



Architetture basi di dati

Richiamo sui DBMS centralizzati

Fonti di riferimento

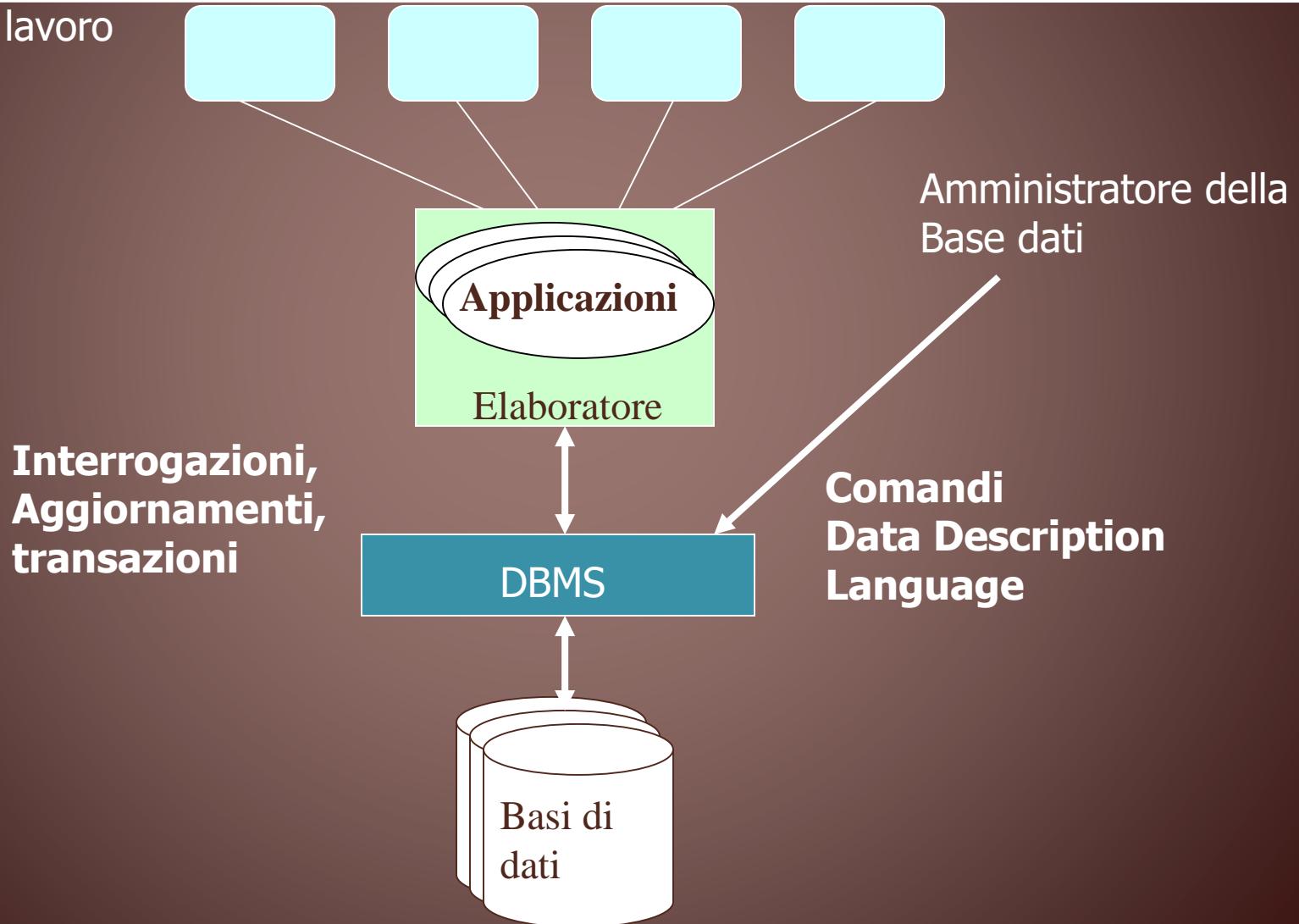
- Per la parte sui DBMS centralizzati
- Basi di Dati (Atzeni, Ceri, Fraternali, Paraboschi, Torlone)

DataBase Management System — DBMS

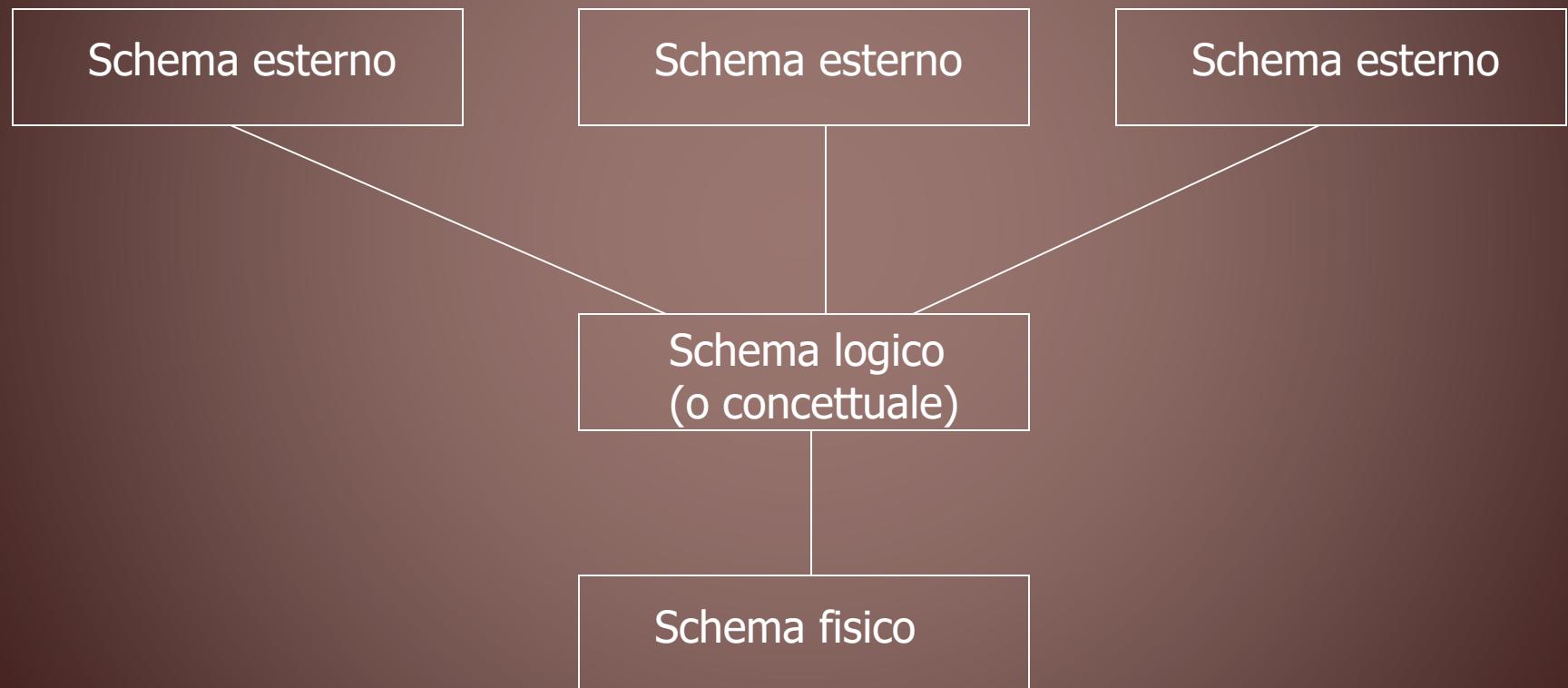
- Sistema (prodotto software) in grado di gestire collezioni di dati che siano:
 - **grandi** - di dimensioni (molto) maggiori della memoria centrale dei sistemi di calcolo utilizzati
 - **persistenti** - con un periodo di vita indipendente dalle singole esecuzioni dei programmi che le utilizzano
 - **condivise** - utilizzate da applicazioni diverse
 - **affidabili** – resistenti ai guasti

