# Basi Di Dati e di conoscenza

Indici e progettazione fisica

# Contenuti della lezione

- Progettazione fisica
- Organizzazione fisica dei dati
- Indici
- Memoria principale e memoria secondaria
- Definizione degli indici in SQL
- Ottimizzazione delle interrogazioni

# Progettazione di una base di dati

Progettare una base di dati significa definirne struttura, caratteristiche e contenuto. E quindi prevede l'uso di opportune metodologie. In base al grado di astrazione, la progettazione prevede:

- **Modello concettuale**: rappresenta la realta` dei dati e le relazioni tra essi attraverso uno schema
- **Modello logico**: descrive il modo attraverso il quale i dati sono organizzati negli archivi del calcolatore
- **Modello fisico**: descrive come i dati sono registrati nelle memorie di massa


# Le basi di dati vengono interrogate .

- Gli utenti vedono il modello logico (relazionale)
- I dati sono in memoria secondaria
- Le strutture logiche non sarebbe efficienti in memoria secondaria:
  - servono strutture fisiche opportune
- La memoria secondaria è molto più lenta della memoria principale:
  - serve un'interazione fra memoria principale e secondaria che limiti il più possibile gli accessi alla secondaria
- Esempio: una interrogazione con un join

# Organizzazione fisica

- In un DBMS relazionale, i dati sono rappresentati come collezioni di record memorizzati in uno o più file:
- **l'organizzazione fisica dei dati all'interno di un file influenza il tempo di accesso alle informazioni**
- ogni organizzazione fisica dei dati rende alcune operazioni efficienti e altre onerose

# Organizzazione fisica: Efficienza

- L'organizzazione fisica dei dati di un database deve essere efficiente, ciò significa che il sistema di gestione di un database (DBMS) deve avere la capacità di rispondere alle richieste dell' utente il più velocemente possibile.
- **Non esiste un'organizzazione fisica dei dati che sia efficiente per qualunque tipo di lettura e scrittura dei dati**

# Indici

**Dipendente(id, nome, cognome, datanascita, residenza, salario)**

| id | nome | cognome | datanascita | residenza | salario |
|---|---|---|---|---|---|
| 1 | Elena | Rossi | 02/01/1967 | Torino | 2.200,00 |
| 2 | Andrea | Verdi | 04/05/1973 | Como | 1.100,00 |
| 3 | Giulia | Neri | 14/04/1975 | Roma | 2.200,00 |
| 4 | Paolo | Bianchi | 12/08/1970 | Milano | 3.000,00 |
| 5 | Daniele | Bruno | 13/02/1968 | Como | 1.900,00 |
| 6 | Antonio | Bianco | 25/11/1964 | Venezia | 1.700,00 |
| 7 | Lucia | Carta | 09/04/1971 | Alessandria | 2.500,00 |
| 8 | Luca | Draghi | 03/08/1973 | Roma | 2.400,00 |
| 9 | Tania | Bravo | 11/06/1976 | Asti | 1.800,00 |
| 10 | Irene | Massa | 28/04/1979 | Torino | 2.600,00 |
| 11 | Lia | Massa | 15/05/1965 | Milano | 3.500,00 |
| 12 | Alessio | Morra | 19/06/1969 | Como | 1.200,00 |

# Indici

- La tabella Dipendente è memorizzata dal DBMS in un file. Supponiamo di dover eseguire la seguente interrogazione:

```
select * from Dipendente where residenza = 'Como';
```

