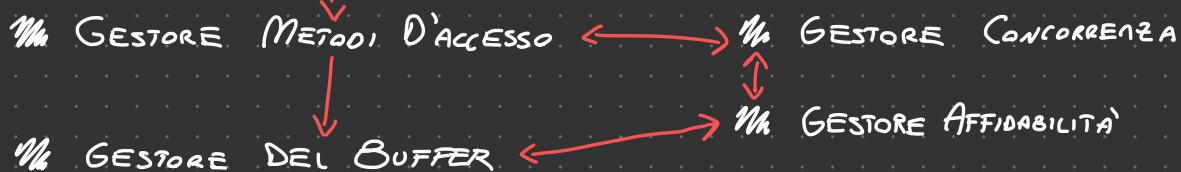


# Ciascuna delle Componenti ACID In dettaglio:

↓  
ACID  
Transazione ti (scritte in SQL) inviate al DBMS e processate da Vari Moduli, il PRIMO:

## ■ GESTORE INTERROGAZIONI

| piano Esecuzione



## ■ GESTORE MEM. SECONDARIA

Base di dati Risiede in Memoria Secondaria essendo che le Basi di Dati

■ Sono GRANDI  $\Rightarrow$  Non ci Stanno in MEM. CENTRALE

■ Sono PERSISTENTI  $\Rightarrow$  devono Rimanere anche al termine delle operazioni

Ci Sono Similarità tra funzionalità del FILESYSTEM e DBMS.

Mem. SECONDARIA è :

■ Non Utilizzabile direttamente dai Programmi e quindi dovremo caricare i dati in MEM. CENTRALE (ma ha dimensione <<)... serve un **Caricamento Intelligente**

■ DATI ORGANIZZATI IN BLOCCHI dette PAGINE  $\Rightarrow$  Non si Carica il Singolo BIT anche se devo leggere SOLO Quelle ma l'intera PAGINA.

Quindi posso Salamente fare **READ/WRITE** di un BLOCCO INTERO

**Il Costo Operazioni di READ/WRITE DA/VERSO MEM. SECONDARIA** è ordini di Grandezze Superiore al costo di Accesso in Memorie Centrale.

Mi Interessa Quindi Quanti ACCESSI A MEMORIA devo fare

### GESTORE DEL BUFFER:

Interazione Tra **MEM. CENTRALE** e quella **SECONDARIA** ed avviene Tramite il Trasferimento di Pagine/Blocchi **INTERI**.

Buffer è Condiviso Tra Tutte le APPLICAZIONI DELLE BASI DI DATI che Condividono la Stesso DBMS.

Deve Gestire in Maniera ottimizzata l'INSIEME LETTURE/SCRITTURE

Mantengo DATI in MEM

differenze SCRITTURE  
(GESTIRE GUASTI?)

Buffer è Organizzato anche Lui in PAGINE/BLOCCHI.

1 Pagine del Buffer è un Multiplo della dimensione del Blocco  
(NO STRA CONVENZIONE  $\Rightarrow$  1 PAG. DI BUFFER = 1 BLOCCO)

Buffer deve ottimizzare il più possibile per ridurre il COSTO.

## LETTOURA DI UN BLOCCO:

- ① VERIFICO SE BLOCCO È GIÀ PRESENTE NEL BUFFER e se così è restituisce PUNTATORE alle Pagine del BUFFER
- ② Altrimenti Cerca Pagine libere e carica il blocco nel BUFFER e poi restituisce PUNTATORE alle Pagine del BUFFER

## CERCARE PAGINA LIBERA:

- Zone libere di Memoria
- Qualche politica per Rimpiazzare una PAGINA

## SCRITTURA DI UN BLOCCO:

- ② di ~~scrivere~~ Scrittura... Modifiche Vengono Registrate SOLO in Mem. Centrale e Solo in un Secondo Momento vengono Registrate su MEM. SECONDARIA (ha un Effetto di prestazioni e affidabilità)

## PRINCIPIO DI LOCALITÀ:

I dati REFERENZIATI di Recent hanno una Maggior Probabilità di ESSERE REFERENZIATI Nuovamente e quindi Mantenuti; i dati appena caricati  $\Rightarrow$  NON SCARICATI SUBITO

Viene Usata anche una LEGGE EMPIRICA, che il 20% dei DATI è ACCEDUTO dall' 80% delle Applicazioni



Quantità dei dati ACCEDUTI << tutti i DATI POSSIBILI

Buffer è una Matrice dove ciascun Elemento è una PAGINA in memoria centrale.