Architettura di un DBMS centralizzato

Posti di lavoro



L' **architettura dati** di un DBMS centralizzato e' stata definita dall'ente ANSI/SPARC ed e' a tre livelli



Le caratteristiche di un DBMS centralizzato sono ... 1

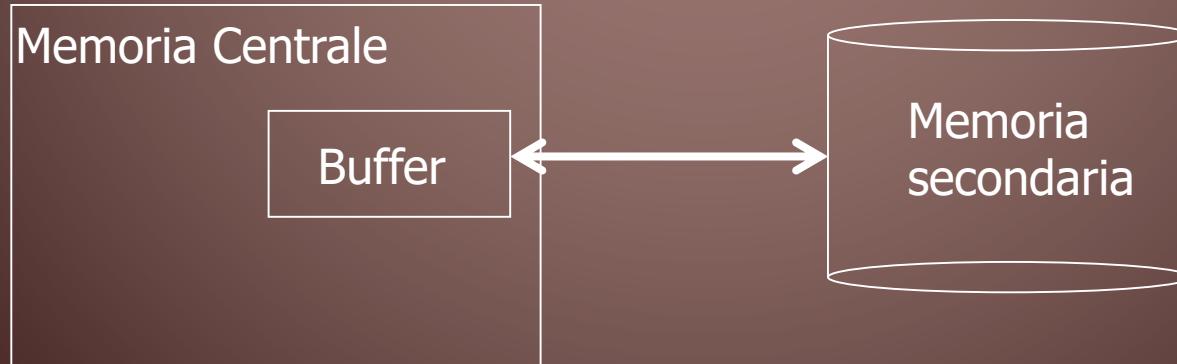
- Unico schema logico, quindi unica semantica (almeno apparentemente, in realta' spesso accade che si diano diversi significati agli stessi termini, ma cosa vuoi dire con "elegante"?)
- Unica base di dati, quindi unico insieme di record interrogati e aggiornati da tutti gli utenti
- → nessuna forma di eterogeneita' concettuale
- Unico schema fisico, quindi unica rappresentazione fisica dei dati
- → nessuna distribuzione e nessuna eterogeneita' fisica

Le caratteristiche di un DBMS centralizzato sono ... 2

- Unico linguaggio di interrogazione, quindi unica **modalita'** di accesso e selezione dei dati di interesse
- Unico sistema di gestione, quindi unica modalità di accesso, aggiornamento e gestione per le transazioni e
- Unica modalità di ripristino a fronte di guasti
- Unico amministratore dei dati
- → nessuna autonomia gestionale

Le basi di dati sono grandi e persistenti

- La persistenza richiede una gestione in memoria secondaria
- La grandezza richiede che tale gestione sia sofisticata (non possiamo caricare tutto in memoria principale e poi riscaricare)
- Le principali azioni nei DBMS riguardano:
- Memorizzazione fisica efficiente delle strutture dati in memoria secondaria
- Scelta efficiente delle pagine da trasferire nel buffer in memoria centrale



Le basi di dati sono condivise - 1

- Una base di dati è una risorsa **integrata** e **condivisa** fra le varie applicazioni
- conseguenze
 - Attività diverse su dati in parte condivisi:
 - meccanismi di **autorizzazione**
 - Attività multi-utente su dati condivisi:
 - controllo della **concorrenza**

Le basi di dati sono condivise - 2

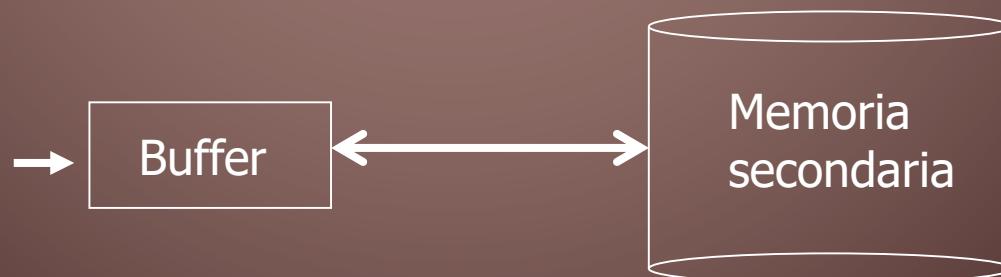
- Esempi:
 - due prelevamenti (quasi) contemporanei sullo stesso conto corrente
 - due prenotazioni (quasi) contemporanee sul posto
- Intuitivamente, le transazioni sono corrette se **seriali** (prima una e poi l'altra)
- Ma in molti sistemi reali l'efficienza sarebbe penalizzata troppo se le transazioni fossero seriali:
 - il **controllo della concorrenza** permette un ragionevole compromesso

Le basi di dati vengono interrogate

...

- Gli utenti vedono il modello logico (relazionale)
- I dati sono in memoria secondaria
- Le strutture logiche non sarebbero 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 limita il più possibile gli accessi alla secondaria
- Esempio: una interrogazione con un join
- E' perciò necessario ottimizzare la esecuzione delle interrogazioni

```
SELECT Nome, Eta'  
FROM Impiegato  
WHERE Eta' > 40
```