| id | nome | cognome | datanascita | residenza | salario |
|---|---|---|---|---|---|
| 1 | Elena | Rossi | 02/01/1967 | Torino | 2.200,00 |
| 2 | Andrea | Verdi | 04/05/1973 | Como | 1.100,00 |
| 3 | Giulia | Neri | 14/04/1975 | Roma | 2.200,00 |
| 4 | Paolo | Bianchi | 12/08/1970 | Milano | 3.000,00 |
| 5 | Daniele | Bruno | 13/02/1968 | Como | 1.900,00 |
| 6 | Antonio | Bianco | 25/11/1964 | Venezia | 1.700,00 |
| 7 | Lucia | Carta | 09/04/1971 | Alessandria | 2.500,00 |
| 8 | Luca | Draghi | 03/08/1973 | Roma | 2.400,00 |
| 9 | Tania | Bravo | 11/06/1976 | Asti | 1.800,00 |
| 10 | Irene | Massa | 28/04/1979 | Torino | 2.600,00 |
| 11 | Lia | Massa | 15/05/1965 | Milano | 3.500,00 |
| 12 | Alessio | Morra | 19/06/1969 | Como | 1.200,00 |

# Indici

- Operazioni eseguite per restituire la tabella risultante:
  - lettura sequenziale dell'intero file
  - durante la lettura, selezione dei record dei dipendenti con residenza a Como
  - Visualizzazione dei record

- E' possibile migliorare? Si!

# Indici

- **`Dipendente(id, nome, cognome, datanascita, residenza, salario)->`**

**è ordinata per residenza**

| id | nome | cognome | datanascita | residenza | salario |
|---|---|---|---|---|---|
| D7 | Lucia | Carta | 09/04/1971 | Alessandria | 2.500,00 |
| D9 | Tania | Bravo | 11/06/1976 | Asti | 1.800,00 |
| D2 | Andrea | Verdi | 04/05/1973 | Como | 1.100,00 |
| D5 | Daniele | Bruno | 13/02/1968 | Como | 1.900,00 |
| D12 | Alessio | Morra | 19/06/1969 | Como | 1.200,00 |
| D4 | Paolo | Bianchi | 12/08/1970 | Milano | 3.000,00 |
| D11 | Lia | Massa | 15/05/1965 | Milano | 3.500,00 |
| D3 | Giulia | Neri | 14/04/1975 | Roma | 2.200,00 |
| D8 | Luca | Draghi | 03/08/1973 | Roma | 2.400,00 |
| D1 | Elena | Rossi | 02/01/1967 | Torino | 2.200,00 |
| D10 | Irene | Massa | 28/04/1979 | Torino | 2.600,00 |
| D6 | Antonio | Bianco | 25/11/1964 | Venezia | 1.700,00 |

# Indici

- Operazioni eseguite per restituire la tabella risultante:
  - lettura sequenziale del file fino al primo record con Residenza uguale a Como
  - lettura sequenziale di tutti i record con Residenza uguale a Como, fino al primo record con Residenza diversa da Como
  - Visualizzazione dei record

- Ma questa è un'organizzazione fisica progettata specificatamente per l'operazione

# Indici

- **Vantaggi**:
  - Si evita la lettura sequenziale di tutto il file
- **Svantaggi**:
  - Bisogna mantenere l'ordinamento (operazione costosa)
  - Organizzazione non efficiente con altre interrogazioni che non coinvolgono residenza

# Indici

- E' possibile definire strutture fisiche accessorie che permettano di facilitare l'accesso ai dati

| residenza | locazione |
|---|---|
| Alessandria | |
| Asti | |
| Como | |
| … | |
| Venezia | |

| id | … | residenza |
|---|---|---|
| 1 | | Torino |
| 2 | | Como |
| 3 | | Roma |
| 4 | | Milano |
| 5 | | Como |
| 6 | | Venezia |
| 7 | | Alessandria |
| 8 | | Roma |
| 9 | | Asti |
| 10 | | Torino |
| 11 | | Milano |
| 12 | | Como |

# Indici

- **La struttura fisica** accessoria con accesso associativo ai dati è realizzata sull'attributo residenza (campo chiave)
  - Per ogni valore di residenza tutte le locazioni fisiche dei record corrispondenti al valore del campo chiave

- **La locazione fisica**
  - indica la posizione di un record all'interno di un file
  - permette di accedere direttamente al record d'interesse (alla pagina fisica che lo contiene)