Alcune Maccole come LIBERE mentre altre hanno info sul

B ② Blocco Caricato

i ② Contatore  $\Rightarrow$  #Transazioni che Stanno Attualmente Utilizzando la Pagina/Blocco

J ③ Booleano  $\Rightarrow \phi = \text{non modificata}$  e  $1 = \text{È stata modificata}$

|     |           |           |           |     |
|-----|-----------|-----------|-----------|-----|
| L   | $B_{2,0}$ | $B_{2,1}$ | $B_{1,0}$ | L   |
| ... | ...       | $B_{i,j}$ | L         | ... |
| ... | L         | ...       | ...       | ... |
| L   | ...       | L         | ...       | L   |

Gestore del BUFFER ha delle primitive:

Fix  $\Rightarrow$  Richiede l'accesso ad un Blocco, restituisce un puntatore alla Pagine su cui Tale Blocco

(1) RICERCA DEL Blocco ALL'INTERNO DEL BUFFER (vedo se c'è Già)

Se è GIÀ PRESENTE  $\Rightarrow$  restituisco un puntatore alla pagina che lo Contiene

Se non esiste viene Scelta una Pagine P libera Su cui effettuo il Caricamento.

Devo cercare però una PAGINA LIBERA:

Se c'è L  $\Rightarrow$  Perfetto Nessun Problema

**Se c'è  $B_{0,j} \Rightarrow i=0$**  per forza altri menti ci sono TRANS.  
che stanno facendo uso di quel blocco

### POLITICHE POSSIBILI:

- **LRU**  $\Rightarrow$  Least Recently Used } SOLO ORDINE TEMPORALE
- **FIFO**  $\Rightarrow$  First In First Out }

Se  $-J=0 \Rightarrow$  posso rimuoverla

$-J=1 \Rightarrow$  devo fare il **Salvataggio su Mem. SECONDARIA**  
con primitiva **FLUSH**

Se NON ci sono pagine libere ho 2 Politiche:

**STEAL**  $\Rightarrow$  Rubo pagina ad un'altra Transazione, da priorità alle Transazioni CORRENTE

**NO STEAL**  $\Rightarrow$  Transazione CORRENTE viene SOSPESA, messa in attesa dando priorità alle TRANSAZIONI già IN ESECUZIONE

### ② INCREMENTARE IL CONTATORE $i$

**SET DIRTY**  $\Rightarrow$  usata per indicare al Gestore del Buffer che il **BLOCCO caricato in una pagina è stato modificato**, quindi il BIT di Stato  $J=1$

**UNFIX**  $\Rightarrow$  Transazioni avvisano che hanno finito di utilizzare quel blocco quindi **CONTATORE  $i = i-1$**

2 Primitive ora che Venne Effettivamente a Scrivere Sulla  
MEMORIA SECONDARIA:

■ FLUSH  $\Rightarrow$  Salvare una Pagina di Memoria Centrale in Secondaria  
in Modo ASYNC (INVOCÀ QUEST'OPERAZIONE E LUI TERMINA SUBITO, LUI NON  
ASPETTA e METTE J=0)

■ FORCE  $\Rightarrow$  Per Salvare in Maniera Sincrona ( $\Leftrightarrow$  uscita del Gestore  
dell' AFFIDABILITÀ) e Sempre mette J=0

### GESTORE AFFIDABILITÀ:

Garantisce

■ Atomicità  $\Rightarrow$  No transazioni Eseguite Parzialmente

■ Persistenza  $\Rightarrow$  Modifiche devono Rimanere con  
SCRITTURE DIFFERITE  $\rightsquigarrow$  ripristinare CONTENUTI della  
mem.

Utilizza il LOG  $\Rightarrow$  archivio Persistente di Tutte le operazioni  
eseguite dal DBMS

MEMORIA STABILE  $\Rightarrow$  Mem. Resistente ai Guasti (assumiamo INFALLIBILE)  
dove memorizziamo il file di LOG

Nel file di log Abbiamo RECORD DI LOG DI:

■ TRANSAZIONE  $\Rightarrow$  Registrazione INFO Operazioni ESEGUITE da TRANSACTION

Come ad Esempio:

- BEGIN  $\Rightarrow$  B(t)

- COMMIT  $\Rightarrow$  C(t)

- ABORT  $\Rightarrow$  A(t)

- INSERT  $\Rightarrow$  I(t, 0, AS)

AFTER STATE

- DELETE  $\Rightarrow D(t, O, BS)$  BEFORE STATE
- UPDATE  $\Rightarrow U(t, O, BS, AS)$

I record permettono al Gestore Aff. di eseguire 2 operazioni di ripristino:

- UNDO  $\Rightarrow$  disfare un'azione di  $t$  su Oggetto  $O$   
RIPRISTINO AL BEFORE STATE
- REDO  $\Rightarrow$  Rifare un'Azione di  $t$  su Oggetto  $O$   
RIPRISTINO AL AFTER STATE

Devono Garantire le proprietà di IDEMPOTENZA

- $UNDO(UNDO(A)) = UNDO(A)$
- $REDO(REDO(A)) = REDO(A)$

### ■ SISTEMA $\Rightarrow$

- DUMP  $\Rightarrow$  È stata fatta un'Operazione di Copia Completa delle Base di dati ad Esempio per BACK-UP
- CHECK POINT  $\Rightarrow$  Ad Intervalli Regolari fatti in Automatico si Registra l'Elenco delle Transizioni Attualmente Attive, ovvero quelle che hanno ESEGUITO un BEGIN ma non ha Registrato ancora un COMMIT o UN ABORT