Le basi di dati sono affidabili

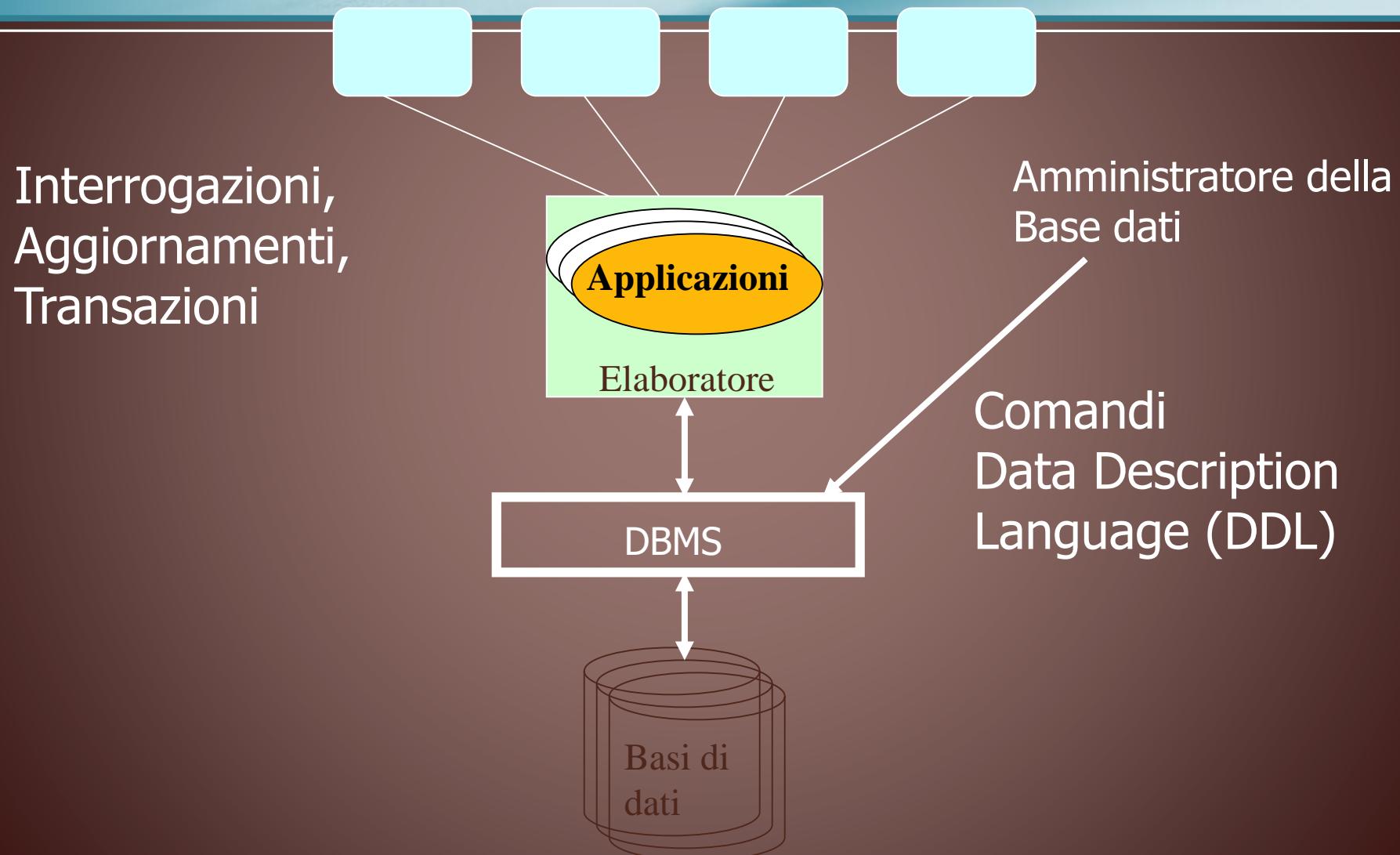
- Le basi di dati sono una risorsa per chi le possiede, e debbono essere conservate anche in presenza di malfunzionamenti
- Esempio:
 - un trasferimento di fondi da un conto corrente bancario ad un altro, con guasto del sistema a metà
- Le transazioni debbono essere
 - atomiche (o tutto o niente)
 - definitive: dopo la conclusione, non si dimenticano gli effetti

Architettura di un DBMS

- Per garantire le precedenti caratteristiche e qualità l'architettura di un DBMS deve essere organizzata in termini di un insieme di funzionalità cooperanti tra loro

