BASI DI DATI 2

TECNICHE DI RECOVERY

a.a 2024/2025

Prof.ssa G. Tortora

Perché è necessario il recovery?

- Se viene sottomessa una transazione T:
  1. o tutte le operazioni di T sono completate ed il loro effetto è registrato permanentemente nel DB,
  2. o T non ha nessun effetto né sul DB né su altre transazioni (*failure*).

- Nel caso **2.** il sistema non può permettere che alcune operazioni di T siano portate a termine ed altre no.

Concetti sul Recovery

- Effettuare un recovery (o recupero) da una transazione fallita significa ripristinare il database al più recente stato consistente appena prima del failure (o guasto).

- Per fare ciò, il sistema deve tener traccia dei cambiamenti causati dall'esecuzione di transazioni sui dati.

- Tali informazioni sono memorizzate nel system log.

Le entry del System Log

- Il log è strutturato come una lista di record.
- In ogni record è memorizzato un **ID** univoco della transazione **T**, generato in automatico dal sistema.
- Tipi di entry possibili nel log:
  - **[start_transaction, T]**
  - **[write_item, T, X, old_value, new_value]**
  - **[read_item, T, X]**
  - **[commit, T]**
  - **[abort, T]**

Strategie di recovery

- Strategie di recovery tipiche:
  1. Se il danno è notevole, a causa di un failure catastrofico (es. crash di un disco), si ripristina una precedente copia di back-up da una memoria di archivio e si ricostruisce lo stato più vicino a quello corrente riapplicando (redo) tutte le transazioni completate (*committed*) salvate nel log fino al momento del failure.

Strategie di recovery (2)

2. Se, a seguito di un failure non catastrofico, il database non è danneggiato fisicamente, ma è solo in uno stato inconsistente, si effettua l'UNDO delle operazioni che hanno causato l'inconsistenza.

- Eventualmente si effettua il REDO di alcune operazioni.
  - Non è necessario effettuare il restore del database, poiché basta la consultazione del system log.

Tecniche di recovery da failure non catastrofici

- Concettualmente possiamo distinguere due tecniche principali per il recovery da failure non catastrofici:
  - Tecniche ad aggiornamento differito – deferred update

  (algoritmo NO-UNDO/REDO)

  - Tecniche ad aggiornamento immediato – immediate update

  (algoritmo UNDO/REDO)

Tecniche ad aggiornamento differito

- Con questa tecnica, i dati **non** sono fisicamente aggiornati fino all'esecuzione della commit di una transazione.
- Le modifiche effettuate da una transazione sono memorizzate in un suo spazio di lavoro locale (workspace o buffer).
- Durante il commit, gli aggiornamenti sono salvati persistentemente prima nel log e poi nel DB.
- Se la transazione fallisce, non è necessario l'UNDO; può però essere necessario il REDO di alcune operazioni (*commit con scrittura nel log ma non nel DB*).

Tecniche ad aggiornamento immediato

- Il DB può essere **aggiornato fisicamente** prima che la transazione effettui il commit.
- Le modifiche sono registrate prima nel log (con un force-writing) e poi sul DB, permettendo comunque il recovery.
- Se una transazione fallisce dopo aver effettuato dei cambiamenti, ma prima del commit, l'effetto delle sue operazioni nel DB deve essere annullato:
  - occorre effettuare il rollback della transazione (UNDO); può però essere necessario il REDO di alcune operazioni (*commit con scrittura nel log ma non nel DB*).

Caching dei blocchi

- Le tecniche di recovery possono essere influenzate da particolari gestioni del file system da parte del sistema operativo, in particolare dal buffering e dal caching di blocchi del disco in memoria centrale.

- Per migliorare l'efficienza degli accessi a disco, i blocchi del disco contenenti i dati manipolati spesso dal DBMS, sono conservati (cached) in un buffer della memoria centrale:
  - i dati sono quindi aggiornati in memoria, prima di essere riscritti su disco.

