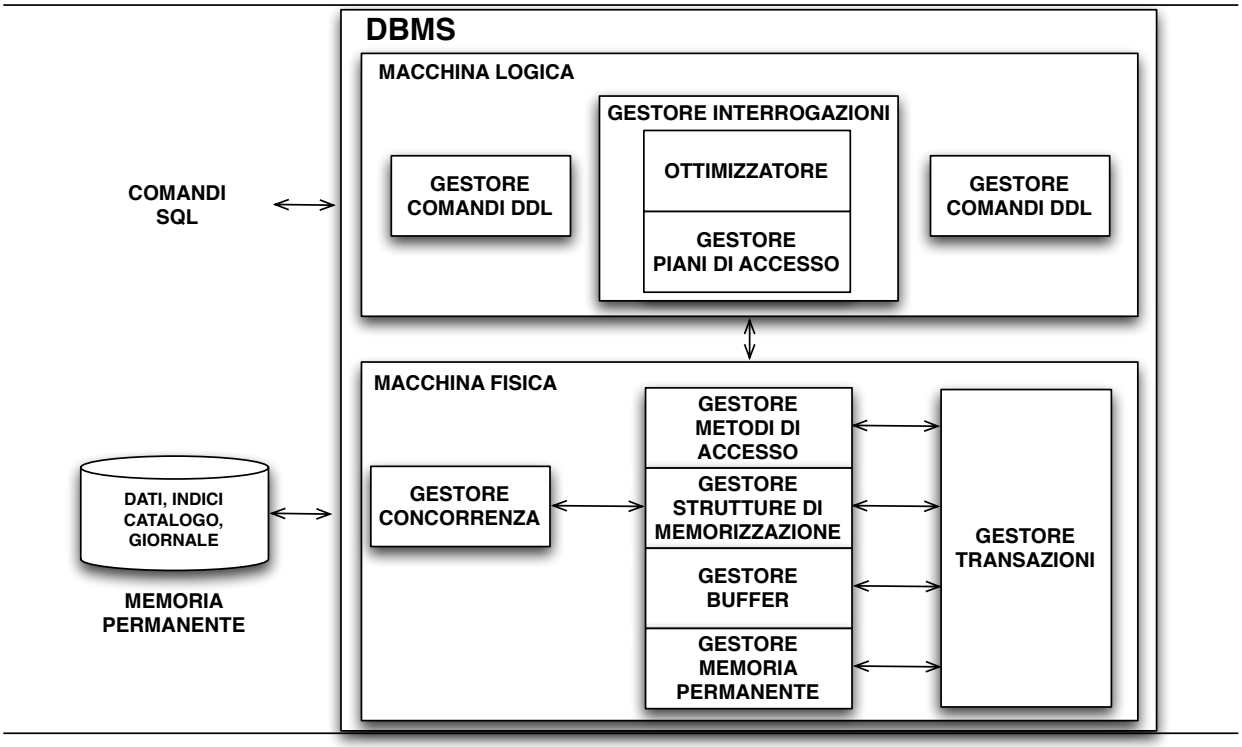
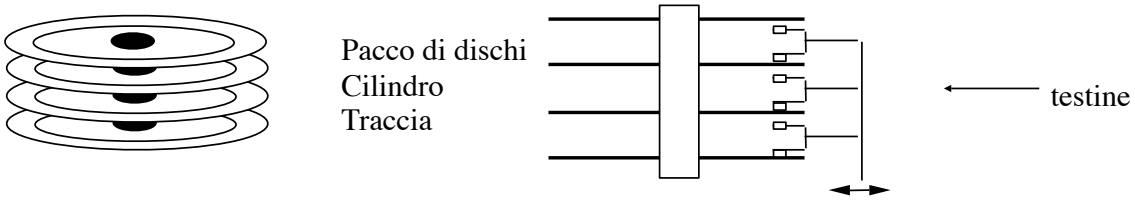
ARCHITETTURA DEI DBMS