① Sospende le Operazioni in Corso

② Esegue Primitiva FORCE sulle Pagine con DIRTYBIT = 1 Per Tutte le TRANSAZIONI che hanno Eseguito il commit

③ Scrive Report di CHECK POINT  $\left[ CK(T_1 \dots T_n) \right]$  sul LOG elencando gli Identificatori delle Transazioni attive

④ Riprende le Operazioni Interrutte al punto ①

Quando lo Scrivo però il LOG? Ho 2 Regole:

① WAL  $\Rightarrow$  Write Ahead Log  $\Rightarrow$  I record di LOG vanno scritti nel file di LOG prima di Eseguire le operazioni sulla BASE DI DATI e questo mi GARANTISCE CHE POSSO SEMPRE FARE UN UNDO

② COMMIT PRECEDENZA  $\Rightarrow$  I Record di LOG vanno scritti nel file di LOG prima di Eseguire il Commit della Transazione e mi GARANTISCE LA POSSIBILITÀ DI FARE I REDO

Se il Guasto si Verifica prima della Scrittura del REPORT di commit  $\Rightarrow$  Esegua un UNDO  $\rightsquigarrow$  ATOMICITÀ

Se invece si Verifica dopo la Scrittura del REPORT di Commit si farà un REDO per rifare le OPERAZIONI DELLA TRANSAZIONE  $\rightsquigarrow$  PERSISTENZA

### TIPI DI GUASTO:

■ DI SISTEMA: perdita Contenuto della Memoria Centrale ad Esempio in Perdita di Corrente ①

■ DI DISPOSITIVO: Perdita del Contenuto della Memoria Secondaria ②

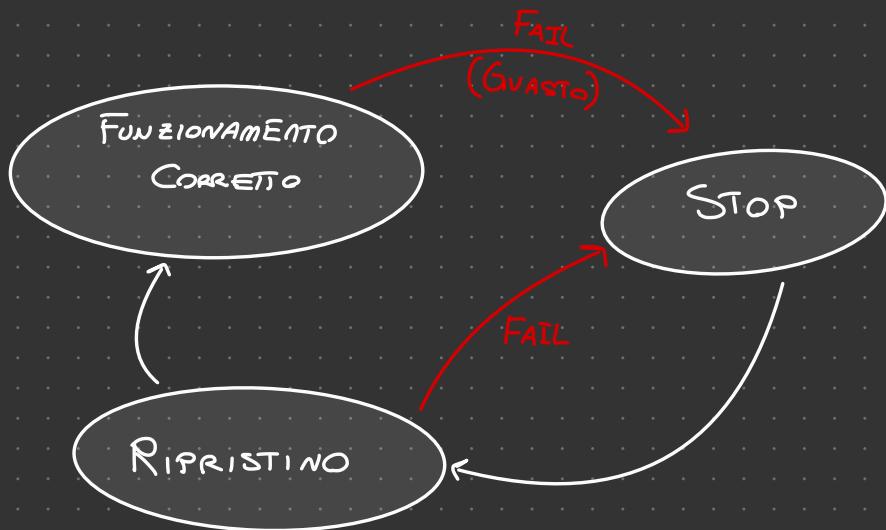
A seconda del Guasto che si Verifica ESEGUA un algoritmo di ripresa

u A CALDO  $\Rightarrow$  ①

u A FREDDO  $\Rightarrow$  ②

### FUNZIONAMENTO FAIL-STOP:

LOG



Memoria Stabile NON Viene Mai persa

## RIPRESA A CALDO:

- ① Si Accede All' Ultimo Blocco del LOG, ripercorrendo All'Indietro il LOG fino al più recente RECORD CK
- ② Si decidono le Trans. da RIFARE/DISFARE inizializzando:
  - INSIEME REDO come Insieme Vuoto
  - INSIEME UNDO con le Trans. Attive al CK
- ③ Si Ripercorre Avanti il LOG e per Ogni Record  $B(T)$  si aggiunge T a UNDO e per ogni Record  $C(T)$  si sposta T da UNDO a REDO
- ④ Si Ripercorre all' Indietro il LOG disfacendo le Operazioni Eseguite delle Transazioni in UNDO risalendo fino Alla Prima Azione della Transazione più Vecchia
- ⑤ Si Rifanno le Operazioni delle Transazioni dell'insieme REDO

## RIPRESA A FREDDO:

- ① Si Accede al DUMP della Base di Dati e si Ricopia Selettivamente la Parte deteriorata della BASE DI DATI
- ② Si Accede al LOG risalendo al record di DUMP
- ③ Si Ripercorre in Avanti il LOG rieseguendo tutte le operazioni relative alla parte Deteriorata Comprese le Azioni di Commit e Abort
- ④ Si Applica una Ripresa a Caldo