Caching dei blocchi (2)

- Sebbene la gestione del caching sia un compito del sistema operativo, spesso è il DBMS ad occuparsene esplicitamente, a causa dello stretto accoppiamento con le tecniche di recovery.
  - Il DBMS gestisce direttamente una serie di buffer, che formano la cache del DBMS.

- Per tenere traccia dei data item presenti in cache, si usa una directory, ovvero una tabella con entry del tipo:

  <indirizzo pagina su disco, locazione nel buffer>

Accesso ai dati con cache

- Quando il DBMS richiede l'accesso ad un data item, se ne controlla la presenza in cache:
  - Se già è presente, il DBMS ne ottiene l'accesso.
  - Se non è presente:
    1. Deve essere individuato il blocco su disco contenente l'item.
    2. Il blocco deve essere copiato nella cache.
    3. Se la cache è piena, sono necessarie strategie tipiche dei sistemi operativi per il rimpiazzamento delle pagine (*LRU – least recently used*, *FIFO – first in first out*, ecc …).

Il Dirty Bit

- Ad ogni blocco nella cache si può associare un dirty bit, per evidenziare se qualche elemento del buffer è stato modificato o meno.
- Al caricamento di un blocco in un buffer, il suo dirty bit è posto a **0**.
- In seguito ad una modifica del contenuto del buffer, il suo dirty bit è posto a **1**.
- Un buffer deve essere salvato su disco solo se il suo dirty bit vale **1**.

Strategie per lo svuotamento della cache

- Esistono due strategie principali per lo svuotamento di buffer modificati:
  1. In-place updating:
     Il buffer è riscritto nella stessa posizione originaria sul disco. Si mantiene in cache una singola copia di ogni blocco del disco. È necessario usare il system log per il recovery.
  2. Shadowing:
     Un buffer può essere riscritto in una locazione differente, permettendo la presenza su disco di più versioni di un data item:
     - Sia il vecchio valore (BFIM – before image), sia quello aggiornato (AFIM – after image) di un data item possono essere presenti sul disco contemporaneamente.

Il Write-Ahead Logging

- Usando l'in-place updating è necessario il system log per il recovery.

- Il log contiene due tipi di informazioni: quelle per l'UNDO e quelle per il REDO.
  - Un'entry del log di tipo UNDO include il vecchio valore (BFIM) dell'item salvato (necessario per effettuare un UNDO).
  - Un'entry del log di tipo REDO include il nuovo valore (AFIM) dell'item salvato (necessario per effettuare un REDO).

Protocollo WAL

- Per permettere il recovery con l'in-place updating, le entry appropriate devono essere salvate nel log su disco prima di applicare i cambiamenti del database:

- Protocollo WAL (Write-Ahead Logging):
  1. L'AFIM non può **sovrascrivere** la BFIM di un elemento sul disco finché non sono stati memorizzati i record di log di tipo UNDO della transazione.
  2. L'operazione **commit** non può essere completata finché non sono scritti su disco tutti i record di log di tipo UNDO e di tipo REDO.

CheckPoint nel Log

- Nel log vengono utilizzate particolari entry, dette checkpoint.
  - Nel log viene registrato un **[checkpoint]** periodicamente, quando il DBMS salva su disco tutti i blocchi modificati in cache.
  - In questo modo, tutte le transazioni che hanno effettuato la **commit prima del checkpoint** non richiedono operazioni di REDO in caso di crash, poiché le loro modifiche sono già state rese permanenti.

Operazioni per un checkpoint

- Il recovery manager crea dei checkpoint ad intervalli regolari (es. ogni **m** minuti o ogni **t** transazioni terminate).

- Per creare un checkpoint si effettuano le seguenti operazioni:
  1. Sospendere temporaneamente l'esecuzione di tutte le transazioni.
  2. Scrivere su disco il contenuto di tutti i buffer modificati (scrittura forzata).
  3. Scrivere una entry di checkpoint nel log.
  4. Riprendere l'esecuzione delle transazioni sospese.