# Indici

- Operazioni eseguite per restituire la tabella risultante:
  - lettura della struttura fisica accessoria per recuperare le locazioni fisiche dei record corrispondenti a Residenza=Como
  - accesso diretto solo ai record del file associati alla Residenza Como
  - Visualizzazione dei record

# Indici

- **Vantaggi**:
  - Si evita la lettura sequenziale di tutto il file
  - Accesso diretto solo ai record di interesse
  - il costo di mantenimento della struttura accessoria è meno oneroso rispetto al costo di mantenimento del file con struttura ordinata
- **Svantaggi**:
  - è necessario spazio supplementare per memorizzare la struttura fisica accessoria
  - possono essere necessarie strutture accessorie per più attributi o combinazioni di attributi (questa è relativa solo ad un attributo)

# Indici

- Gli indici sono le strutture fisiche accessorie offerte dai DBMS per migliorare l'efficienza delle operazioni di accesso ai dati. Possono essere realizzati mediante le strutture fisiche:
  - Alberi
  - Hash table


# Indici di file

- **Indice**:
  - struttura ausiliaria per l'accesso (efficiente) ai record di un file sulla base dei valori di un campo (o di una "concatenazione di campi") detto chiave (o, meglio, pseudochiave, perché non è necessariamente identificante);
- Idea fondamentale: l'indice analitico di un libro: lista di coppie (termine, pagina), ordinata alfabeticamente sui termini, posta in fondo al libro e separabile da esso
- Un indice **I** di un file f è un altro file, con record a due campi: chiave e indirizzo (dei record di f o dei relativi blocchi), ordinato secondo i valori della chiave


