# Basi di Dati

prof. A. Longheu

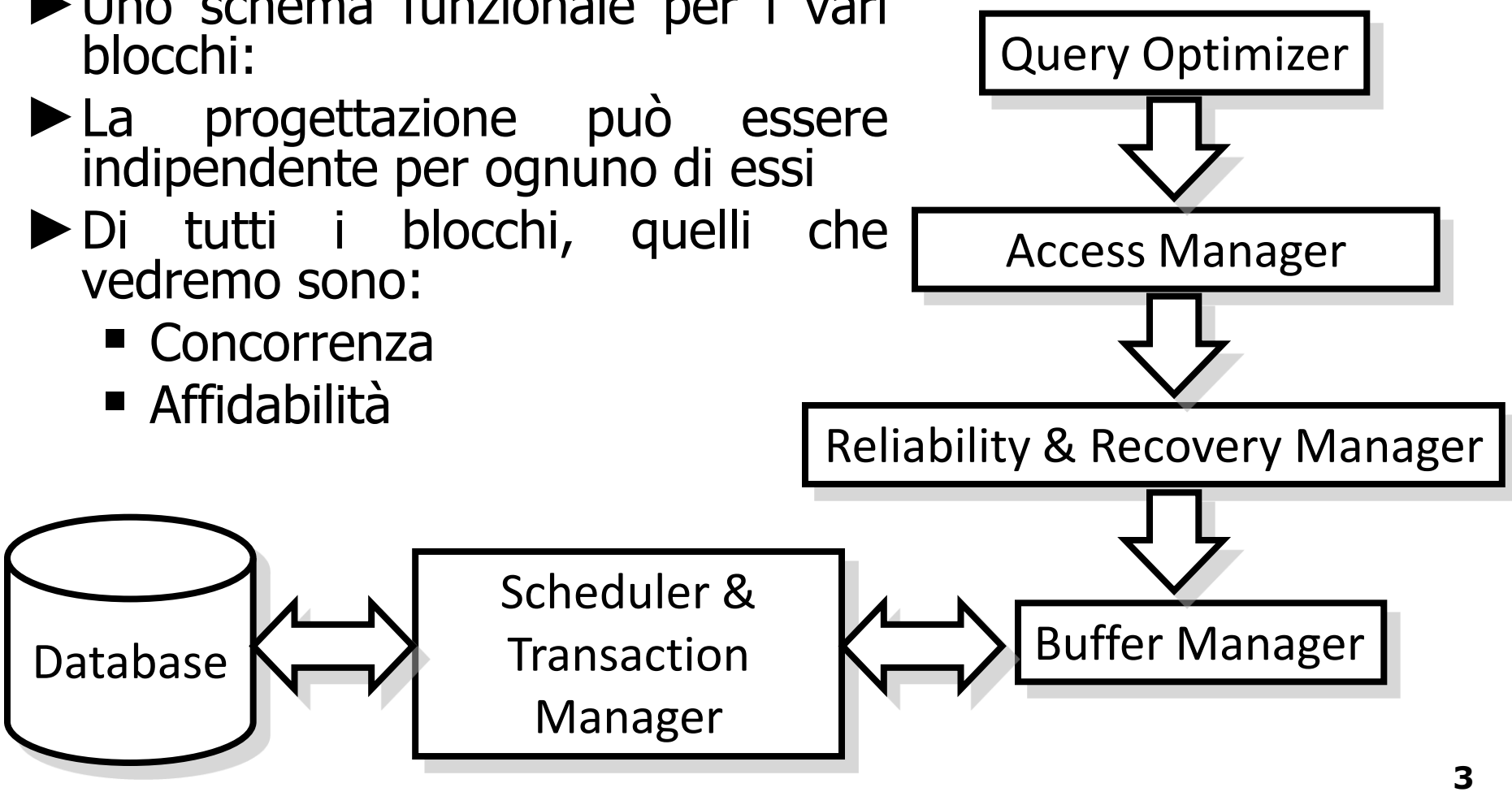
## *5 – Progettazione fisica*

# Progettazione Fisica

- Per effettuare la progettazione fisica, ossia l'implementazione reale del modello logico creato nella fase della progettazione logica, occorre conoscere la tecnologia di un DB server.
- In un DB server esistono **cinque componenti**:
  - **ottimizzatore**, che interpreta le query (comandi inviati al DB), ed elabora successivamente la strategia migliore per l'accesso ai dati
  - **gestore** di accesso fisico ai dati
  - **controllore di affidabilità**, per preservare il funzionamento del DB in caso di guasti e malfunzionamenti
  - **buffer manager**, che gestisce i trasferimenti dalla memoria di massa verso la memoria centrale e viceversa
  - **controllore della concorrenza**, per regolare gli accessi concorrenti e garantire la consistenza della base di dati

# Progettazione Fisica

- Uno schema funzionale per i vari blocchi:
- La progettazione può essere indipendente per ognuno di essi
- Di tutti i blocchi, quelli che vedremo sono:
  - Concorrenza
  - Affidabilità



# Controllo di concorrenza

- un DBMS deve spesso servire diverse applicazioni e/o diversi utenti simultaneamente, quindi occorre in generale un protocollo per la concorrenza, passando attraverso le **fasi** di:
  1. definizione di **operazione** nel mondo dei DBMS
  2. individuazione dei **problemi** di esecuzione concorrente
  3. stesura di una **teoria** che risolva i problemi, stabilendo come si possano evitare soddisfando opportune proprietà
  4. ricavare dalla teoria il **protocollo** per il controllo della concorrenza