Rollback di transazioni

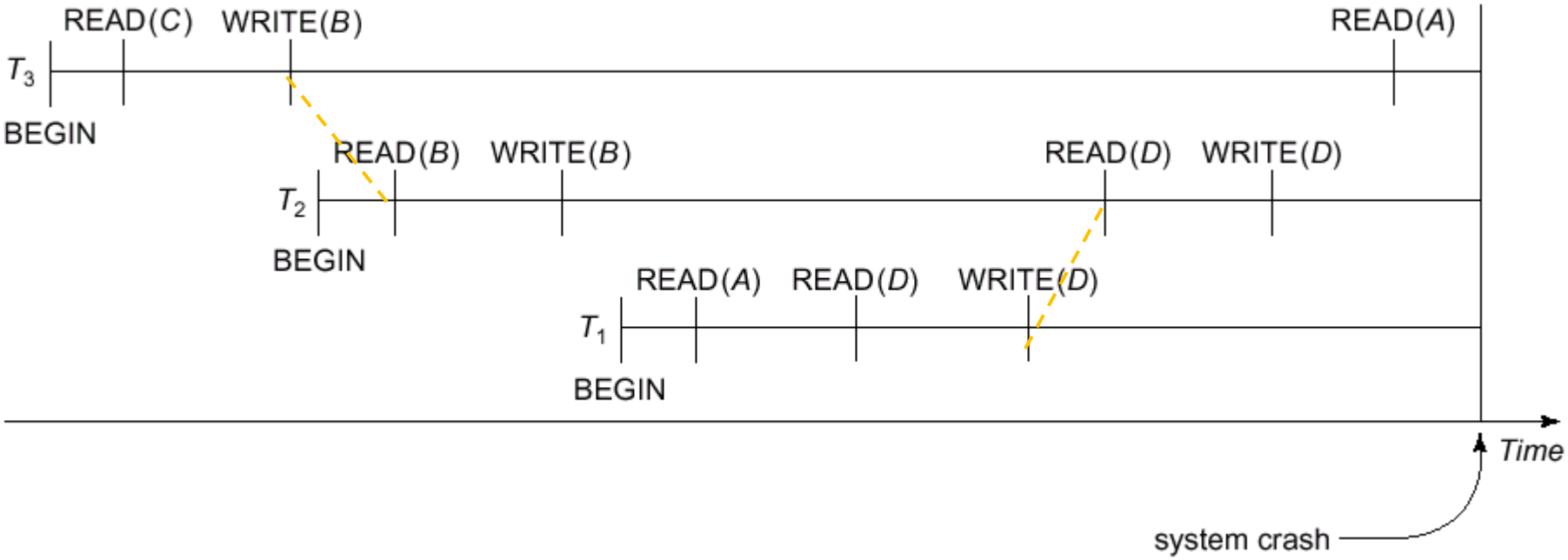
- Se una transazione fallisce per una qualsiasi ragione, può essere necessario effettuarne il rollback.

- Il rollback di una transazione **T** richiede il rollback di tutte le transazioni che hanno letto il valore di qualche dato scritto da **T**, e così via (rollback in cascata).
  - È complesso da gestire e richiede molto tempo: i meccanismi di recovery sono progettati per non usarlo.

Rollback: *Esempio*

T_1	T_2	T_3
read_item(A)	read_item(B)	read_item(C)
read_item(D)	write_item(B)	write_item(B)
write_item(D)	read_item(D)	read_item(A)
	write_item(D)	write_item(A)

Le operazioni di read/write per tre transazioni.



Operazioni effettuate prima del crash.

Rollback: *Esempio* (2)

		A	*B*	*C*	*D*
		30	15	40	20
	[start-transaction, T_3]				
	[read_item, T_3, C]				
*	[write_item, T_3, B, 15, 12]		12		
	[start-transaction, T_2]				
	[read_item, T_2, B]				
**	[write_item, T_2, B, 12, 18]		18		
	[start-transaction, T_1]				
	[read_item, T_1, A]				
	[read_item, T_1, D]				
	[write_item, T_1, D, 20, 25]				25
	[read_item, T_2, D]				
**	[write_item, T_2, D, 25, 26]				26
	[read_item, T_3, A]				
	← system crash				

Il System log al momento del crash.