Architettura generale di un DBMS

Utenti dai posti di lavoro



Architettura funzionale di un DBMS

Queries

Aggiornamenti

Utente/Applicazione

Transazioni

Amministratore della BD

Comandi DDL

Query compiler

DDL Compiler

Gestore di
Interrogazioni e aggiornamento

Gestore delle
transazioni

Gestore dei
metodi d'accesso

Gestore della
concorrenza

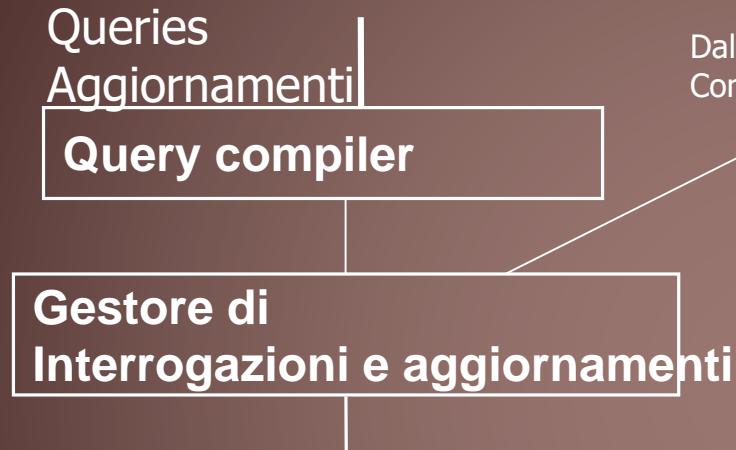
Gestore
del buffer

Gestore della
affidabilità

Gestore della
memoria secondaria

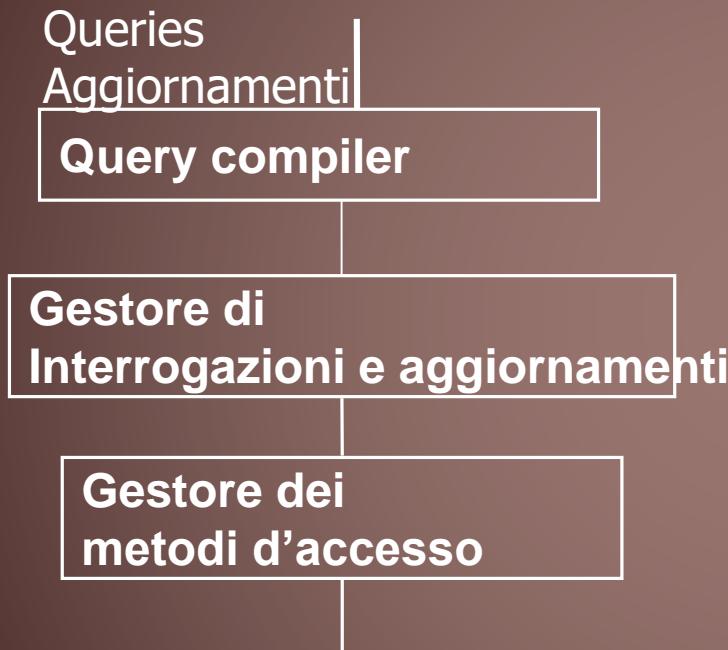


Gestore degli accessi e delle interrogazioni



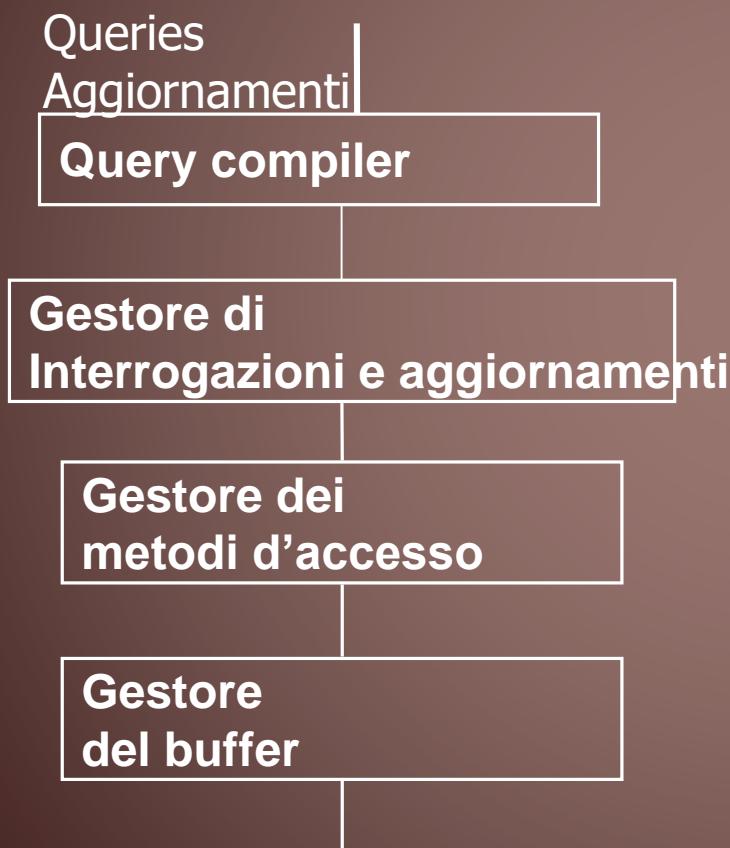
Il Database administrator emette al DDL compiler i comandi del Data Description Language riguardanti la struttura dello schema. Il Query compiler compila le queries e le invia al G. delle interrogazioni che le ottimizza e le frammenta in comandi elementari di accesso, che ...

Gestore degli accessi e delle interrogazioni



.... invia al G. dei metodi di accesso, che li trasforma in comandi di accesso a pagine, che invia al

Gestore degli accessi e delle interrogazioni



Gestore del Buffer,
responsabile della
ottimizzazione della
gestione del buffer, che
le invia al ...

Gestore degli accessi e delle interrogazioni

Queries

Aggiornamenti

Query compiler

**Gestore di
Interrogazioni e aggiornamenti**

**Gestore dei
metodi d'accesso**

**Gestore
del buffer**

**Gestore della
memoria secondaria**



**Gestore della Memoria
secondaria** che le
traduce in accessi a
pagine su disco.

Il tutto deve avvenire
in pochissimo tempo!!

Gestore delle transazioni



Il G. delle transazioni, esegue le transazioni, e garantisce, interagendo con il G. della affidabilità e il G. della concorrenza (o Scheduler), il rispetto delle proprietà transazionali (ACID)