# Definizione degli indici in SQL

- Non è standard, ma presente in forma simile nei vari DBMS

  **`create [unique] index`** *IndexName* **`on`** *TableName*`(`*AttributeList*`)`

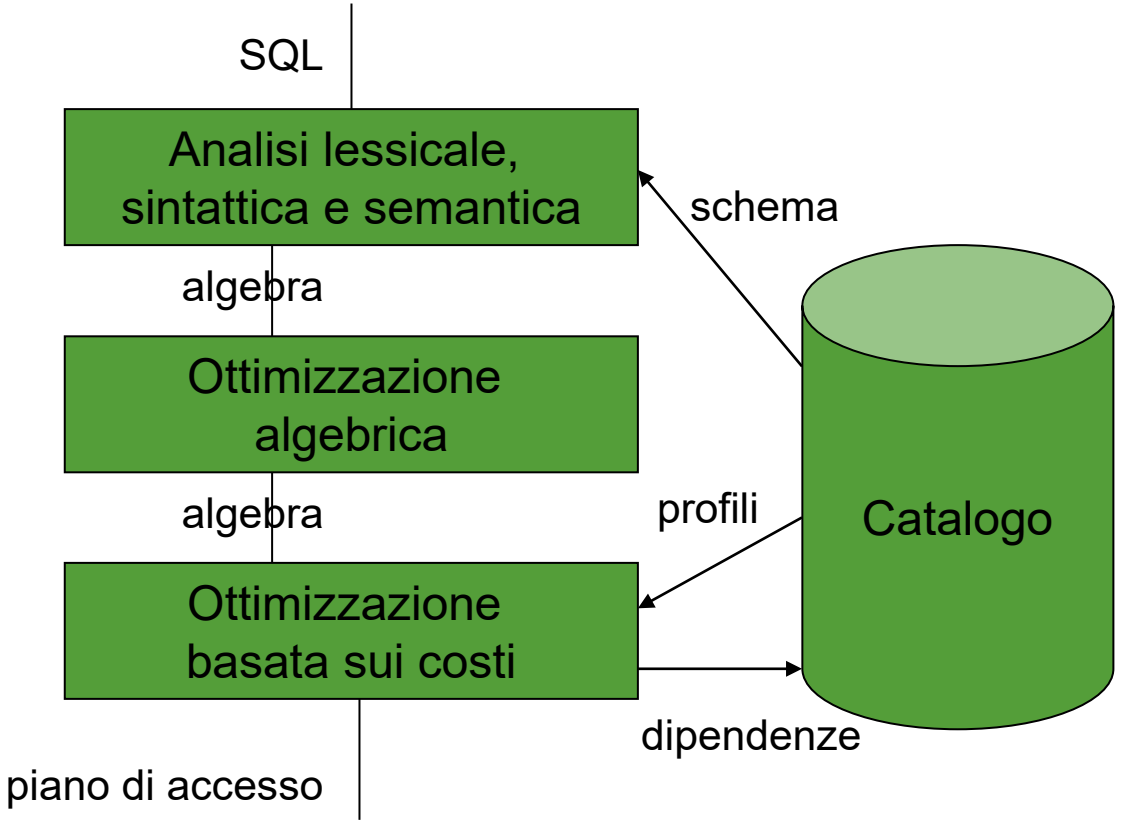
  **`drop index`** *IndexName*

# Esecuzione e ottimizzazione delle interrogazioni


*Query processor* (o *Ottimizzatore*): un modulo del DBMS

- Più importante nei sistemi attuali che in quelli "vecchi" (gerarchici e reticolari):
  - le interrogazioni sono espresse ad alto livello (ricordare il concetto di **indipendenza dei dati**):
    - insiemi di tuple
    - poca proceduralità
  - l'ottimizzatore sceglie la strategia realizzativa (di solito fra diverse alternative), a partire dall'istruzione SQL

# Il processo di esecuzione delle interrogazioni




**PS: L'ottimizzazione agisce a tempo di compilazione**


# "Profili" delle relazioni

- Informazioni quantitative:
  - cardinalità di ciascuna relazione
  - dimensioni delle tuple
  - dimensioni dei valori
  - numero di valori distinti degli attributi
  - valore minimo e massimo di ciascun attributo
- Sono memorizzate nel "catalogo" e aggiornate con comandi del tipo `update statistics`
- Utilizzate nella fase finale dell'ottimizzazione, per stimare le dimensioni dei risultati intermedi


# Ottimizzazione algebrica

- Il termine ottimizzazione è improprio (anche se efficace) perché il processo utilizza euristiche
- Si  basa sulla nozione di equivalenza:
  - Due espressioni sono equivalenti se producono lo stesso risultato qualunque sia l'istanza attuale della base di dati
- I DBMS cercano di eseguire espressioni equivalenti a quelle date, ma meno "costose"
- Euristica fondamentale:
  - selezioni e proiezioni il più presto possibile (per ridurre le dimensioni dei risultati intermedi):
    - "push selections down"
    - "push projections down"


# "Push selections"

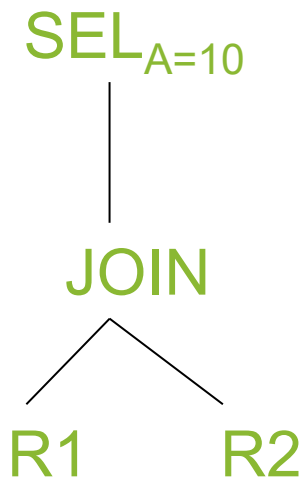
- Assumiamo A attributo di $R_2$

$$\text{SEL}_{A=10}(R_1 \text{ JOIN } R_2) = R_1 \text{ JOIN } \text{SEL}_{A=10}(R_2)$$

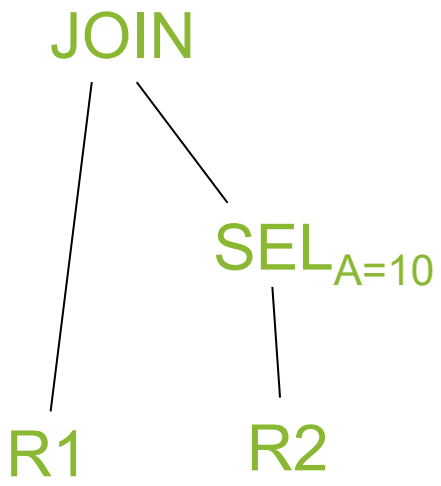
- Riduce in modo significativo la dimensione del risultato intermedio (e quindi il costo dell'operazione)

