Basi di Dati
Modulo Tecnologie

Strutture fisiche e strutture di accesso ai dati (I parte)

Dr. Sara Migliorini

a.a. 2024-2025

DBMS e memoria secondaria

- Le basi di dati gestite da un DBMS risiedono in memoria secondaria, in quanto sono:
  - Grandi: non possono essere contenute in memoria centrale, e
  - Persistenti: hanno un tempo di vita che non è limitato all'esecuzione dei programmi che le utilizzano
- Caratteristiche della memoria secondaria:
  - Non è direttamente utilizzabile dai programmi
  - I dati sono organizzati in blocchi (o pagine)
  - Le uniche operazioni possibili sono la lettura e la scrittura di un intero blocco (pagina)
  - Il costo di tali operazioni è **ordini di grandezza maggiore** del costo per accedere ai dati in memoria centrale

Architettura DBMS

Gestore del Buffer (Cap. 11.1.2)

Il Buffer

- L'interazione tra **memoria secondaria** e **memoria centrale** avviene attraverso il trasferimento di pagine della memoria secondaria in una zona appositamente dedicata della memoria centrale detta BUFFER.

- Il buffer è una zona di memoria **condivisa dalle applicazioni**
- Quando uno stesso dato viene utilizzato più volte in tempi ravvicinati il buffer evita l'accesso alla memoria secondaria
- La gestione ottimale del buffer è strategica per ottenere buone prestazioni nell'accesso ai dati

Gestore del buffer

- Il buffer è organizzato in **PAGINE**.
- Una pagina ha le dimensioni di un **blocco della memoria secondaria**.
  - Ogni pagina ha la dimensione di un numero intero di blocchi.
  - Per semplicità assumiamo che 1 blocco in memoria secondaria = 1 pagina in memoria centrale
- Il gestore del buffer si occupa del caricamento/salvataggio delle pagine in memoria secondaria a fronte di richieste di lettura/scrittura.
  - Le tempistiche di caricamento/salvataggio possono non coincidere necessariamente con quella delle richieste ricevute.

Politica del gestore dei buffer

- Lettura di un blocco:
  - se il blocco è **presente in una pagina del buffer** allora non si esegue una lettura su memoria secondaria e si restituisce un puntatore alla pagina del buffer,
  - altrimenti si cerca una pagina libera e si carica il blocco nella pagina, restituendo il puntatore alla pagina stessa.
- Scrittura di un blocco:
  - In caso di richiesta di scrittura di un blocco precedentemente caricato in una pagina del buffer, il gestore del buffer può decidere di **differire la scrittura** su memoria secondaria in un secondo momento.

Politica del gestore dei buffer

- In entrambi i casi (lettura e scrittura) l'obiettivo e quello di aumentare la velocità di accesso ai dati.
- Tale comportamento del gestore dei buffer si basa sul **principio di LOCALITÀ**: "I dati referenziati di recente hanno maggiore probabilità di essere referenziati nuovamente in futuro".
- Inoltre, una nota legge empirica dice che: "il 20% dei dati e acceduto dall'80% delle applicazioni".
- Tutto ciò rende conveniente dilazionare la scrittura su memoria secondaria delle pagine del buffer.

Gestione delle pagine

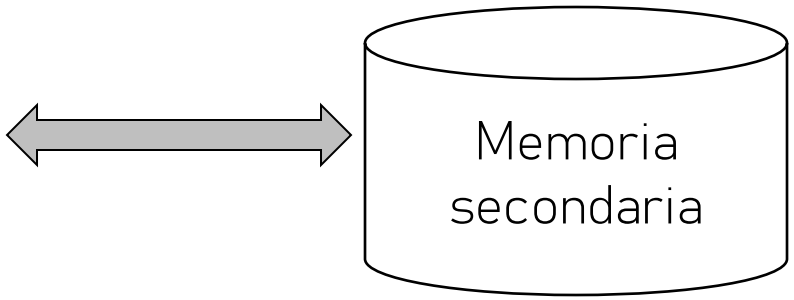
- Dati necessari per la gestione delle pagine:
  - Per ogni **pagina del buffer** si memorizza il **blocco B** contenuto indicando il file e il numero di blocco (o offset)
  - Per ogni pagina del buffer si memorizza un insieme di **variabili di stato** tra cui si trovano sicuramente:
    - Un **contatore I** (B_I) per indicare il numero di transazioni che utilizzano le pagine.
    - Un **bit di stato J** ($B_{I,J}$) per indicare se la pagina è stata modificata (=1) o no (=0).

