Fisico
(Piano di accesso)

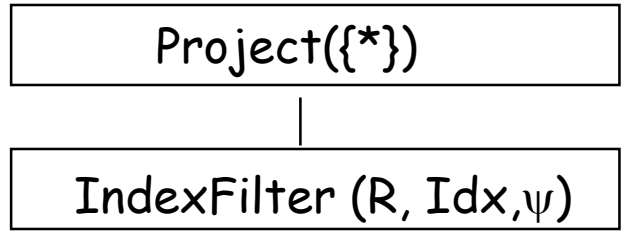
ψ = A **BETWEEN** 50 **AND** 100

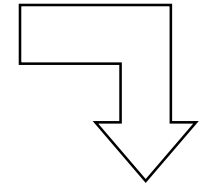
3) ESEMPIO DI PIANO DI ACCESSO CON INDICE

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

con Idx un indice su *A*

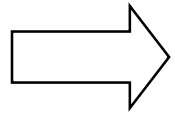






IndexFilter (R, Idx,ψ)

Albero logico



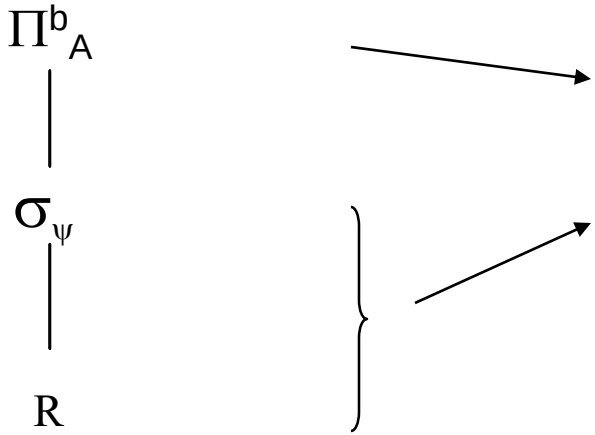
Albero Fisico
(Piano di accesso)