- Punto 1: l'unità atomica di operazione è la **transazione**. Una transazione si può rappresentare con una sequenza di azioni iniziate da un ***begin transaction***, e concluse da un ***end transaction***, mentre all'interno delle due si deve trovare una sola delle istruzioni ***commit*** o ***rollback (abort)***, che indicano rispettivamente la corretta esecuzione e conclusione della transazione oppure l'annullamento della stessa

# Controllo di concorrenza

- Punto 2: dopo la definizione della transazione, si introducono alcuni **possibili problemi** che si potrebbero presentare durante l'esecuzione concorrente di più transazioni:
  - perdita di aggiornamento
  - lettura sporca
  - lettura inconsistente
  - aggiornamento fantasma

# Controllo di concorrenza

- **Perdita di aggiornamento**

| *TRANSAZIONE 1* | *TRANSAZIONE 2* |
|---|---|
| *bot* | |
| *r1(x)* | |
| *x=x+1* | |
| | *bot* |
| | *r2(x)* |
| | *x=x+1* |
| | *w2(x)* |
| | *commit* |
| *w1(x)* | |
| *commit* | |

- se il valore iniziale di x è 2, l'esecuzione sequenziale delle due transazioni darebbe 4, invece qui il valore finale è 3 perché entrambe le transazioni utilizzano il valore 2 per l'operazione di incremento;
- il problema si potrebbe risolvere con il lock sulla risorsa condivisa x, che permetterebbe si posporre la r2(x) a dopo il commit della prima transazione (momento del rilascio del lock su x)

# Controllo di concorrenza

- **Letture sporche**

| *TRANSAZIONE 1* | *TRANSAZIONE 2* |
|---|---|
| *bot* | |
| *r1(x)* | |
| *x=x+1* | |
| *w1(x)* | |
| | *bot* |
| | *r2(x)* |
| | *x=x+1* |
| | *w2(x)* |
| | *commit* |
| *abort* | |

- se il valore iniziale di x è 2, l'esecuzione sequenziale delle due transazioni darebbe 3 (una sola va a buon fine), invece qui il valore finale è 4 perché la seconda transazione utilizza il valore 3 successivo a w1(x) per il suo incremento;
- il problema stavolta nasce dal mancato isolamento (proprietà che quindi viene imposta)
- L'abort della prima provocherebbe un effetto domino, abortendo anche la seconda