$B_{2,0}$	L	L	$B_{1,1}$	L	L
$B_{1,1}$	$B_{1,1}$	$B_{1,0}$	$B_{1,0}$	L	L
L	L	$B_{0,0}$	L	$B_{0,0}$	$B_{0,0}$
L	L	L	L	$B_{0,1}$	L
L	L	L	$B_{0,0}$	$B_{0,0}$	L
L	L	L	L	L	L

Gestione delle pagine

BUFFER

$B_{2,0}$	L	L	$B_{1,1}$	L	L
$B_{1,1}$	$B_{1,1}$	$B_{1,0}$	$B_{1,0}$	L	L
L	L	$B_{0,0}$	L	$B_{0,0}$	$B_{0,0}$
L	L	L	L	$B_{0,1}$	L
L	L	L	$B_{0,0}$	$B_{0,0}$	L
L	L	L	L	L	L



$B_{i,j}$ indica che nella pagina del buffer è caricato il blocco B, inoltre "i" indica che il blocco è attualmente utilizzato da i transazioni mentre "j" è 1 se il blocco è stato modificato e 0 altrimenti. L indica pagina libera.

Primitive per la gestione del buffer

fix: viene usata dalle transazioni per richiedere l'accesso ad un blocco e restituisce al chiamante un puntatore alla pagina contenente il blocco richiesto.

- Passi della primitiva:
  - Il blocco richiesto viene cercato nelle pagine del buffer, in caso sia presente, si restituisce un puntatore alla pagina contenente il blocco
  - Altrimenti, viene scelta una pagina libera P (contatore I = zero). Si sceglie la pagina in base a criteri diversi: la pagina usata meno di recente (LRU) o caricata da più tempo (FIFO). Se il bit di stato di P è a 1, P viene salvata in memoria secondaria (primitiva flush). Si carica il blocco in P aggiornando file e numero di blocco corrispondenti.
  - Se non esistono pagine libere, il gestore del buffer può adottare due politiche (STEAL): "ruba" una pagina ad un'altra transazione (primitiva flush); oppure (NO STEAL) sospende la transazione inserendola in una coda di attesa. Se una pagina si libera (contatore I = zero) il gestore si comporterà come al punto precedente.
- Quando una transazione accede ad una pagina per la prima volta il contatore si incrementa.

Primitive per la gestione del buffer

setDirty: viene usata dalle transazioni per indicare al gestore del buffer che il blocco della pagina è stato modificato

- L'effetto è la modifica del bit di stato J a 1

unfix: viene usata dalle transazioni per indicare che la transazione ha terminato di usare il blocco

- L'effetto è il decremento del contatore I che indica l'uso della pagina.

Primitive per la gestione del buffer

force: viene usata per salvare in memoria secondaria in modo SINCRONO il blocco contenuto nella pagina (primitiva usata dal modulo Gestore dell'Affidabilità).