* **T3** deve essere rolled-back, poiché non ha effettuato il commit.

** **T2** deve essere rolled-back, poiché legge il valore di item scritti da **T3**.

Tecniche di recovery basate su aggiornamento differito

Protocollo di Deferred Update

- Sappiamo che con questa tecnica i dati non sono aggiornati fisicamente fino all'esecuzione della commit di una transazione.
- Definiamo un protocollo di deferred update:
  1. Una transazione non può cambiare il database finché non raggiunge il punto di commit.
  2. Una transazione non raggiunge il punto di commit finché tutte le sue operazioni di aggiornamento non sono registrate nel log e il log è scritto su disco.
- L'algoritmo è NO-UNDO / REDO:
  - Non è mai necessario l'UNDO.
  - Il REDO è richiesto solo se il crash si ha dopo il commit, ma prima dell'aggiornamento del DB.

Recovery con Deferred Update in ambiente Monoutente (Single-user)

- In questo caso l'algoritmo di recovery è abbastanza semplice: l'algoritmo RDU_S (**R**ecovery usando la **D**eferred **U**pdate in ambiente **S**ingle-user) chiama una procedura REDO per rieseguire delle operazioni di write_Item.
- La procedura RDU_S mantiene due liste di transazioni:
  1. transazioni committed a partire dall'ultimo checkpoint,
  2. transazioni attive (contiene al più una transazione, perché il sistema è single-user).

Algoritmo RDU_S

- Algoritmo RDU_S:
  - Applicare l'operazione REDO a tutte le operazioni write_item delle transazioni committed nel log, nell'ordine in cui sono scritte nel log.
  - Rilanciare la transazione attiva.

- REDO (WRITE-OP):
  - Per effettuare il REDO dell'operazione WRITE-OP esaminare la sua entry nel log
    [write_item, T, X, new value]
    e porre il valore dell'elemento X a new-value (l'AFIM).

Algoritmo RDU_S: *Esempio*

T_1
read_item(A)
read_item(D)
write_item(D)

T_2
read_item(B)
write_item(B)
read_item(D)
write_item(D)

[start-transaction, T_1]
[write_item, T_1, D, 20]
[commit, T_1]
[start-transaction, T_2]
[write_item, T_2, B, 10]
[write_item, T_2, D, 25] ← system crash

Le operazioni di read/write per due transazioni.

Il System log.

- L'algoritmo rieffettuerà l'operazione [write_item, T_1, D, 20], poiché **T_1** era committed.
- **Nota:** La REDO è idempotente, perché se il sistema fallisce durante il recovery, deve essere possibile rifare il recovery, ed il risultato, con o senza crash, deve essere lo stesso.

Recovery con Deferred Update in ambiente Multiutente

- In ambienti multiutente, i processi di recovery e di controllo della concorrenza sono interrelati:
  - Maggiore è il grado concorrenza, più tempo viene impiegato per effettuare il recovery.