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

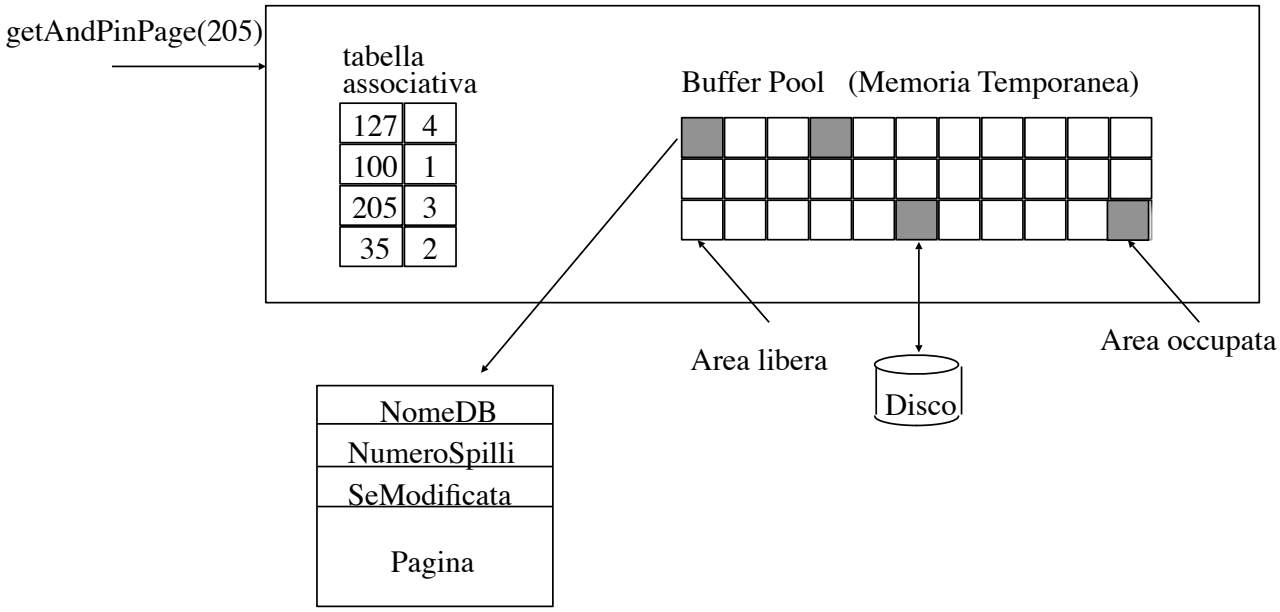
L'IMPORTANZA DELLA LOCALITÀ

- 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

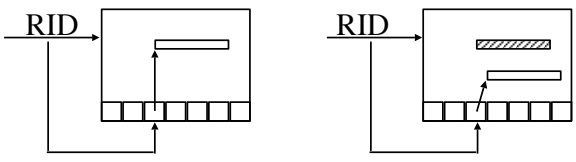
- **GESTORE MEMORIA PERMANENTE**
  - Fornisce un'astrazione della memoria permanente in termini di insiemi di file logici di pagine fisiche di registrazioni (blocchi), nascondendo le caratteristiche dei dischi e del sistema operativo.
- **GESTORE DEL BUFFER**
  - Si preoccupa del trasferimento delle 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**

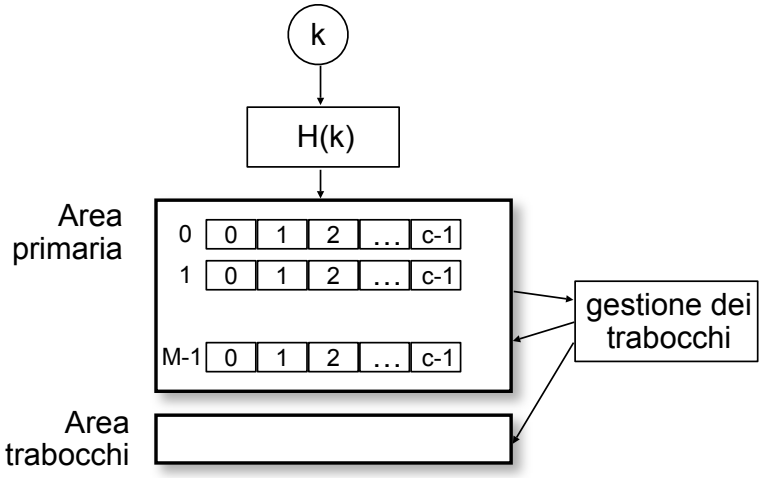
ORGANIZZAZIONI PER CHIAVE

- **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.
  - Gli indici possono essere multi-attributi.
- L'indice è gestito di solito con un'opportuna struttura albero detta **B^+-albero**, la struttura più usata e ottimizzata dai DBMS