Componenti che sono rivedremo in ambito distribuito

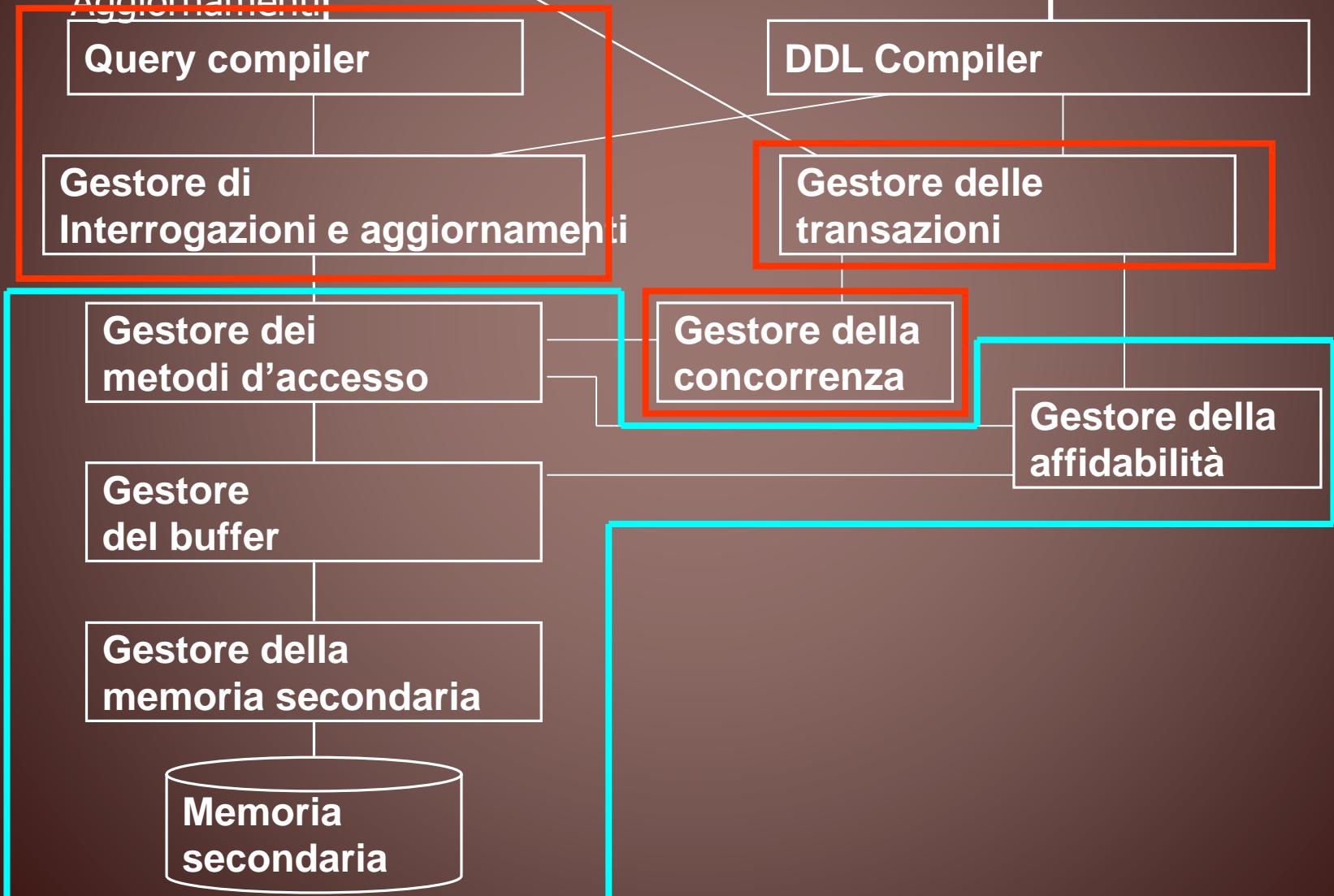
Utente/Applicazione

Amministratore della BD

Queries
Aggiornamenti

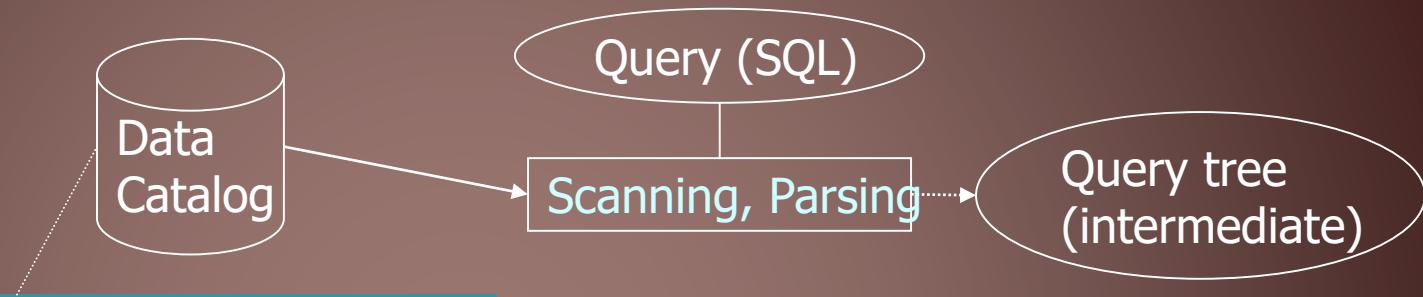
Transazioni

Comandi DDL



Ottimizzazione delle interrogazioni

Passi di elaborazione: parsing



Usa informazioni sullo schema logico

Effettua una analisi lessicale, sintattica e semantica, usando il dizionario e traduzione in algebra relazionale, corrispondente a un query tree

Passi di elaborazione: costruzione query plan logico e sua ottimizzazione



Trasforma il query tree in un query plan logico contenente operazioni di algebra relazionale sulle tabelle logiche dello schema, e lo ottimizza rispetto al costo

Query plan logico

- La query viene rappresentata come un albero nel quale:
 - Le foglie corrispondono alle strutture dati logiche (tabelle) e
 - I nodi intermedi rappresentano operazioni algebriche:
 - Selezione
 - Proiezione
 - Join
 - Prodotto cartesiano
 - Operazioni insiemistiche

Esempio: query sul seguente schema

- Employee [Ename, Iname, **ssn** (social security number), bdate (birthdate), address]
- Department [Dname, **dnumber**, mgrssn (manager ssn)]
- Project [Pname, **pnumber**, plocation, **dnum**]
- Works-on [**Essn** (Employee ssn), pno (**pnumber**), hours]
- Trovare Progetti localizzati a Stafford e loro Manager, con nome, indirizzo, e data nascita

Query in SQL

- Employee [Ename, Lname, ssn (social security number), bdate (birthdate), address]
- Department [Dname, dnumber, mgrssn (manager ssn)]
- Project [Pname, pnumber, plocation, dnum]

Trovare Progetti localizzati a Stafford e loro Manager

SELECT Pnumber, Dnum, Lname, Address, Bdate

FROM Project P, Dept D, Emp E

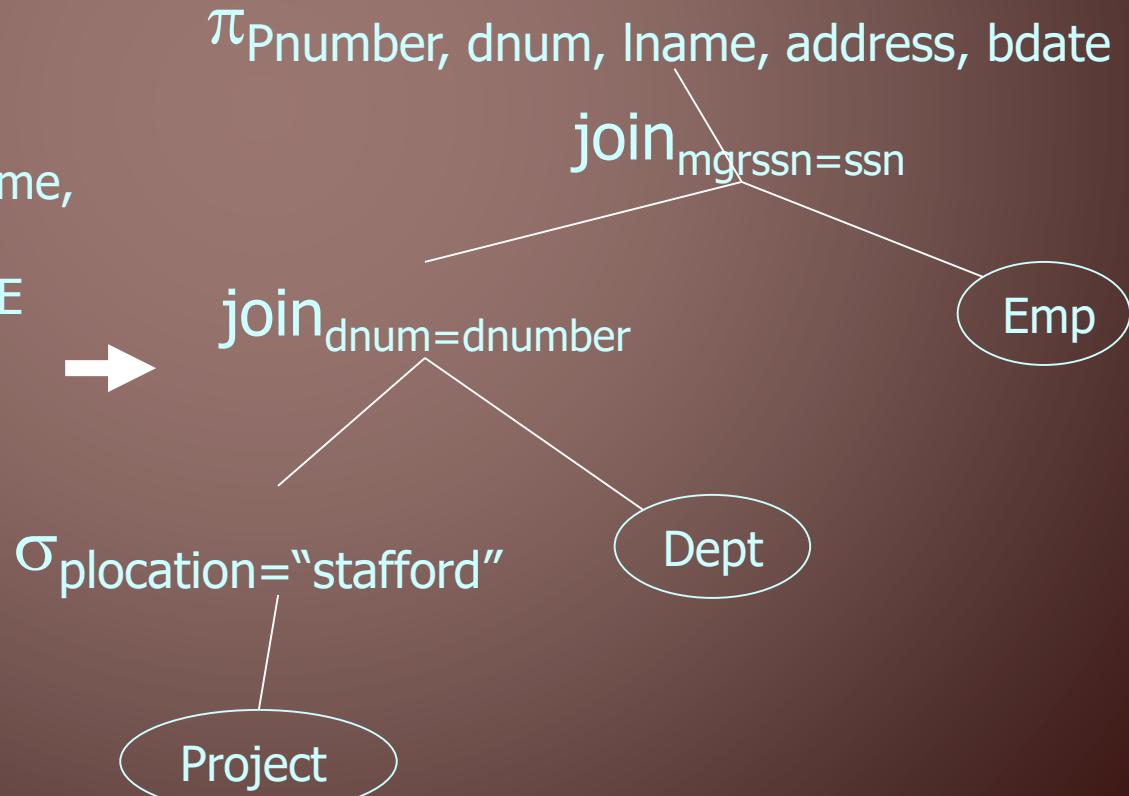
**Where P.dnum=D.dnumber and E.ssn = D.mgrssn
and P.location = 'Stafford'**

Corrispondente query plan logico

$R1 = (\sigma_{\text{plocation}=\text{"stafford"}}(\text{project})) \text{ join}_{\text{dnum}=\text{dnumber}} \text{dept}$

$R = (\pi_{\text{Pnumber}, \text{dnum}, \text{Lname}, \text{address}, \text{bdate}} (R1 \text{ join}_{\text{mgrssn}=\text{ssn}} \text{emp}))$

SELECT Pnumber, Dnum, Lname,
Address, Bdate
FROM Project P, Dept D, Emp E
Where P.dnum=D.dnumber
and E.ssn = D.mgrssn
and P.location = 'Stafford'



Ottimizzazione del query plan logico

- 1

- A una interrogazione SQL possono corrispondere diverse espressioni in algebra relazionale e quindi diversi query plan logici.
- Nel precedente esempio posso esprimere nell'algebra le clausole della
- Where P.dnum=D.dnumber and E.ssn = D.mgrssn and P.location = 'Stafford'
- secondo diversi ordini possibili