- L'effetto è il salvataggio in memoria secondaria del blocco e il bit di stato J posto a zero.

flush: viene usata dal gestore del buffer per salvare blocchi sulla memoria secondaria in modo ASINCRONO. Tale operazione libera pagine "dirty" (il bit di stato viene posto a zero)

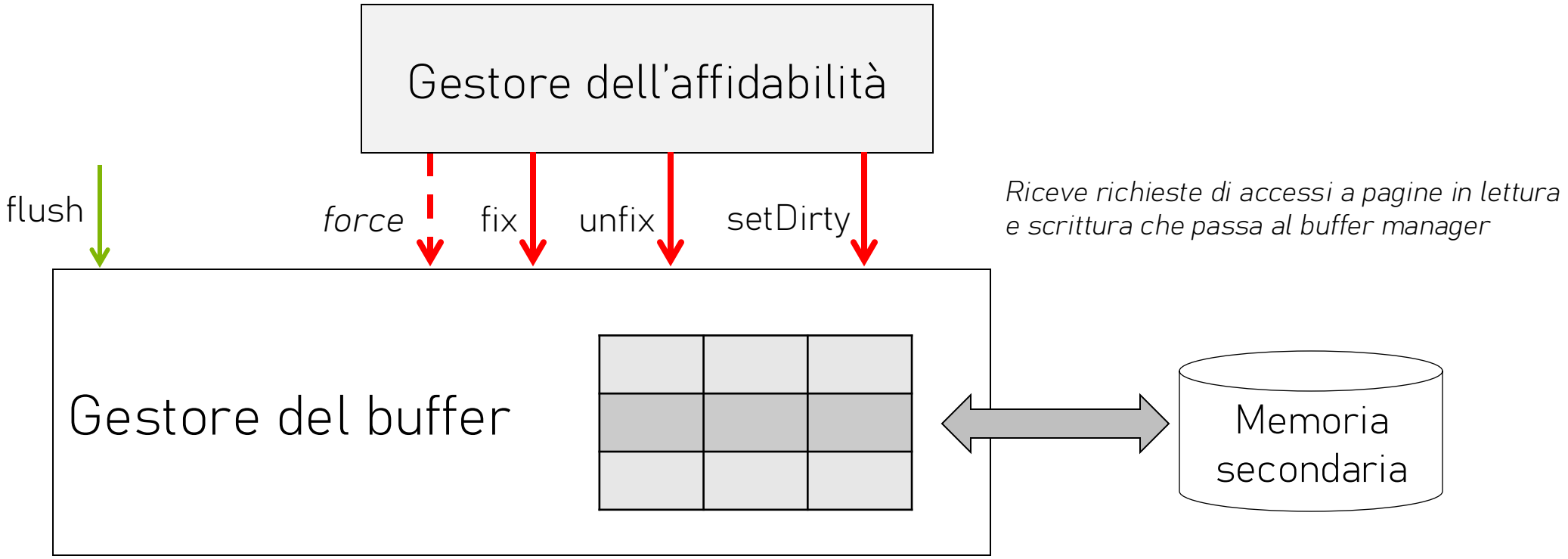
Architettura DBMS

Gestore dell'affidabilità (Cap. 12.1)

Gestore dell'affidabilità

- È il modulo garantisce due proprietà fondamentali delle transazioni:
  - Atomicità: garantisce che le transazioni non vengano lasciate incomplete con alcune operazioni eseguite ed altre no
  - Persistenza: garantisce che gli effetti di ciascuna transazione con conclusa con un commit siano mantenuti in modo permanente
- Il suo funzionamento si basa sull'utilizzo del LOG: un archivio persistente su cui vengono registrate tutte le operazioni eseguite dal DBMS
- Per il suo funzionamento il gestore dell'affidabilità deve disporre di un dispositivo di MEMORIA STABILE.
  - MEMORIA STABILE = memoria resistente ai guasti

Gestore dell'affidabilità



Gestione del file di LOG

- In **memoria stabile** viene memorizzato il **file di LOG** che registra in modo sequenziale le operazioni eseguite dalle transazioni sulla base di dati.
- RECORD memorizzati nel file di LOG:
  - Record di TRANSAZIONE
  - Record di SISTEMA

File di LOG- Record di Transazione

Record di TRANSAZIONE:

- Begin della transazione T: record `B(T)`
- Commit della transazione T: record `C(T)`
- Abort della transazione T: record `A(T)`
- Insert, Delete e Update eseguiti dalla transazione T sull'oggetto O:
  - record `I(T,O,AS)`: AS = After State
  - record `D(T,O,BS)`: BS = Before State
  - record `U(T,O,BS,AS)`

File di LOG- Record di Transazione

- I record di transazione salvati nel LOG consentono di eseguire in caso di ripristino le seguenti operazioni:
  - UNDO: per disfare un'azione su un oggetto O è sufficiente ricopiare in O il valore BS; l'insert/delete viene disfatto cancellando/inserendo O;
  - REDO: per rifare un'azione su un oggetto O è sufficiente ricopiare in O il valore AS; l'insert/delete viene rifatto inserendo/cancellando O;
- Gli inserimenti controllano sempre l'esistenza di O (non si inseriscono duplicati)

- Proprietà di IDEMPOTENZA:
  - UNDO(UNDO(A))=UNDO(A)
  - REDO(REDO(A))=REDO(A)

File di LOG- Record di Sistema

Record di SISTEMA:

- Operazione di **DUMP** della base di dati: record di `DUMP`
  - Copia completa e consistente della base di dati.
- Operazione di **ChechPoint**: record `CK(T1, …, Tn)` indica che all'esecuzione del CheckPoint le transazioni attive erano T1, …, Tn.
  - Sono elencate le transazioni che non hanno ancora espresso la loro scelta relativamente all'esito finale.
  - Operazione svolta periodicamente dal gestore dell'affidabilità per garantire che gli effetti delle transazioni che hanno eseguito il commit siano registrati in modo permanente.
  - Prevede i seguenti passi:
    1. Sospensione delle operazioni di scrittura, commit e abort delle transazioni
    2. Esecuzione della primitiva force sulle pagine "dirty" di transazioni che hanno eseguito il commit
    3. Scrittura sincrona (primitiva force) sul file di LOG del record di CheckPoint con gli identificatori delle transazioni attive
    4. Ripresa delle operazioni di scrittura, commit e abort delle transazioni

Regole di scrittura sul LOG

- Regole per la **scrittura sul LOG** ed esecuzione delle transazioni:
  - Regola WAL (**Write Ahead Log**): i record di log devono essere scritti sul LOG prima dell'esecuzione delle corrispondenti operazioni sulla base di dati (garantisce la possibilità di fare sempre UNDO)
  - Regola Commit-Precedenza: i record di log devono essere scritti sul LOG prima dell'esecuzione del COMMIT della transazione (garantisce la possibilità di fare sempre REDO)
- Tali regole consentono inoltre di salvare i blocchi delle pagine "dirty" in modo totalmente asincrono rispetto al commit delle transazioni.

Azione di COMMIT

- Una transazione T sceglie in modo atomico l'esito di COMMIT nel momento in cui scrive nel file di LOG in modo sincrono (primitiva force) il suo record di COMMIT - `C(T)`.