# Controllo di concorrenza

- **Letture inconsistenti**

| *TRANSAZIONE 1* | *TRANSAZIONE 2* |
|---|---|
| *bot* | |
| *r1(x)* | |
| | *bot* |
| | *r2(x)* |
| | *x=x+1* |
| | *w2(x)* |
| | *commit* |
| *r1(x)* | |
| *commit* | |

- se il valore iniziale di x è 2, le due letture della prima transazione darebbero 3 e 4
- il problema nasce ancora dal mancato isolamento

# Controllo di concorrenza

- Il passo successivo (punto 3) è trovare una **teoria** del controllo della concorrenza che risolva i problemi visti prima, e ciò **seguendo un percorso in due fasi**:
  - imponendo le **proprietà** che una transazione deve soddisfare
  - stabilendo successivamente un **criterio** per permettere e gestire la concorrenza

# Controllo di concorrenza

- Ogni transazione deve godere di **quattro proprietà**:
  - atomicità
  - consistenza
  - isolamento
  - durabilità
- Le iniziali delle proprietà danno luogo al diffuso acronimo "**ACID**" da cui "proprietà acide delle transazioni"

# Controllo di concorrenza

- **Atomicità**
  - una transazione rappresenta una unità atomica ed indivisibile di azione
  - nel concreto, una transazione può essere implementata tramite più operazioni, ma si deve garantire che se qualcuna provoca un errore, tutte quelle svolte sino al verificarsi dell'errore devono essere disfatte (undo), riportando il DB allo stato antecedente l'inizio della transazione, come se questa non fosse mai iniziata; se invece tutte vanno a buon fine, il sistema deve garantire che il DB resti nello stato finale così determinato, richiesta che talvolta potrebbe implicare il dovere ripetere alcune operazioni della transazione (redo)

# Controllo di concorrenza

- **Consistenza**
  - questa proprietà implica che le operazioni della transazione non violino nessun vincolo di integrità sulla base di dati
  - la verifica può essere effettuata durante l'evoluzione della transazione (controllo immediato) o al massimo all'atto della richiesta di commit (controllo differito); se il controllo differito restituisce esito negativo, la richiesta di commit non può essere soddisfatta e alla transazione viene imposto un abort in extremis

# Controllo di concorrenza

- **Isolamento**
  - la proprietà impone che transazioni eseguite parallelamente devono produrre lo stesso risultato che si otterrebbe eseguendole in un ordine sequenziale qualunque, quindi le transazioni non devono potere l'una vedere eventuali modifiche parziali al DB operate dall'altra nel corso della sua esecuzione (appunto devono essere "isolate")
  - questo ha anche lo scopo di evitare che, in caso di fallimento di una transazione, anche quelle che sfruttano eventuali sue modifiche parziali al DB debbano essere abortite (evitare l'effetto domino)
- **Durabilità**
  - detta anche persistenza, questa proprietà impone che l'effetto di una transazione che ha eseguito un commit non venga perso

# Controllo di concorrenza

- Queste sin qua le proprietà. Assieme ad esse, occorre un criterio per gestire le transazioni concorrenti, raggiungendo l'obiettivo (non scontato) di garantire le proprietà.
- Di criteri ne sono stati definiti diversi, ed essi permettono di stabilire i possibili protocolli, ossia insiemi di regole che le transazioni concorrenti devono seguire per evitare problemi (punto 4).
- Omettendo i criteri, i protocolli più usati nella pratica sono:
  - Two phase locking protocol (2PL)
  - Timestamp protocol