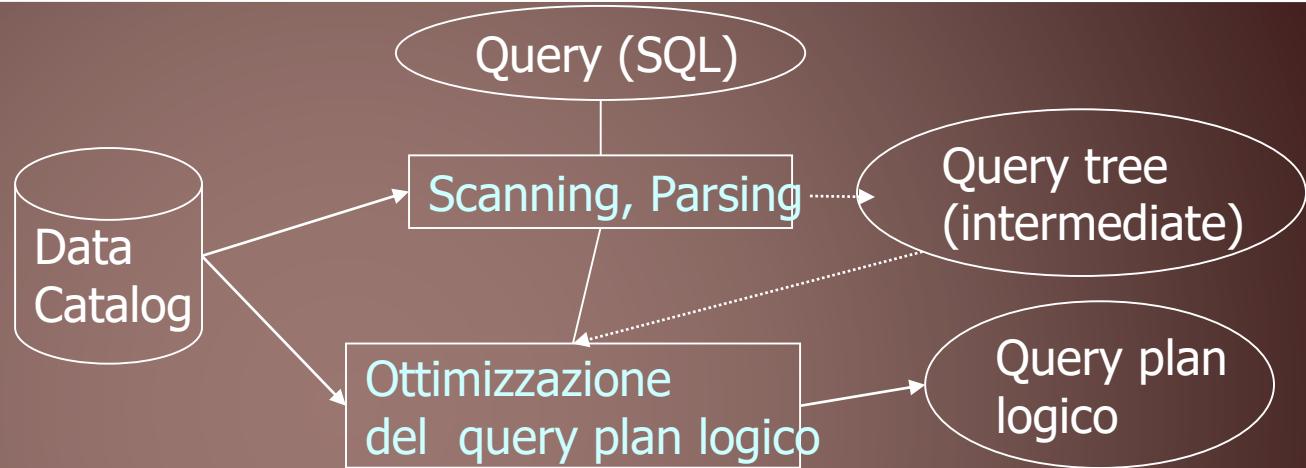
Ottimizzazione del query plan logico

- 2

In questa fase viene scelto, tra tutti i piani logici equivalenti, il query plan logico che ottimizza il costo di esecuzione

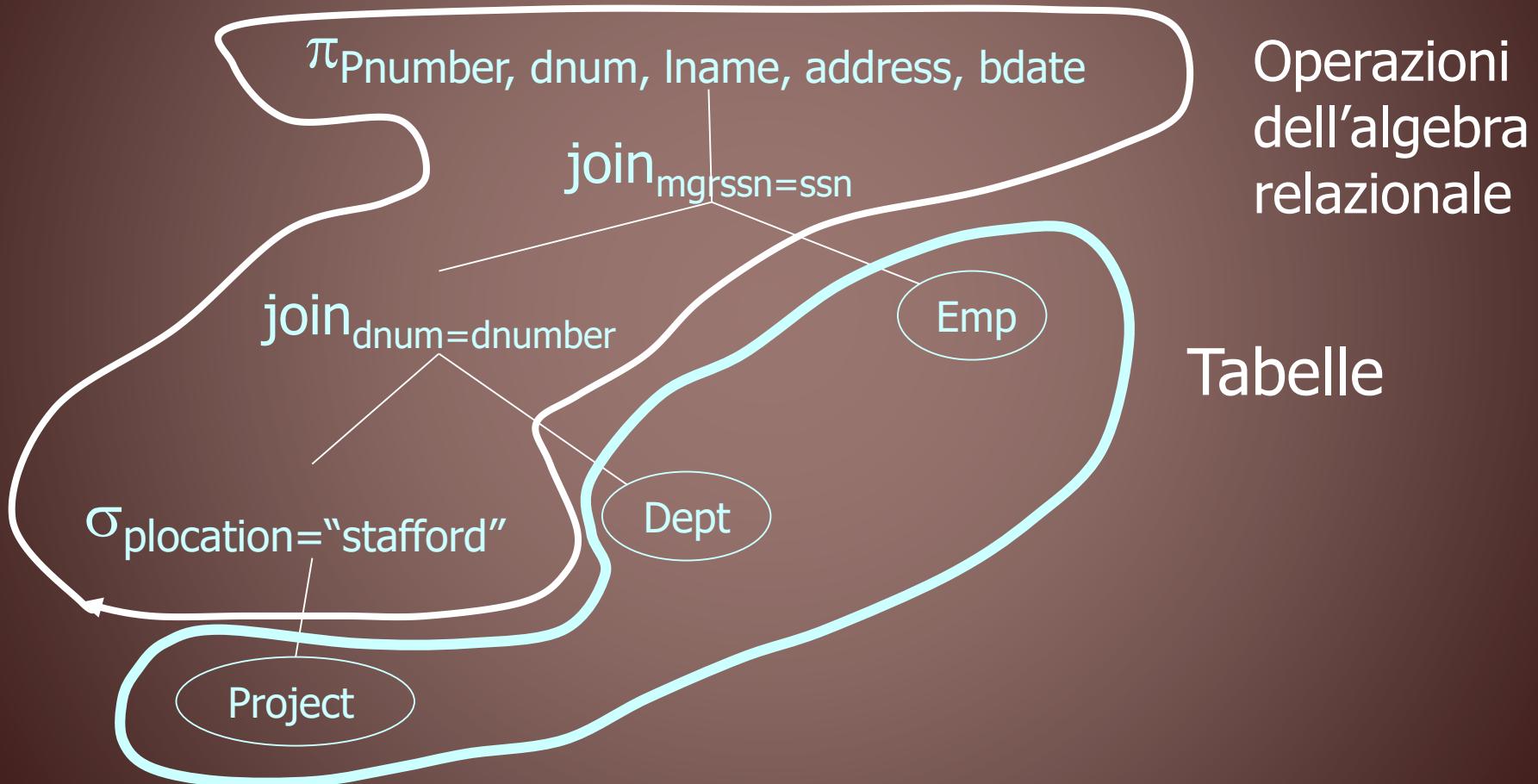
Ad esempio, in genere conviene che le selezioni siano anticipate rispetto ai join, perche' in questo modo i join operano su tabelle di minore dimensione.

Ottimizzazione del query plan logico - 3



Ottimizzazione algebrica:
Trasformazione delle espressioni di Algebra
relazionale rappresentate dal query plan logico
in altre **equivalenti**, ma piu' efficienti

