Università degli Studi di Verona

Dipartimento di Informatica
A.A. 2024 - 2025

Tecnologie per le basi di dati

Corso tenuto da
S. MIGLIORINI

Appunti a cura di
NICK

Made with LaTeX

Indice

1 Transizioni

1.1 DBMS - Data Base Management System

Un DBMS è un sistema software di gestione di basi di dati. Il sistema deve gestire collezioni di dati:

- **Grandi**: non possono essere caricati in memoria centrale (RAM).
- **Condivise**: ai dati possono accedere diversi programmi **contemporaneamente**.
- **Persistenti**: devono rimanere anche quando terminano i programmi e, in caso di guasti, dovranno esserci dei sistemi per ripristinare la situazione antecedente al guasto.

Un DBMS deve anche assicurare l'**affidabilità**, sempre inerente alla gestione dei guasti, e la **privatezza**, ovvero assicurare che accessi, scritture e letture avvengano secondo schemi definiti.

Un DBMS, inoltre, deve dare accesso ai dati in modo *efficace* ed *efficiente*. Queste caratteristiche dipendono dal modo in cui vengono strutturate le basi di dati.

Un DBMS condivide con il sistema operativo alcune funzionalità, estendendo quelle del file system. Infatti, il DataBase Management System si basa sulla struttura dle file system per memorizzare i dati grezzi, aggiungendo delle strutture interne per facilitare la ricerca dei dati.

1.1.1 Architettura a tre livelli

Un DBMS è rappresentato con un'architettura a tre livelli:

R-DBMS Si tratta di DBMS basati sul modello relazionale, e vengono definiti **sistemi transazionali**.

1.2 Transazione

Transazione è una unità indivisibile di lavoro. Una transazione va a buon fine, oppure non ha alcun effetto. Non sono ammesse esecuzioni parziali di una transazione.

Una transazione è una sorta di programma che segue il seguente schema:

```
1 BEGIN TRANSACTION
2    <PROGRAMMA>
3 END TRANSACTION
```

Il programma è della forma [ISTRUZIONE] | [COMMIT WORK] | [ROLLBACK WORK] .

***Esempio**: passaggio di 100.00€ tra due conti correnti*

```
1  BEGIN TRANSACTION
2    // istruzione 1
3    UPDATE conto SET saldo = saldo - 100
4      WHERE conto = '15';
5    // istruzione 2
6    UPDATE conto SET saldo = saldo + 100
7      WHERE conto = '18';
8
9    // commit
10   COMMIT WORK;
11 END TRANSACTION
```

Il fatto di aver inserito all'interno di un'unica transazione le due istruzioni farà si che, in caso di problemi, le istruzioni vengano eseguite entrambe oppure nessuna delle due.

1.3 Proprietà ACID

Il DBMS garantisce quattro priprietà dette *proprietà acide* dall'acronimo inglese ACID. Queste proprietà sono **A**tomicity, **C**onsistency, **I**solation e **D**urability[1].

1.3.1 Atomicity (atomicità)

Una transazione è una unità di lavoro indivisibile, ovvero o vengono eseguite tutte le istruzioni che compongono la transazione, oppure non ne viene eseguita nessuna. Se durante la transazione dell'esempio precedente si interrompe dopo aver tolto i 100 euro dal primo conto, il DBMS deve riportare quel conto al valore che aveva precedentemente.

1.3.2 Consistency (consistenza)

L'esecuzione di una transazione deve rispettare i vincoli di integrità. Il DBMS andarà a verificare in due modi che i vincoli vengano rispettati:

- **Verifica immediata**: si ha quando il DBMS verifica la consistenza dei vincoli durante l'esecuzione della transazione. Questo da la possibilità di reagire.
- **Verifica differita**: in questo caso il controllo viene fatto al momento del commit. Questo non da la possibilità di reagire alla transazione perché il sistema annulla la transazione.

1.3.3 Isolation (isolamento)

L'esecuzione di ogni transazione è indipendente dall'esecuzione contemporanea di altre transazioni. Servono dei sistemi di gestione della concorrenza e, nel caso in cui una transazione subisca un rollback, non deve causare un rollback a catena di altre transazioni.