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

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

OPERATORI FISICI

- 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 FISICI PER TABELLE E PER ORDINARE

Operatori per tabelle (R):

- **TableScan** (R): per la scansione di R;
- **SortScan** (R, $\{A_i\}$): per la scansione di R ordinata sugli attributi $\{A_i\}$;
- **IndexScan** (R, Idx): per la scansione di R con l'indice Idx sugli attributi $\{A_i\}$;

Operatore per ordinare ($\tau_{\{Ai\}}$):

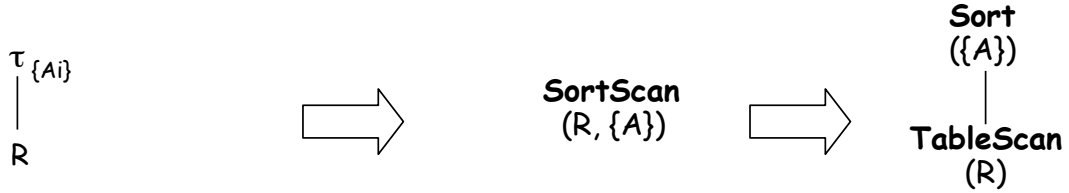
- **Sort** (O, $\{A_i\}$): per ordinare i record di O per valori crescenti degli $\{A_i\}$;

ESEMPI


```
SELECT  *
FROM    R ;
```

Albero logico — Piano di accesso

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



PROIEZIONE

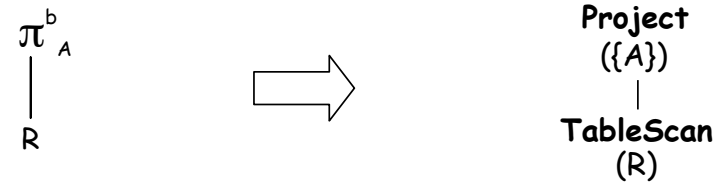

```
SELECT  DISTINCT Provincia
FROM    Studenti;
```

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

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

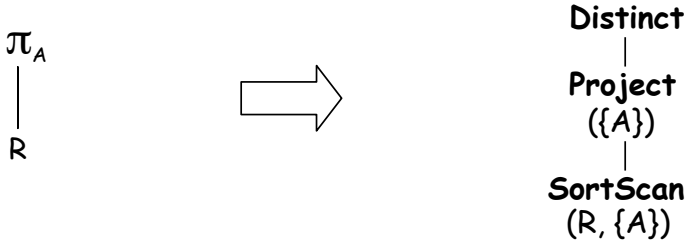
- **Project** (O, $\{A_i\}$): per la proiezione dei record di O senza l'eliminazione dei duplicati;
- **Distinct** (O): per eliminare i duplicati dei record ordinati di O;
- **HashDistinct**(O): per eliminare i duplicati dei record di O;

SELECT *A*
FROM R ;



Albero logico Piano di accesso

SELECT **DISTINCT** *A*
FROM R ;



RESTRIZIONE CON CONDIZIONE SEMPLICE


```
SELECT *
FROM   Studenti
WHERE  Provincia = 'PI';
```

Algoritmo che controlla la condizione su ogni dato della tabella.

Algoritmo che trova i dati che soddisfano la condizione usando un indice.

E nel caso di condizioni complesse?