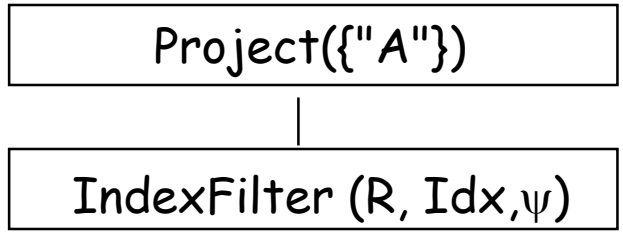
ψ = *A* **BETWEEN** 50 **AND** 100

4) ESEMPIO DI PIANO DI ACCESSO CON INDICE

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

con Idx un indice su A





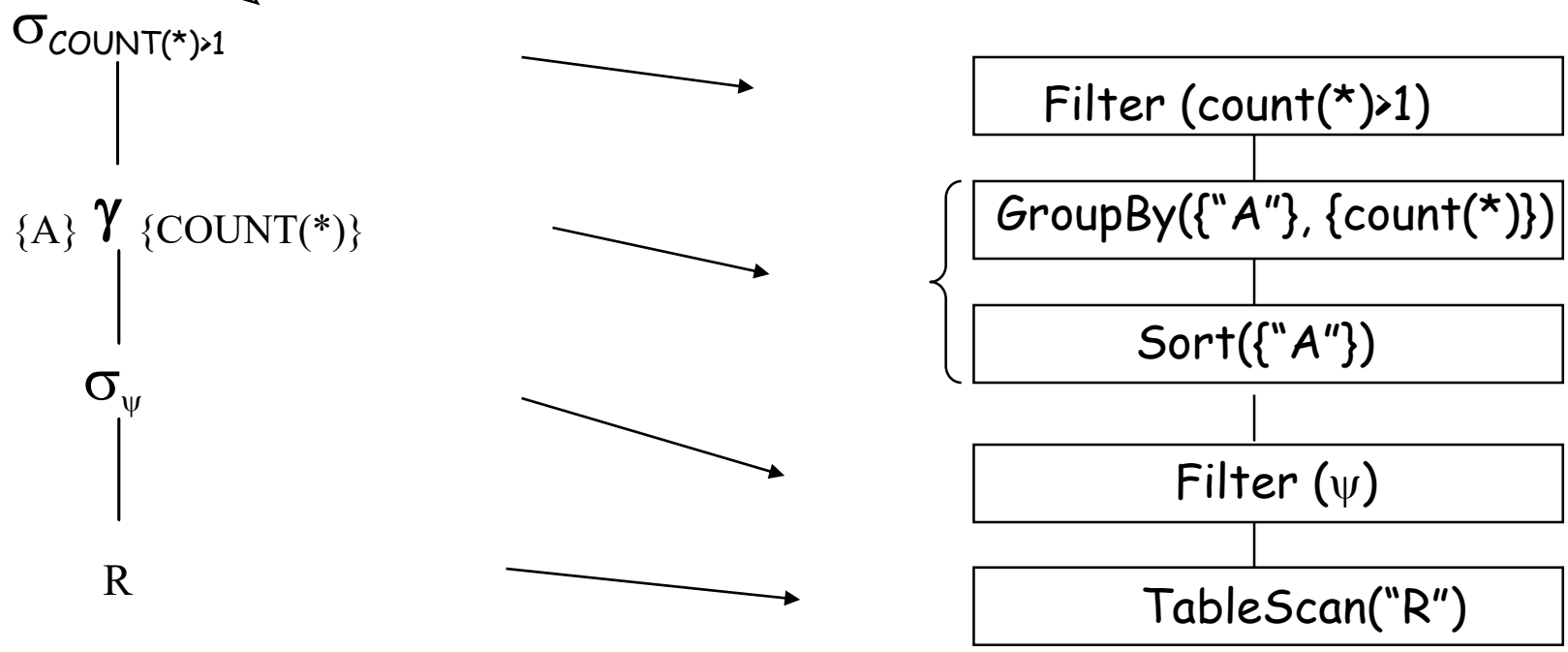
IndexOnlyFilter (R, Idx,ψ)