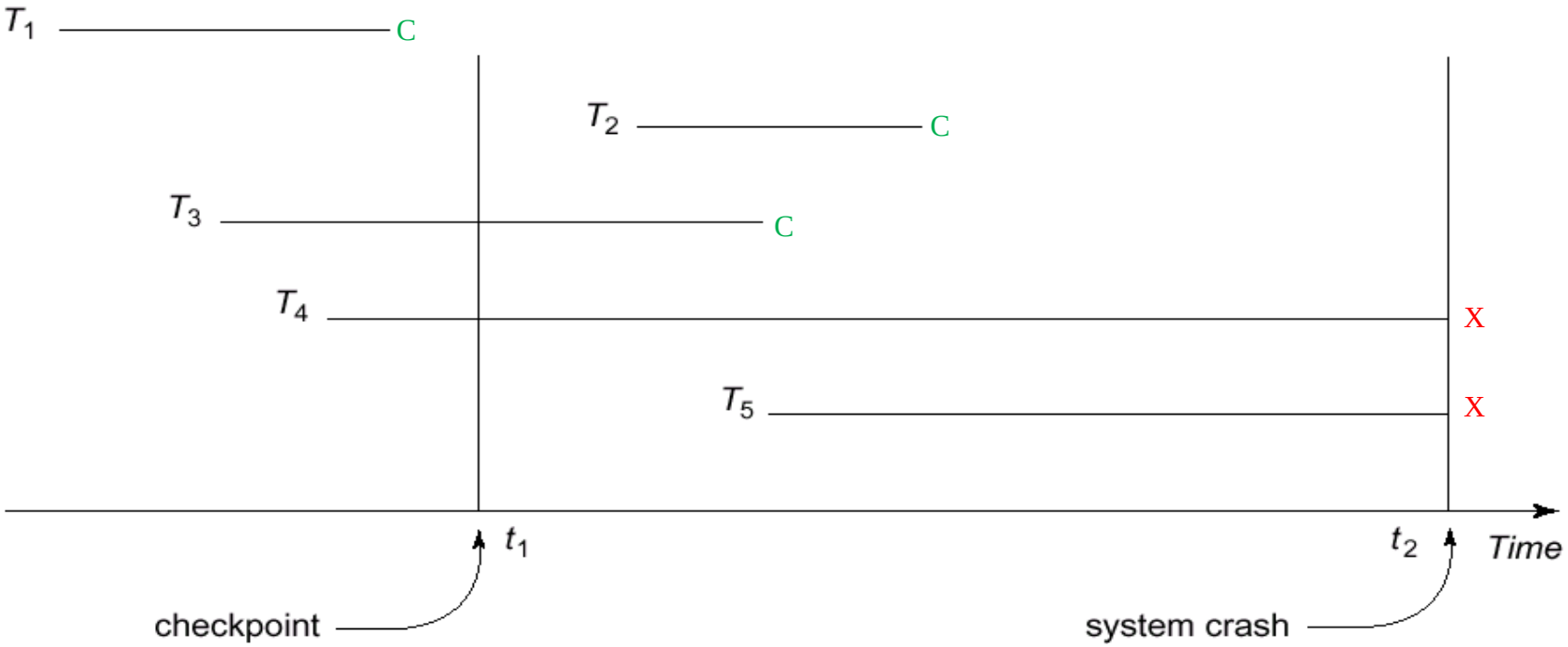
- Consideriamo un sistema multiutente così gestito:
  - Controllo della concorrenza 2PL stretto (two-phase locking): lock mantenuto fino al punto di commit.

Algoritmo RDU_M

- Usa due liste di transazioni:
  1. Transazioni committed **T** a partire dall'ultimo checkpoint.
  2. Transazioni attive **T'**.

- Algoritmo RDU_M:
  - Fare il REDO di tutte le operazioni di write_Item delle transazioni **T** committed nel log, nell'ordine in cui sono state scritte nel log.
  - Le transazioni **T'** attive e non ancora committed devono essere rilanciate.

Algoritmo RDU_M: *Esempio*



- Non c'è bisogno di fare il REDO delle write per $\mathbf{T_1}$, poiché è stata committed (nel DB) prima del crash.
- Si deve effettuare il REDO delle write di $\mathbf{T_2}$ e $\mathbf{T_3}$.
- $\mathbf{T_4}$ e $\mathbf{T_5}$ devono essere rilanciate.

Miglioramento dell'algoritmo

- Se un elemento X è stato aggiornato più volte da transazioni committed a partire dall'ultimo checkpoint, è sufficiente effettuare il REDO solo dell'ultimo aggiornamento.

Svantaggi del Deferred Update

- Limitazione dell'esecuzione concorrente delle transazioni dovuto al 2PL stretto.

- Eccessivo spazio richiesto per memorizzare gli elementi aggiornati prima del commit di una transazione.