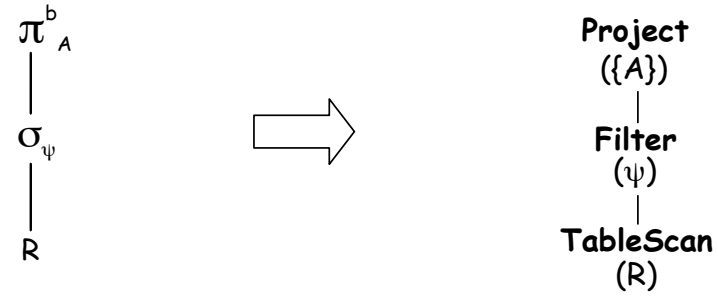
OPERATORI FISICI PER LA RESTRIZIONE

Operatori per (σ_{ψ}):

- **Filter (O, ψ)**: per la restrizione senza indici dei record di O;
- **IndexFilter (R, Idx, ψ)**: per la restrizione con un indice dei record di R;

ESEMPIO DI PIANO DI ACCESSO PER SFW


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



Albero logico — Piano di accesso

ψ = *A* **BETWEEN** 50 **AND** 100

ESEMPIO DI PIANO DI ACCESSO CON INDICE PER SFW


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

con Idx un indice su *A*

Albero logico — Piano di accesso

ψ = *A* **BETWEEN** 50 **AND** 100

OPERAZIONI DI AGGREGAZIONE


```
SELECT   AnnoNascita, COUNT(*)
FROM     Studenti
WHERE    Provincia = 'PI'
GROUP BY AnnoNascita;
```

Approccio basato sull'ordinamento (non è l'unico!):

- Si ordinano i dati sugli attributi del **GROUP BY**, poi si visitano i dati e per ogni gruppo si calcolano le funzioni di aggregazione.

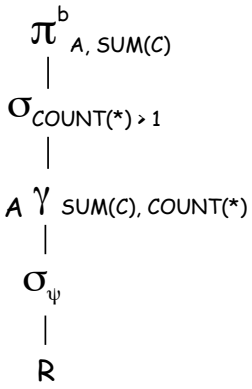
OPERATORI FISICI PER IL RAGGRUPPAMENTO

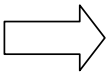
Operatori per $_{\{Ai\}}\gamma_{\{fi\}}$

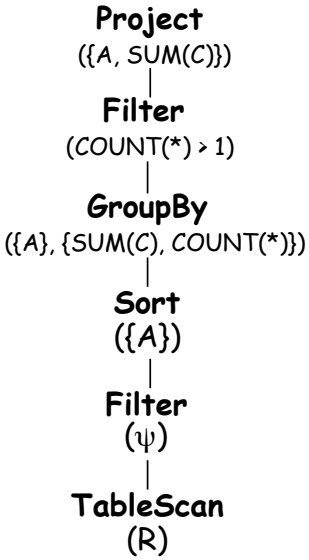
- **GroupBy(O, $\{A_i\}$, $\{f_i\}$):** per raggruppare i record di O sugli $\{A_i\}$ usando le funzioni di aggregazione in $\{f_i\}$.
  - Nell'insieme $\{f_i\}$ vi sono le funzioni di aggregazione presenti nella **SELECT** e nella **HAVING**.
  - L'operatore ritorna record con attributi gli $\{A_i\}$ e le funzioni in $\{f_i\}$.
  - I record di O, e del risultato, sono ordinati sugli $\{A_i\}$

ESEMPIO


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







Albero logico

ψ = A **BETWEEN** 50 **AND** 100

Piano di accesso

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.

L'ALGORITMO NESTED LOOPS


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

Costo: $Npag(R) + Nreg(R) \times Npag(S) \approx Npag(R) \times \frac{Nreg(R)}{Npag(R)} \times Npag(S)$

Con S esterna:

Costo: $Npag(S) + Nreg(S) \times Npag(R) \approx Npag(S) \times \frac{Nreg(S)}{Npag(S)} \times Npag(R)$

Come esterna conviene la relazione con record più lunghi !

ALTRI ALGORITMI

Nested loop con indice:

Si usa quando esiste l'indice ISj sull'attributo di giunzione j della relazione interna S:

```
foreach  r in R do
        foreach s in S with r_i = s_j do
                aggiungi <r, s> al risultato;
```

Sort-merge:

Si usa quando R e S sono ordinate sugli attributi di giunzione ed uno di loro è chiave.