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;
```

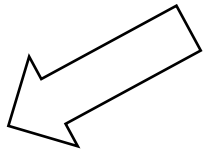


Albero logico

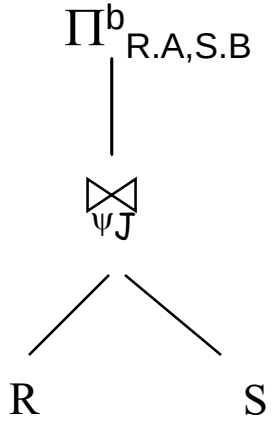
Albero Fisico
(Piano di accesso)

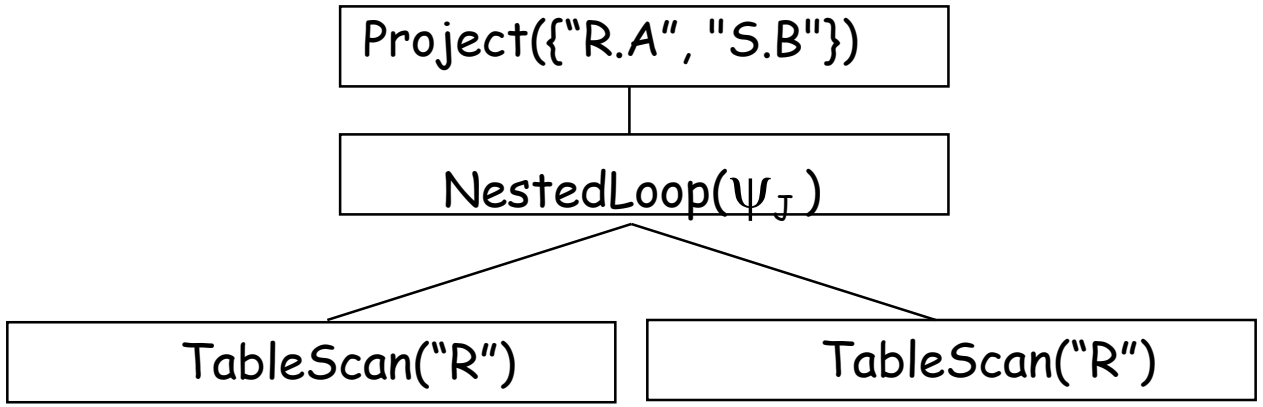
ψ = A **BETWEEN** 50 **AND** 100

6) ESEMPIO DI PIANO DI ACCESSO: GIUNZIONE



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



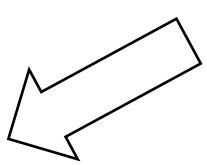


Albero logico

Albero Fisico
(Piano di accesso)

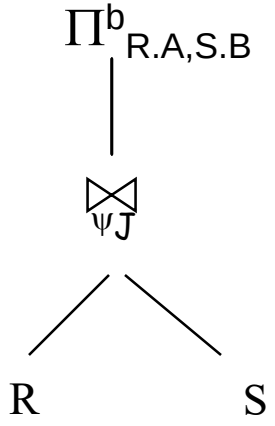
ψ_J = R.A = S.B

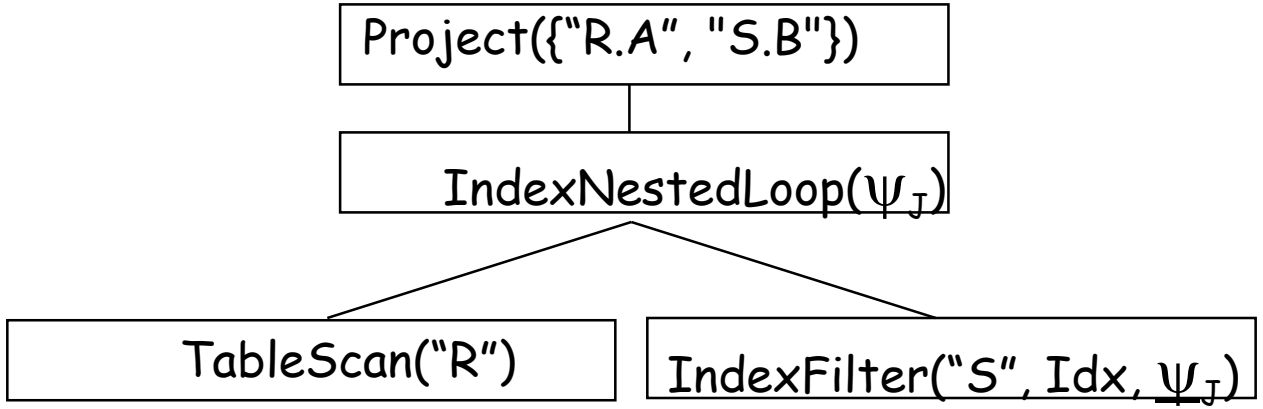
7) ESEMPIO DI PIANO DI ACCESSO: GIUNZIONE CON INDICE



```
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

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;
```

ESERCIZI

7)