Vantaggi del Deferred Update

- Se una transazione è abortita, viene risottomessa senza che sia stato alterato il DB su disco.
  - Non è necessario il rollback.

- Una transazione non leggerà mai un dato che sia stato modificato da una transazione non committed (2PL stretto).
  - È escluso il **cascading rollback**.

Tecniche di Recovery basate su Immediate Update

- Quando una transazione effettua un comando di aggiornamento:

1. L'operazione viene registrata nel log (write-ahead logging protocol).
2. L'operazione viene applicata nel DB.

Necessità di rollback.

Tipi di Immediate Update

- Si dividono in due categorie:
  - Se tutti gli aggiornamenti di una transazione sono riportati nel DB prima del commit, non è mai richiesto il REDO (algoritmo di recovery di tipo UNDO/NO-REDO).
  - Se la transazione raggiunge il commit prima che tutti gli aggiornamenti siano riportati nel DB può essere necessario il REDO (algoritmo di recovery di tipo UNDO/REDO).

Algoritmo RIU_S

- **R**ecovery usando l' **I**mmediate **U**pdate in ambiente **S**ingle-user
- Usa due liste di transazioni:
  1. Transazioni committed **T** a partire dall'ultimo checkpoint.
  2. Transazione attiva **T'**.

- Algoritmo RIU_S:
  - Eseguire l'UNDO delle operazione write_item della transazione attiva **T'** contenuta nel log.
  - Eseguire il REDO delle operazioni write_item delle transazioni committed **T** nell'ordine scritto nel log.

Algoritmo RIU_M

- Usa due liste di transazioni:
  1. Transazioni committed **T** a partire dall'ultimo checkpoint.
  2. Transazioni attive **T'**.

- Algoritmo RIU_M:
  - Eseguire l'UNDO delle operazione write_item delle transazioni attive **T'** contenute nel log nell'ordine inverso rispetto a quello del log.
  - Eseguire il REDO delle operazioni write_item delle transazioni committed **T** nell'ordine scritto nel log.

Recovery nei sistemi Multidatabase

- Transazione Multidatabase: *una transazione che richiede l'accesso a database multipli*.

- Questi DB possono essere gestiti da DBMS differenti (relazionali, OO, gerarchici, ...):
  - Meccanismo di recovery a due livelli:
    1. Local recovery manager.
    2. Global recovery manager (*coordinatore*).

Protocollo di commit a due fasi: *fase 1*

1. Tutti i database coinvolti dalla transazione segnalano al coordinatore di aver completato la loro parte.
2. Il coordinatore manda ad ogni partecipante il messaggio ***"prepare for commit"***.
3. Ogni partecipante forza su disco tutte le informazioni per il recovery locale e risponde ***"OK"*** al coordinatore.

   Se per qualche ragione non può fare il commit risponde ***"not OK"***.

Protocollo di commit a due fasi: *fase 2*

- Se il coordinatore riceve ***"OK"*** da tutti i partecipanti:
  - Manda un comando di commit ai DB coinvolti.
  - Ogni partecipante segnala il **[commit]** nel system log, ove necessario, ed aggiorna il DB.
- Se qualche partecipante ha fornito un ***"not OK"***:
  - La transazione fallisce.
  - Il coordinatore invia un messagio **UNDO** ai partecipanti.

Backup e Recovery di Database da failure catastrofici

Backup-Restore del db

- La principale tecnica è quella del back-up di database.
- Periodicamente il **DB** e il **system log** sono ricopiati su un device di memoria economico.
  - Il system log è back-upped più di frequente del DB intero.
- In caso di failure catastrofico, tutto il DB viene ricaricato su dischi e seguendo il system log, gli effetti delle transazioni committed vengono ripristinati.

Backup di file di Log

- Per non perdere tutte le transazioni effettuate dall'ultimo back-up, i file di log sono ricopiati molto frequentemente.

- È possibile fare ciò grazie alle ridotte dimensioni di tali file rispetto alla taglia dell'intero database.