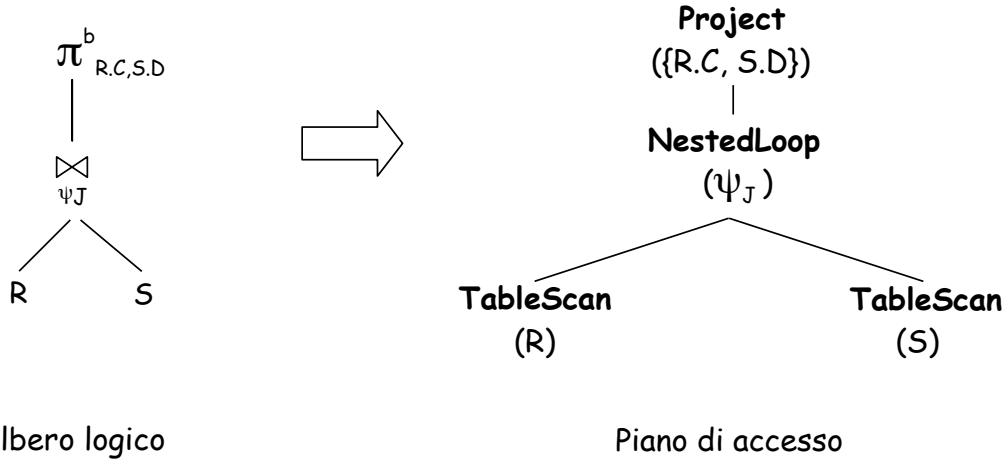
OPERATORI FISICI PER LA GIUNZIONE

Operatori per $\bowtie_{\psi_J}$

- **NestedLoop** (O_E,O_I, ψ_J): per la giunzione con il nested loop e ψ_J la condizione di giunzione;
- **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.

ESEMPIO

SELECT R.C, S.D
FROM R , S
WHERE R.A = S.B;



Albero logico

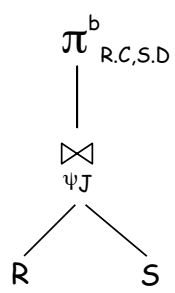
Piano di accesso

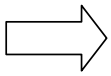
ψ_J = (R.A = S.B)

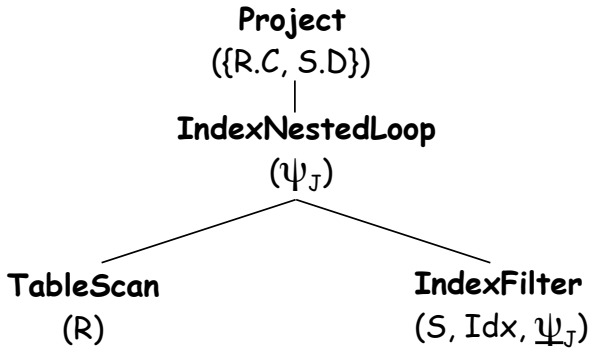
ESEMPIO DI GIUNZIONE CON INDICE


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

con Idx un indice su S.B







Albero logico

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)

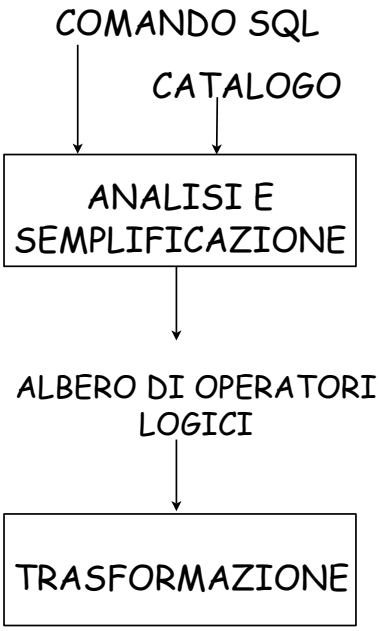
SELECT Matricola, Nome, Materia
FROM Studenti S, Esami E
WHERE S. Matricola = E.Matricola
AND Provincia = 'PI' **AND** Materia = 'BD'

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

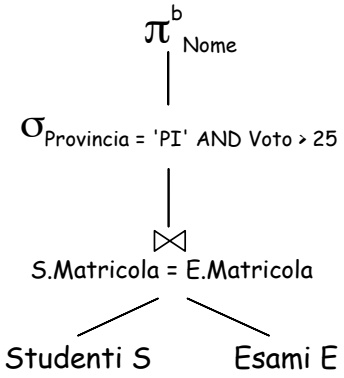
OTTIMIZZATORE DELLE INTERROGAZIONI

- 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.