Rappresentazione iniziale del query plan



Passi di elaborazione: query plan fisico e sua ottimizzazione

Usa statistiche sulla storia delle esecuzioni delle interrogazioni



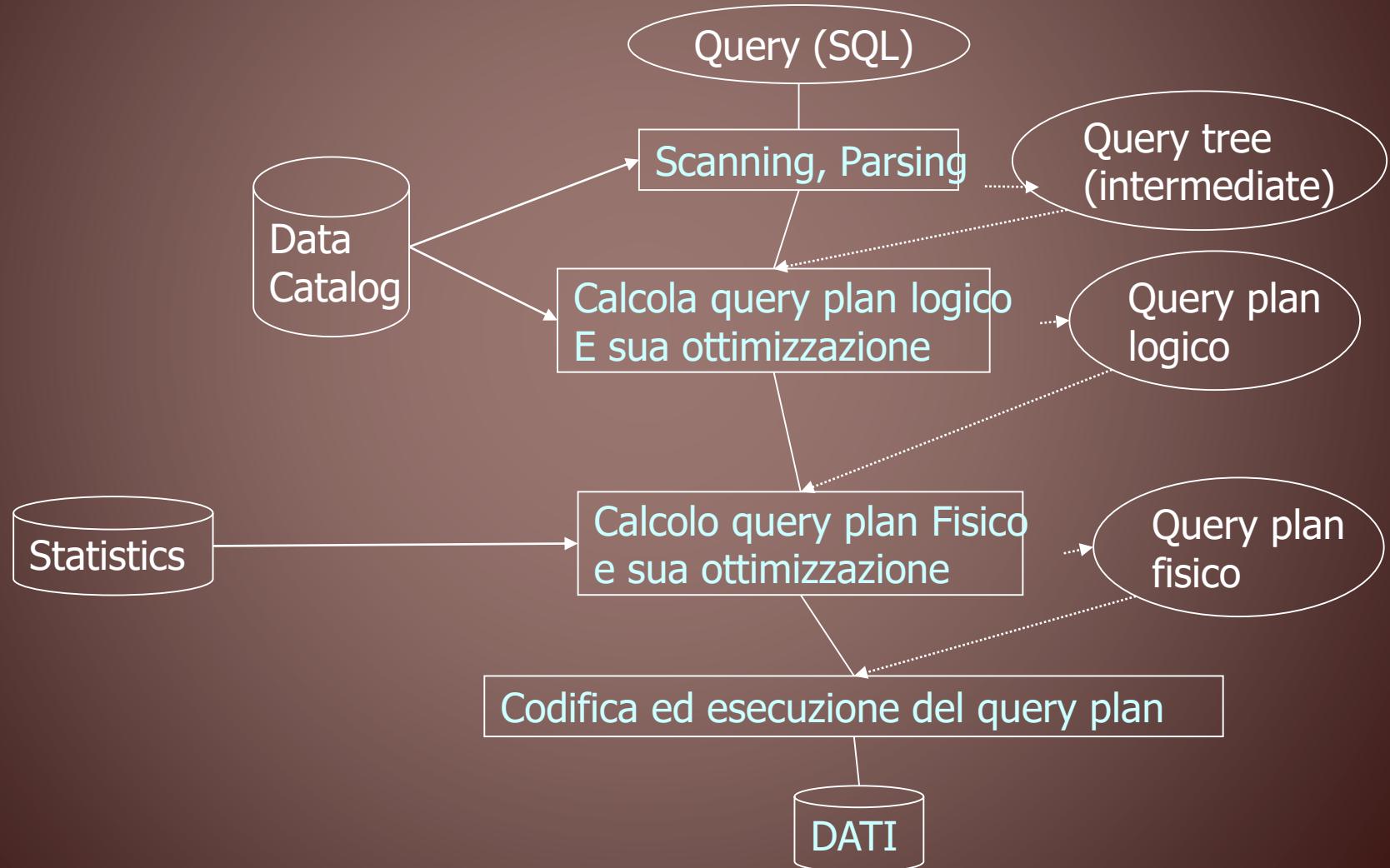
Trasformazione delle tabelle logiche in strutture fisiche e metodi di accesso alla memoria

Trasformazione delle operazioni algebriche nelle loro implementazioni sulle strutture fisiche

Ottimizzazione del query plan fisico basata sui costi

- La trasformazione e' effettuata mediante una strategia che prevede di utilizzare:
 - 1. proprieta' algebriche;
 - 2. una stima dei costi delle operazioni fondamentali per diversi metodi di accesso;
- In generale, il problema di ottimizzare la query ha complessita' esponenziale
- In pratica, si introducono delle approssimazioni ragionevoli in base a euristiche

Passi di elaborazione: esecuzione del query plan



Transazioni

Definizione di transazione

- Insieme di istruzioni di lettura e scrittura sulla base di dati
- Eventualmente inserite in un linguaggio programmatico
- L'insieme delle istruzioni gode di alcune proprietà che garantiscono la corretta esecuzione in un ambiente concorrente e non affidabile

Definizione di transazione

- Transazione: parte di programma caratterizzata da un inizio (**begin-transaction**, **start transaction** in SQL), una fine (**end-transaction**, non esplicitata in SQL) e al cui interno deve essere eseguito una e una sola volta uno dei seguenti comandi
 - **commit work** per terminare correttamente
 - **rollback work** per abortire la transazione
- Un **sistema transazionale (OLTP)** e' in grado di definire ed eseguire transazioni per conto di un certo numero di applicazioni concorrenti

Differenza fra applicazione e transazione

PROGRAMMA
APPLICATIVO



Una transazione

```
start transaction;  
  
update ContoCorrente  
    set Saldo = Saldo + 10 where  
        NumConto = 12202;  
  
update ContoCorrente  
    set Saldo = Saldo - 10 where  
        NumConto = 42177;  
  
commit work;
```

Una transazione con varie decisioni

```
start transaction;
update ContoCorrente
    set Saldo = Saldo + 10 where NumConto =
12202;
update ContoCorrente
    set Saldo = Saldo - 10 where NumConto =
42177;
select Saldo into A
    from ContoCorrente
    where NumConto = 42177;
if (A>=0)  then commit work
                else rollback work;
```

Sintassi SQL per specificare le transazioni

- Le istruzioni commit work e rollback work possono comparire più volte all'interno del programma
- Durante l'esecuzione della transazione è necessario che esattamente una istanza delle due istruzioni sia eseguita
 - Commit work se la transazione è avvenuta correttamente
 - Rollback (abort) e la transazione non deve essere completata
 - Il sistema deve ripristinare lo STATO precedente
- Approccio binario **NON SONO AMMESSE SITUAZIONI INTERMEDIE**

Il concetto di transazione

- Una unità di elaborazione che gode delle proprietà "ACID"
 - Atomicità
 - Consistenza
 - Isolamento
 - Durata (persistenza)

Atomicità

- Una transazione è una unità atomica di elaborazione
- Non può lasciare la base di dati in uno stato intermedio
 - un guasto o un errore prima del commit debbono causare l'annullamento (UNDO) delle operazioni svolte
 - un guasto o errore dopo il commit non deve avere conseguenze; se necessario vanno ripetute (REDO) le operazioni
- Esito
 - Commit = caso "normale" e più frequente (99% ?)
 - Abort (o rollback)
 - richiesto dall'applicazione = suicidio
 - requested dal sistema (violazione dei vincoli, concorrenza, incertezza in caso di fallimento) = omicidio

Consistenza

- La transazione rispetta i vincoli di integrità
- Conseguenza:
 - se lo stato iniziale e' corretto
 - anche lo stato finale e' corretto
- Durante l'esecuzione ci possono essere delle violazioni, ma non devono restare alla fine della transazioni
 - Nel caso rollback work

Isolamento/Concorrenza

- La transazione non risente degli effetti delle altre transazioni concorrenti
 - l'esecuzione concorrente di una collezione di transazioni deve produrre un risultato che si potrebbe ottenere con una esecuzione sequenziale
- Conseguenza: una transazione non espone i suoi stati intermedi
 - Si evita l' "effetto domino"

Durabilità (Persistenza)

- Gli effetti di una transazione andata in commit non vanno perduti ("durano per sempre"), anche in presenza di guasti
 - Commit significa impegno
- Come vedremo questo implica la presenza di un componente nel DBMS che garantisca l'affidabilità (recovery manager)

Gestore degli accessi e delle interrogazioni

Gestore delle transazioni



Isolamento??