# Controllo di concorrenza

**Locking a due fasi (2PL)**

- Il 2PL **previene i problemi** che possono provocare le transazioni concorrenti, utilizzando meccanismi di lock sulle risorse. **Il protocollo si basa sulle seguenti regole**:
  - ogni operazione deve preventivamente richiedere un lock sulla risorsa (tabella) a cui accedere e poi rilasciarlo; il lock è condivisibile con altri se l'operazione è di lettura (tanti possono leggere contemporaneamente), mentre il lock è esclusivo se di scrittura (solo uno per volta può scrivere sullo stesso oggetto)
  - Inoltre, una transazione non può acquisire nuovi lock se ne ha rilasciato uno, quindi una transazione deve inizialmente acquisire prima tutti i lock su tutti gli oggetti che utilizzerà, poi utilizza gli oggetti, infine rilascia tutti i lock, da qui il nome di **due fasi** (una crescente di acquisizione lock ed una decrescente relativa al loro rilascio alla fine), inframmezzate dall'effettiva esecuzione delle operazioni sulle risorse lockate
  - i lock possono essere rilasciati solo dopo aver correttamente effettuato le operazioni di commit/abort

# Controllo di concorrenza

**controllo basato sui timestamp (TS)**

- il timestamp è un identificatore temporale (ad esempio un codice che contiene hhmmss dell'istante in cui viene rilasciato) che ogni transazione deve ottenere.
- il controllo di concorrenza basato sui timestamp (TS) prevede che l'ordine di esecuzione delle transazioni deve riflettere l'ordinamento temporale dei timestamp associati alle transazioni stesse

# Controllo di concorrenza

- Confronto fra TS e 2PL, i due unici approcci ad essere realmente implementati:
  - Il TS è facile da implementare (basta controllare i tempi), il 2PL meno facile
  - con TS le transazioni non ammesse sono uccise (e verranno riavviate successivamente), con 2PL invece sono messe in attesa del rilascio dei lock
  - il restart della transazione in genere è molto costoso, però il 2PL potrebbe incorrere in deadlock, ossia le transazioni si aspettano l'una con l'altra e restano ferme per sempre. In tal caso si può aspettare un certo tempo, scaduto il quale (timeout) si uccidono comunque le transazioni bloccate, oppure si cerca di capire se sono bloccate veramente o solo in (lunga) attesa
  - Il 2PL o il TS in ogni caso sono implementati dal DBMS sottostante, non si devono creare da zero; è importante pero' sapere se esistono nel DBMS che si sta usando e se eventualmente si possono personalizzare

# Reliability & Recovery Manager

- Il controllore di affidabilità e recupero guasti serve a garantire persistenza e atomicità alle transazioni
- opera tramite un log, un archivio che si suppone persistente e realizzato su memoria stabile (astrazione), di fatto implementabile con mirroring (RAID1, RAID5, clustering…)
- sul log si scrivono tutte le azioni che si stanno per intraprendere, in modo da
  - poterle disfare in caso di abort o guasto prima della fine
  - poterle rifare in caso di commit e successivo guasto
- nel log sono anche presenti periodici dump (copie integrali del DB) e checkpoint (registrazione delle transazioni attive, più frequente del dump)

# Reliability & Recovery Manager

- i malfunzionamenti a cui fare fronte prevedono due recovery:
  - ripresa a caldo, attivata in caso di guasto di sistema, che modella la perdita delle operazioni in corso ma non del DB
  - ripresa a freddo, relativa ai guasti di dispositivo, ossia quelli che comportano anche la perdita del DB (in tutto o in parte)
  - in entrambi i casi non è prevista la perdita del log (catastrofe)
  - nella ripresa a caldo, si ripercorre il log all'indietro fino al checkpoint antecedente il guasto, e si annullano (rollback) tutte le transazioni che al momento del guasto non avevano eseguito il commit, e si rifanno (redo) tutte quelle che invece avevano eseguito un commit
  - nella ripresa a freddo, si opera come in quella a caldo ma prima di deve tornare all'ultimo dump per ricostruire il DB, riapplicando poi tutte le transazioni memorizzate da quel punto in poi