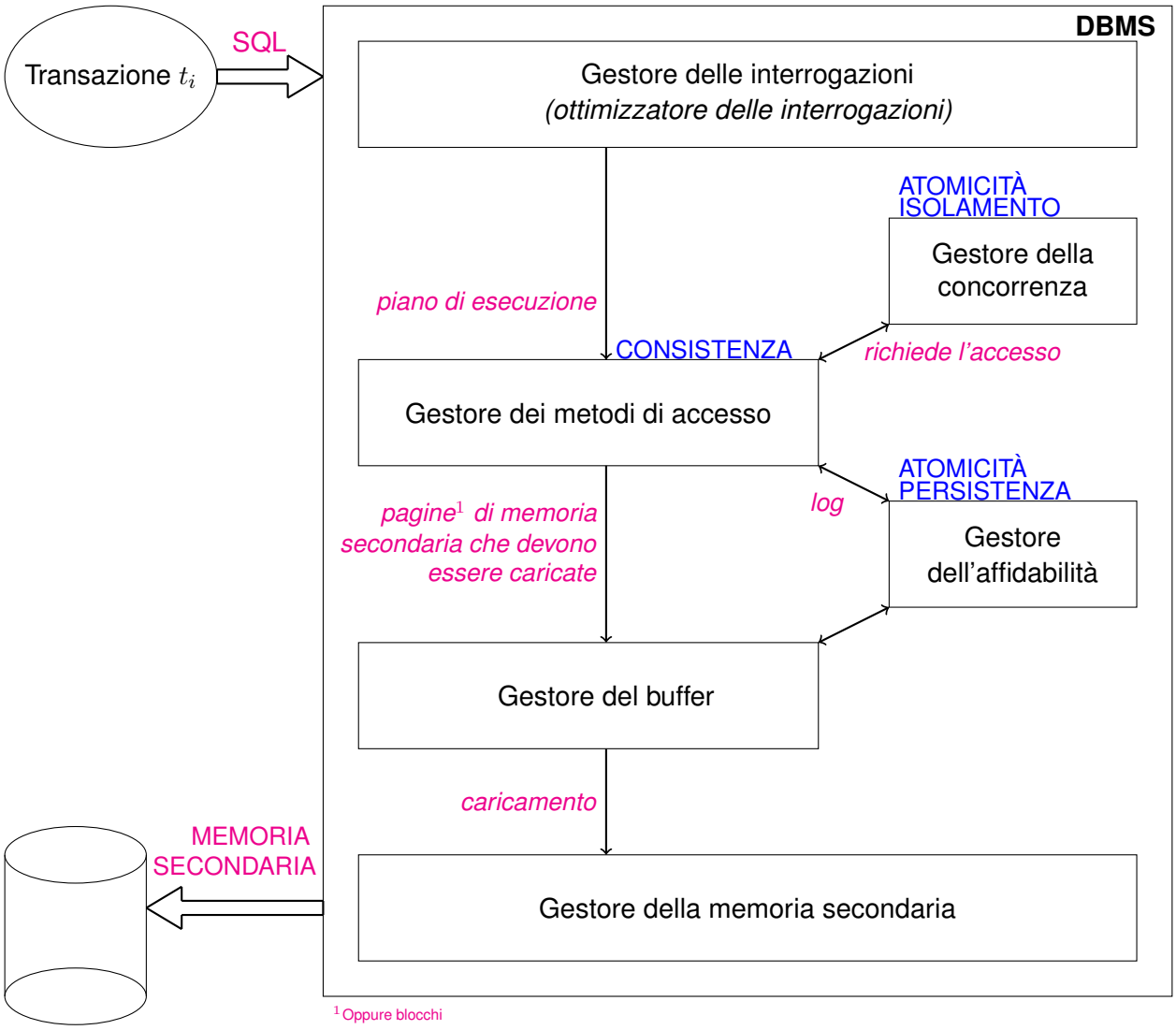
1.3.4 Durability (persistenza)

Gli effetti devono permanere anche dopo il completamento della transazione.

[1] Si traduce in persistenza

1.4 DBMS

Il DBMS quindi è composto da dei moduli software che permettono di garantire le proprietà ACID delle transizioni.



1.5 Gestore della memoria secondaria

Sul gestore della memoria secondaria ricordiamo solo due concetti chiave:

1. Il contenuto della base di dati risiede in memoria secondaria, questo per soddisfare il fatto che le basi di dati sono **grandi** e **persistenti**.
2. Caratteristiche della memoria secondaria:
   - Non è direttamente usabile dai programmi. I dati vanno caricati in memoria centrale, che ha delle dimensioni molto inferiori.
   - I dati sono organizzati in blocchi (pagine). Le uniche operazioni sono **lettura** e **scrittura** di un *intero blocco*.
   - Il costo delle operazioni lettura/scrittura da/verso la memoria secondaria è di ordini di grandezza superiore al costo di accesso alla memoria centrale.