Gestore della concorrenza

Schedule

- Una sequenza di esecuzione di un insieme di transazioni e' detta **schedule**
- Es e' uno schedule la sequenza
- T1: trova posto, T1: alloca posto; **T2: trova posto; T2 alloca posto**
- Una schedule e' seriale se una transazione termina prima che la successiva inizi
 - Il precedente è un esempio di schedule seriale
- Lo schedule seguente NON è seriale
- T1: trova posto, **T2: trova posto;** T1: alloca posto; **T2 alloca posto**

Serializzabilità

- La proprietà di isolamento:
- Ogni transazione esegue come se non ci fosse concorrenza
- coincide con la proprietà per cui:
 - un insieme di transazioni eseguite concorrentemente produce lo stesso risultato che produrrebbe una (qualsiasi) delle possibili esecuzioni sequenziali delle stesse transazioni.
- Ne basta una!
- Quella esecuzione è proprio la esecuzione che garantisce l'isolamento (... come se fosse eseguita da sola)

Serializzabilità

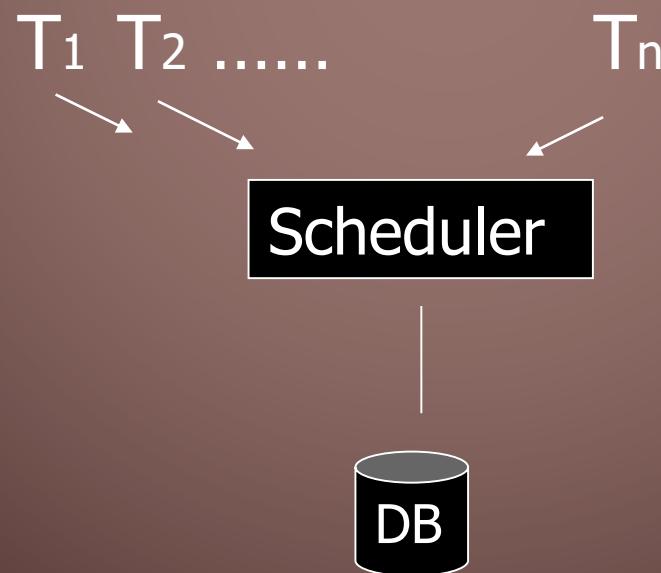
- Uno schedule e' serializzabile se l'esito della sua esecuzione e' lo stesso che si avrebbe se le transazioni in esso contenute fossero eseguite in una (*qualsiasi*) sequenza seriale

Possibili sistemi per il controllo della concorrenza

- I diversi algoritmi per il controllo della concorrenza
 - Controllo basato su Conflict Equivalence
 - Controllo di concorrenza basato su locks
 - Protocollo 2PL o Two phase locking
 - Shared locks
 - Gestione dei deadlock
 - Controllo di concorrenza basato su timestamps

Controllo di concorrenza tramite locks

- Idea: definizione di un protocollo che garantisca a priori la conflict serializability
 - Richiede la presenza di uno **scheduler** nell'architettura del DBMS

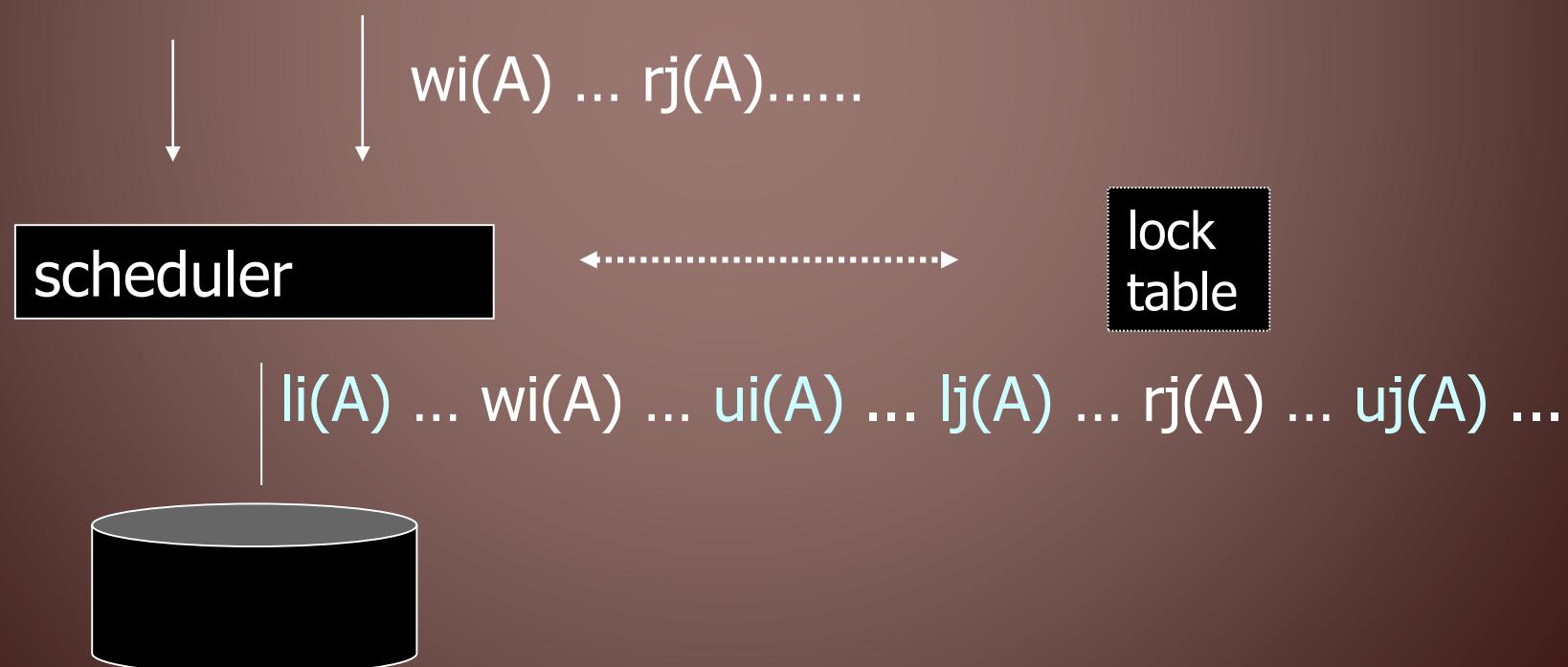


Primitive di lock

- Introduciamo due nuove operazioni, che richiedono l'utilizzo ed il rilascio, per ora, esclusivo di una risorsa:
 - Lock (esclusivo): $l_i(A)$ → Chiedo la risorsa in modo esclusivo
 - Unlock: $u_i(A)$ → Rilascio la risorsa
- Le primitive di lock vengono aggiunte alle transazioni e inserite all'interno delle sequenze di operazioni $w_i(A)$, $r_j(A)$

Compito dello scheduler

- Trasformare le transazioni aggiungendo i comandi di lock, sulla base della conoscenza delle assegnazioni precedenti, in modo da garantire la serializzabilità
- Memorizzare tali assegnazioni in una tabella di lock



Two-Phase Locking

Aggiungiamo l'ulteriore terza regola di **Two Phase Locking (2PL)**:

In ogni transazione tutte le richieste di lock precedono tutti gli unlock