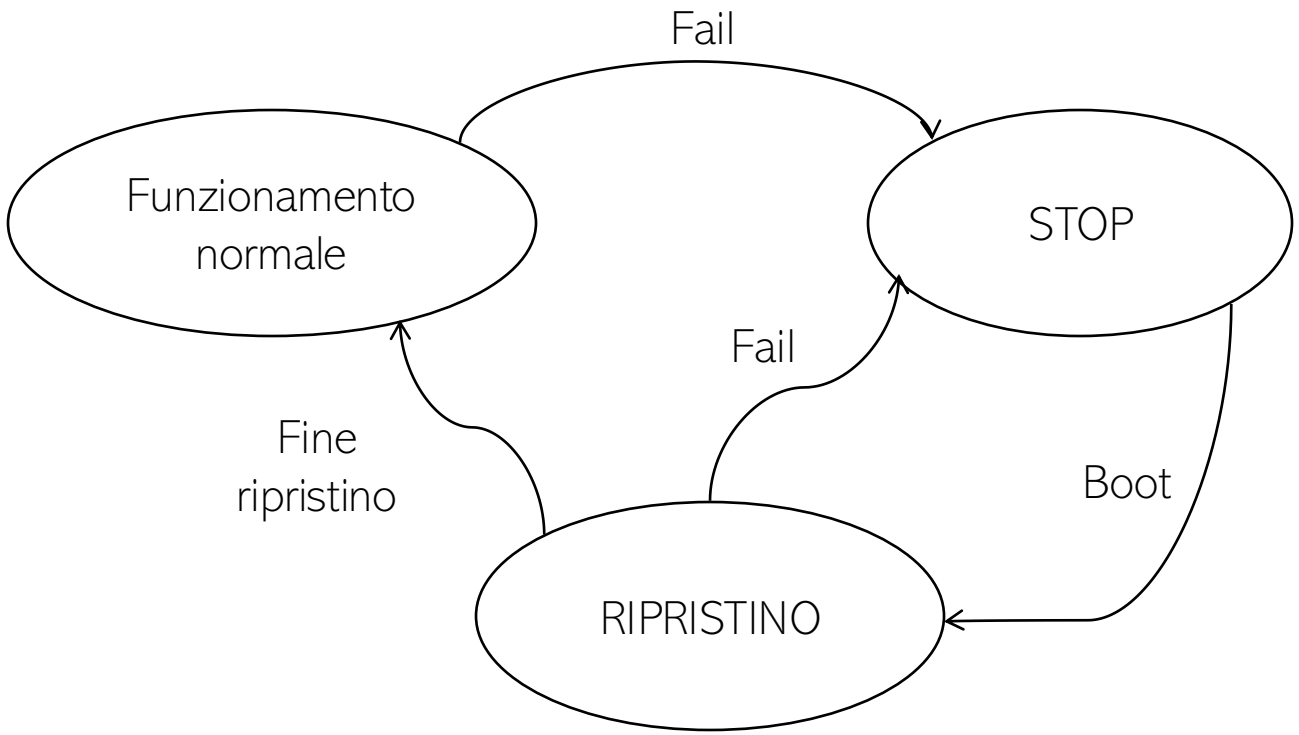
  - Guasto prima della scrittura del RECORD di COMMIT → UNDO
  - Guasto dopo la scrittura del RECORD di COMMIT → REDO

Gestore dell'affidabilità

- Due tipologie di di guasto:
  - Guasto di sistema: perdita del contenuto della memoria centrale
    → Ripresa a caldo
  - Guasto di dispositivo: perdita di parte o tutto il contenuto della base di dati in memoria secondaria
    → Ripresa a freddo

Gestore dell'affidabilità

Modello di funzionamento: fail-stop



Ripresa a caldo

Passi:

1. Si accede all'ultimo blocco del LOG e si ripercorre all'indietro il log fino al più recente record CK.
2. Si decidono le transazioni da rifare/disfare inizializzando l'insieme UNDO con le transazioni attive al CK e l'insieme REDO con l'insieme vuoto.
3. Si ripercorre in avanti il LOG e per ogni record B(T) incontrato si aggiunge T a UNDO e per ogni record C(T) incontrato si sposta T da UNDO a REDO.
4. Si ripercorre all'indietro il LOG disfacendo le operazioni eseguite dalle 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

Passi:

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

Esercizio

- Data la seguente situazione del file di LOG:
- B(T1), B(T2), U(T2,O1,B1,A1), I(T1,O2,A2), B(T3), C(T1), B(T4), U(T3,O2,B3,A3), U(T4,O3,B4,A4), CK(T2,T3,T4), C(T4), B(T5), U(T3,O3,B5,A5), U(T5,O4,B6,A6), D(T3,O5,B7), A(T3), C(T5), I(T2,O6,A8) guasto

- Che passi svolge la ripresa a caldo?