GESTORE DELLE INTERROGAZIONI

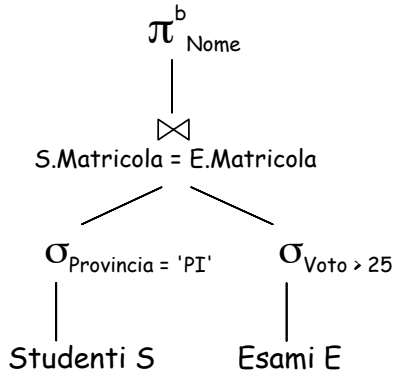



```
SELECT  Nome
FROM    Studenti S, Esami E
WHERE   S.Matricola=E.Matricola AND
        Provincia='PI' AND Voto>25
```



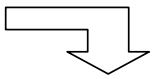
FASI DEL PROCESSO (cont)

Trasformazione dell'albero con regole di equivalenza



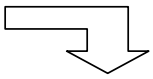
TRASFORMAZIONI INTERESSANTI


```
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

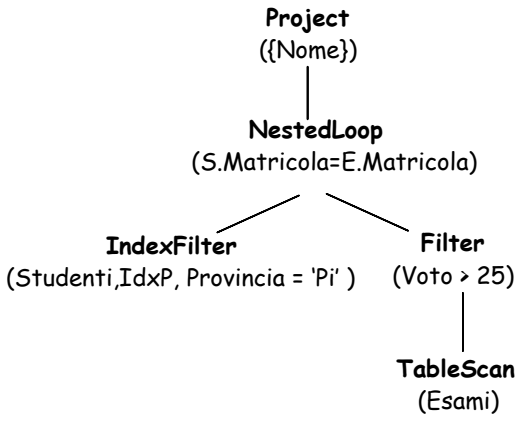
ESECUZIONE PIANO DI ACCESSO

RISULTATO

nome
Tonio
Pino

Ideale: Trovare il piano migliore

Euristica: evitare i piani peggiori!



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();
```


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 esegue garantendo:
  - Atomicità, Serializzabilità , Persistenza
  - (**A**tomicity, **C**onsistency, **I**solation, **D**urability - **ACID**)

LA TRANSAZIONE PER IL DBMS

- Al DBMS interessano solo le operazioni 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**
- **Fallimenti di sistema**
- **Disastri**

PROTEZIONE DEI DATI DA MALFUNZIONAMENTI

- **Copia** della BD.

- **Giornale**: contiene la storia delle azioni effettuate sulla BD dal momento in cui ne è stata fatta l'ultima copia:
  - (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, oldV, newV);
  - (T, commit) o (T, abort).

PUNTO DI ALLINEAMENTO ("CHECKPOINT")

- 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.

GESTIONE DELL'AFFIDABILITÀ

- 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.
- **Regola per poter disfare: Write Ahead Log**

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.

- **Regola per poter rifare**: **Commit Rule**

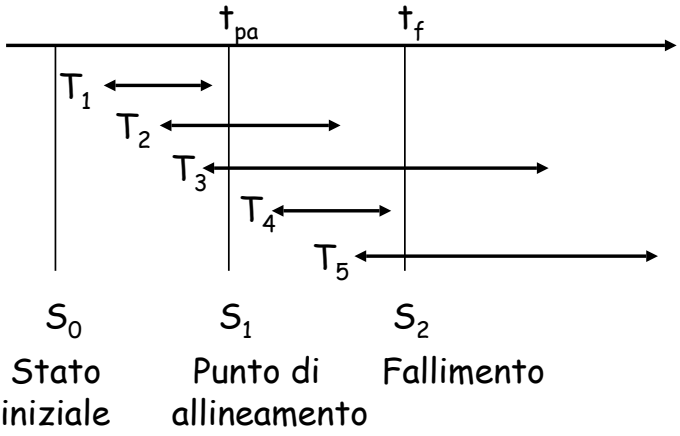
RIPRESA DAI MALFUNZIONAMENTI (disfare-rifare)

- **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



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

GESTIONE DELLA CONCORRENZA

- 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** Una 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**) 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.

- 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 delle transazioni in attesa (**deadlock detection and recovery**).