1.6 Gestore del buffer

Il gestore del buffer si occupa dell'interazione tra la memoria centrale e la memoria secondaria, ovvero del trasferimento di pagine (o blocchi).

Il buffer è condiviso tra le applicazioni ed è organizzato in pagine. Generalmente una pagina del buffer è un multiplo di un blocco, noi consideriamo 1 pagina = 1 blocco.

Il buffer ottimizza le letture e le scritture. Ha il compito di mantenere i dati in memoria e di differire le scritture.

1.6.1 Lettura di un blocco

Per la lettura di un blocco si deve verificare se il blocco è già presente in memoria centrale; in questo caso il gestore del buffer restituisce un **puntatore** alla pagina del buffer. Altrimenti, il gestore *cerca* una pagina libera e carica il blocco nel buffer, restituendo poi il puntatore alla pagina del buffer.

1.6.2 Scrittura di un blocco

La scrittura di un blocco può essere differita dal gestore del buffer, in questo caso le modifiche vengono registrate solo in memoria centrale. L'obiettivo è quello di **ridurre il numero di accessi** alla memoria secondaria.

1.6.3 Principio di località

Il principio di località dice che i dati referenziati di recente hanno una maggiore probabilità di essere referenziati nuovamente, quindi i dati appena caricati in memoria vengono mantenuti.

Legge empirica: il 20% dei dati è acceduto dall'80% delle applicazioni. Infatti, la quantità di dati acceduti è molto minore ai dati possibili.

1.6.4 Buffer

Il **buffer** è organizzato come una matrice, in cui ogni cella rappresenta una pagina della memoria centrale.

$B_{i,j}$	L		
L			$B_{2,0}$
		L	

Dove L indica una pagina libera, B indica il blocco di memoria e i, j sono le *variabili di stato*.

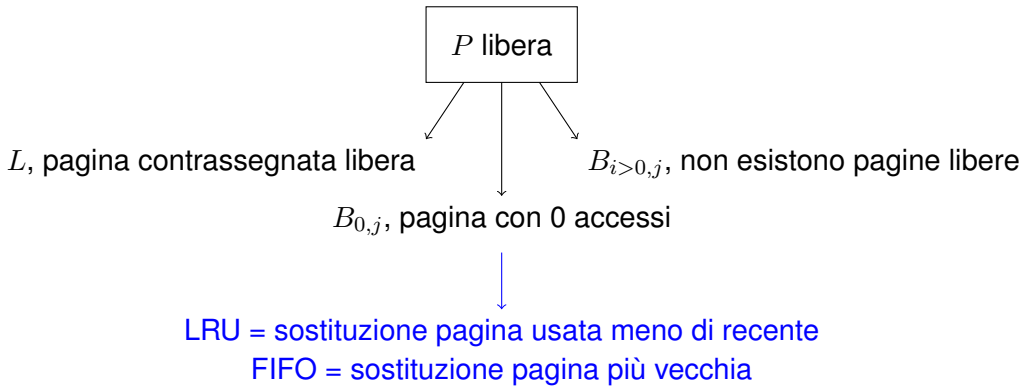
- i = *contatore del numero di transizioni che stanno utilizzando la pagina* B.
- $j \in \{0, 1\}$ = bit che indica se la pagina è stata modificata (1) oppure no (0).
- Esempio: $B_{2,0}$ indica che ci sono due transizioni sulla pagina e che al momento non è stata scritta.

1.6.5 Primitive

FIX

FIX: viene usata dalle transazioni per richiedere l'accesso ad un blocco . Restituisce un puntatore alla pagina su cui tale blocco è caricato.

1. Il blocco viene ricercato tra le pagine del buffer.
   - $\rightarrow$ Se è già presente viene restituito un puntatore alla pagina che lo contiene.
   - $\rightarrow$ Se non esiste viene scelta una pagina P libera su cui effettuare il caricamento.



- $\rightarrow$ Se non ci sono pagine libere si applica uno dei due paradigmi: **STEAL**, che prevede di *rubare* una pagina ad un'altra transazione, oppure **NOSTEAL**, ovvero la transazione corrente viene sospesa.

2. Incrementa il contatore i.

SET DIRTY

SET DIRTY: viene usata dalle transazioni per indicare al gestore del buffer che il blocco della pagina è stato modificato (bit di stato $j = 1$).