```
SELECT   DISTINCT  A, SUM(B)
FROM     R
WHERE    A > 100
GROUP   BY A
HAVING COUNT(*) >1;
```

8)

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

1) Senza indici
2) Con indice su S.Matricola

9)

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

1) Senza indici
2) Con indice su S.Matricola

GESTIONE DELLE TRANSAZIONI

- Una funzionalità essenziale di un DBMS è la protezione dei dati da malfunzionamenti e da interferenze dovute all'accesso contemporaneo ai dati da parte di più utenti.
- La transazione per il programmatore: Una transazione è un programma sequenziale costituito da operazioni che il sistema deve eseguire garantendo:
  - Atomicità, Serializzabilità , Persistenza
  - (**A**tomicity, **C**onsistency, **I**solation, **D**urability - **ACID**)

LA TRANSAZIONE PER IL DBMS

- Una transazione può eseguire molte operazioni sui dati recuperati da una base di dati, ma al DBMS interessano solo quelle di lettura o scrittura della base di dati, indicate con $r_i[x]$ e $w_i[x]$.

- Un dato letto o scritto può essere un record, un campo di un record o una pagina. Per semplicità supporremo che sia una pagina.

- Un'operazione $r_i[x]$ comporta la lettura di una pagina nel buffer, se non già presente.

- Un'operazione $w_i[x]$ comporta l'eventuale lettura nel buffer di una pagina e la sua modifica nel buffer, ma non necessariamente la sua scrittura in memoria permanente. Per questa ragione, in caso di malfunzionamento, si potrebbe perdere l'effetto dell'operazione.

TIPI DI MALFUNZIONAMENTO

- Fallimenti di transazioni: non comportano la perdita di dati in memoria temporanea né persistente (es.: violazione di vincoli, violazione di protezione, stallo)
- Fallimenti di sistema: comportano la perdita di dati in memoria temporanea ma non di dati in memoria persistente (es.: comportamento anomalo del sistema, caduta di corrente)
- Disastri: comportano la perdita di dati in memoria persistente (es.: danneggiamento di periferica)

PROTEZIONE DEI DATI DA MALFUNZIONAMENTI

- Copia della BD.
- Giornale: durante l'uso della BD, il sistema registra nel giornale la storia delle azioni effettuate sulla BD dal momento in cui ne è stata fatta l'ultima copia.
- Contenuto del giornale:
  - (T, begin);
  - Per ogni operazione di modifica:
    - la transazione responsabile;
    - il tipo di ogni operazione eseguita;
    - la nuova e vecchia versione del dato modificato: (T,write, address, oldV, newV);
  - (T, commit) o (T, abort).

PUNTO DI ALLINEAMENTO ("CHECKPOINT")

- Al momento del ripristino, solo gli aggiornamenti più recenti tra quelli riportati sul giornale potrebbero non essere stati ancora riportati sulla base di dati. Come ottenere la certezza che non è necessario rieseguire le operazioni più vecchie?
- Periodicamente si fa un Checkpoint (CKP): si scrive la marca CKP sul giornale per indicare che tutte le operazioni che la precedono sono state effettivamente effettuate sulla BD.
- Un modo (troppo semplice) per fare il CKP: si sospende l'attivazione di nuove transazioni, si completano le precedenti, si allinea la base di dati (ovvero si riportano su disco tutte le pagine "sporche" dei buffer), si scrive nel giornale la marca CKP.

CHECKPOINT

- Si scrive sul giornale una marca di inizio checkpoint che riporta l'elenco delle transazioni attive (BeginCkp, {T1,…,Tn})
- In parallelo alle normali operazioni delle transazioni, il gestore del buffer riporta sul disco tutte le pagine modificate
- Si scrive sul giornale una marca di EndCkp
- La marca di EndCkp certifica che tutte le scritture avvenute prima del BeginCkp ora sono sul disco. Le scritture avvenute tra BeginCkp e EndCkp forse sono sul disco e forse no.

GESTIONE DELL'AFFIDABILITA'