# Alberi per la rappresentazione di interrogazioni

- $\mathbf{SEL_{A=10}\,(R_1\ JOIN\ R_2)}$




- $R_1$ JOIN $SEL_{A=10}$ ( $R_2$)




# Una procedura euristica di ottimizzazione


- Decomporre le selezioni congiuntive in successive selezioni atomiche
- Anticipare il più possibile le selezioni
- In una sequenza di selezioni, anticipare le più selettive
- Combinare prodotti cartesiani e selezioni per formare join
- Anticipare il più possibile le proiezioni (anche introducendone di nuove)


# Esempio

R1(ABC), R2(DEF), R3(GHI)

```
SELECT     A , E
FROM       R1, R2, R3
WHERE      C=D AND B>100 AND F=G AND H=7 AND I>2
```

- prodotto cartesiano (**FROM**)
- selezione (**WHERE**)
- proiezione (**SELECT**)

```
PROJ AE (SEL C=D AND B>100 AND F=G AND H=7 AND I>2 (
        (R1 JOIN R2) JOIN R3))
```


# Esempio, continua

$$\text{PROJ}_{AE}\ (\text{SEL}_{C=D\ \text{AND}\ B>100\ \text{AND}\ F=G\ \text{AND}\ H=7\ \text{AND}\ I>2}\ (\ (R1\ \text{JOIN}\ R2)\ \text{JOIN}\ R3))$$

- diventa qualcosa del tipo

$$\text{PROJ}_{AE}\ (\text{SEL}_{B>100}\ (R1)\ \text{JOIN}_{C=D}\ R2)\ \text{JOIN}_{F=G}\ \text{SEL}_{I>2}(\text{SEL}_{H=7}(R3)))$$

- oppure

$$\text{PROJ}_{AE}(\ \text{PROJ}_{AEF}((\text{PROJ}_{AC}(\text{SEL}_{B>100}\ (R1)))\ \text{JOIN}_{C=D}\ R2)\ \text{JOIN}_{F=G}\ \text{PROJ}_{G}\ (\text{SEL}_{I>2}(\text{SEL}_{H=7}(R3))))$$


# Join

- L'operazione più costosa
- Vari metodi; i più noti:
  - *nested-loop*, *merge-scan* and *hash-based*


# Il processo di ottimizzazione

- Si costruisce un albero di decisione con le varie alternative ("**piani di esecuzione**")
- Si valuta il costo di ciascun piano
- Si sceglie il piano di costo minore

- L'ottimizzatore trova di solito una "buona" soluzione, non necessarimante l"ottimo"


# Progettazione fisica

- La fase finale del processo di progettazione di basi di dati

- **input**
  - lo schema logico e informazioni sul carico applicativo

- **output**
  - schema fisico, costituito dalle definizione delle relazioni con le relative strutture fisiche (e molti parametri, spesso legati allo specifico DBMS)

# Progettazione fisica nel modello relazionale


- La caratteristica comune dei DBMS relazionali è la disponibilità degli indici:
  - la progettazione fisica spesso coincide con la scelta degli indici (oltre ai parametri strettamente dipendenti dal DBMS)
- Le **chiavi (primarie)** delle relazioni sono di solito coinvolte in selezioni e join: molti sistemi prevedono (oppure suggeriscono) di definire indici sulle chiavi primarie
- Altri indici vengono definiti con riferimento ad altre selezioni o join "importanti"
- Se le prestazioni sono insoddisfacenti, si "tara" il sistema aggiungendo o eliminando indici
- È utile verificare se e come gli indici sono utilizzati con il comando SQL **`show plan`**