UNFIX

UNFIX: viene usata dalle trasnsazioni per indicare che la transazione ha terminato l'uso di un blocco (bit di stato $i = i - 1$).

FLUSH

FLUSH: viene usata dal **gestore del buffer** per salvare una pagina di memoria centrale in memoria secondaria in modo **asincrono**. (Bit di stato $j = 0$).

FORCE

FORCE: viene usata dal **gestore dell'affidabilità** per effettuare scritture in modo **sincrono**. (Bit di stato $j = 0$).

1.7 Gestore dell'affidabilità

Il gestore dell'affidabilità è il modulo che garantisce l'atomicità delle transazioni e la persistenza dei dati. 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**: una memoria resistente ai guasti.

1.7.1 Record di LOG

Nella memoria stabile viene memorizzato il file di LOG che registra in modo sequenziale le operazioni eseguite dalle transazioni sulla base di dati. In questo file vengono memorizzati i **record di transazione** e il **record di sistema**.

Record di transazione

I record di transazione sono:

- Inizio (begin) di una transazione t: `record B(t);`
- Commit di una transazione t: `record C(t);`
- Abort di una 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).`

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:

$$\texttt{UNDO(UNDO(A))} = \texttt{UNDO(A)}$$
$$\texttt{REDO(REDO(A))} = \texttt{REDO(A)}$$

Record di sistema

I record di sistema consistono in due operazioni: **DUMP** e **CHECKPOINT**. Il DUMP consiste nell'eseguire una copia **completa e consistente** della base di dati.

L'operazione di CHECKPOINT ($\texttt{record CK(T}_1\texttt{, ..., T}_n\texttt{)}$) indica che all'esecuzione del checkpoint le transazioni attive erano $\texttt{T}_1\texttt{, ...,T}_n$. Sono elencate le transazioni che non hanno ancora espresso la loro scelta relativamente all'esito finale. L'operazione viene svolta periodicamente dal **gestore dell'affidabilità** per garantire che gli effetti delle transazioni che hanno eseguito il commit siano registrati in modo permanente.

Passi del CHECKPOINT :

1. Sospende le operazioni in corso.
2. Esegue la primitiva `FORCE` sulle pagine *dirty* per tutte le transazioni che hanno eseguito il commit.
3. Scrive il record ($\texttt{CK(T}_1\texttt{, ..., T}_n\texttt{)}$) di checkpoint sul LOG elencando gli identificatori delle transazioni attive.
4. Riprende le operazioni.

1.7.2 Regole di scrittura sul LOG

Esistono due regole per la scrittura sul LOG e l'esecuzione delle transazioni: **Write Ahead Log** e **commit-precedenza**. Tali regole consentono di salvare i blocchi delle pagine *dirty* in modo totalmente asincrono rispetto al commit delle transazioni.

WAL - Write Ahead Log

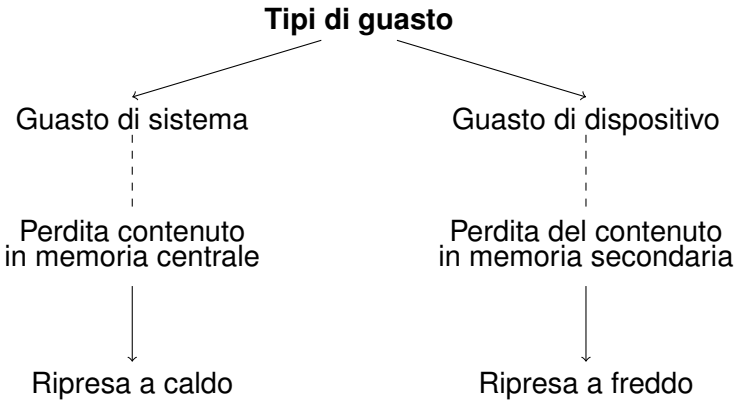
I record di LOG devono essere scritti prima dell'esecuzione delle corrispondenti operazioni sulla base di dati, questo garantisce la possibilità di fare sempre UNDO.

Commit-precedenza

I record di log devono essere scritti sul LOG prima dell'esecuzione del COMMIT della transazione, questo garantisce la possibilità di fare sempre REDO.

Guasto primadella scrittura record commit ⇒ UNDO Atomicità. **Guasto dopo** della scrittura record commit ⇒ REDO Persistenza.

1.7.3 Tipi di guasto



FAIL-STOP