- Gli algoritmi si differenziano a seconda del modo in cui si trattano le scritture sulla BD e la terminazione delle transazioni
  - Disfare–Rifare
  - Disfare–NonRifare
  - NonDisfare–Rifare
  - NonDisfare–NonRifare
- Ipotesi: Le scritture nel giornale vengono portate subito nella memoria permanente!

DISFARE

- Quando si portano le modifiche nella BD ?
  - Politica della modifica *libera* : le modifiche *possono* essere portate nella BD stabile prima che la T termini (disfare o *steal*).
- Regola per poter disfare: prescrittura nel giornale ("Log Ahead Rule" o "Write Ahead Log"):
  - *se la nuova versione di una pagina rimpiazza la vecchia sulla BD stabile prima che la T abbia raggiunto il punto di Commit, allora la vecchia versione della pagina deve essere portata prima sul giornale in modo permanente.*

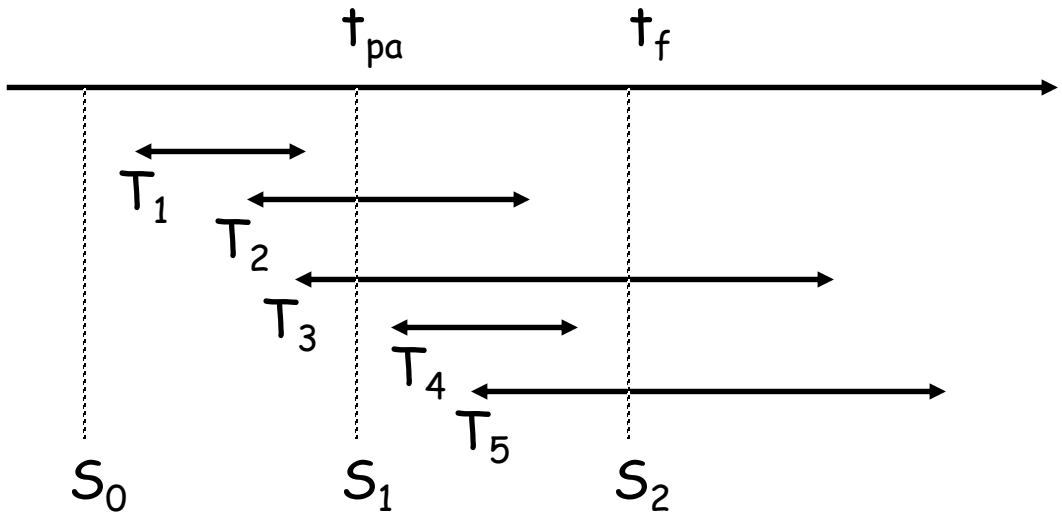
RIFARE

- Come si gestisce la terminazione ?
  - *Commit libero* : una T può essere considerata terminata normalmente prima che tutte le modifiche vengano riportate nella BD stabile (occorre rifare).
- Regola per poter rifare una T: ("Commit Rule")
  - *Le modifiche (nuove versioni delle pagine) di una T devono essere portate stabilmente nel giornale prima che la T raggiunga il Commit (condizione per rifare).*

RIPRESA DAI MALFUNZIONAMENTI (disfare-rifare)

- Fallimenti di transazioni: si scrive nel giornale (T, abort) e si applica la procedura disfare.
- Fallimenti di sistema:
  - La BD viene ripristinata con il comando Restart (ripartenza di emergenza), a partire dallo stato al punto di allineamento, procedendo come segue:
    - Le T non terminate vanno disfatte
    - Le T terminate devono essere rifatte.
- Disastri: si riporta in linea la copia più recente della BD e la si aggiorna rifacendo le modifiche delle T terminate normalmente (ripartenza a freddo).

ESEMPIO



S_0 Stato iniziale
S_1 Stato al punto di allineamento
S_2 Stato persistente al momento del fallimento

- T_1 va ignorata
- T_2 e T_4 vanno rifatte
- T_3 e T_5 vanno disfatte

- L'esecuzione concorrente di transazioni è essenziale per un buon funzionamento del DBMS.
- Il DBMS deve però garantire che l'esecuzione concorrente di transazioni avvenga senza interferenze in caso di accessi agli stessi dati.

T1	tempo	T2
	↓	
begin	t1	-
r[x]	↓	begin
-	t2	r[x]
–	↓	x := x – 800
x := x + 500	t3	-
-	t4	w[x]
w[x]	t5	*Commit*
Commit	t6	
	↓	

SERIALITÀ E SERIALIZZABILITÀ

- **Definizione** Un'esecuzione di un insieme di transazioni $\{T_1, \ldots, T_n\}$ si dice seriale se, per ogni coppia di transazioni T_i e T_j, tutte le operazioni di T_i vengono eseguite prima di qualsiasi operazione T_j o viceversa.

- **Definizione** Un'esecuzione di un insieme di transazioni si dice serializzabile se produce lo stesso effetto sulla base di dati di quello ottenibile eseguento serialmente, in un qualche ordine, le sole transazioni terminate normalmente.

SERIALIZZATORE 2PL STRETTO

- Il gestore della concorrenza (serializzatore) dei DBMS ha il compito di stabilire l'ordine secondo il quale vanno eseguite le singole operazioni per rendere serializzabile l'esecuzione di un insieme di transazioni.
- ***Definizione*** Il protocollo del blocco a due fasi stretto (Strict Two Phase Locking) è definito dalle seguenti regole:
  1. Transazioni diverse non ottengono blocchi in conflitto.
  2. Ogni transazione, prima di effettuare un'operazione acquisisce il blocco corrispondente .
  3. I blocchi si rilasciano alla terminazione della transazione.

CONDIZIONI DI STALLO

- Il problema si può risolvere con tecniche che prevengono queste situazioni (deadlock prevention), oppure con tecniche che rivelano una situazione di stallo e la sbloccano facendo abortire una o più transazioni in attesa (deadlock detection and recovery).