Corso di Laurea
in
Ingegneria Informatica
"*Basi di dati*"
a.a. 2019-2020

Docente: Gigliola Vaglini
Docente laboratorio SQL: Francesco Pistolesi

1

Lezione 8

Gestione delle transazioni

2

Esempio di sistema informativo

3

3

Sistemi mono o multiutente

- Un criterio per classificare un sistema di basi di dati è il numero degli utenti che possono fruirne simultaneamente
- Un DBMS è monoutente se il sistema può essere usato al massimo da un utente alla volta
- Un DBMS è multiutente se invece può essere usato contemporaneamente da più utenti, generando accessi concorrenti alla base di dati
- La maggior parte dei DBMS è multiutente

4

4

Multitasking

- L'accesso alla base di dati e l'utilizzo dei sistemi di elaborazione da parte di più utenti contemporaneamente è possibile grazie al concetto di multitasking
- Il multitasking consente al calcolatore di eseguire più programmi (o meglio, processi) nello stesso momento

5

5

Multitasking

- Se esiste una sola CPU questa è in realtà in grado di eseguire al massimo un processo alla volta, tuttavia i sistemi operativi multitasking eseguono alcuni comandi di un processo, poi lo sospendono per eseguirne altri, e così via.
- L'esecuzione di un processo viene ripresa nel punto in cui era stata sospesa ogniqualvolta il processo torna ad usare la CPU.

6

6

Modalità interleaved o concorrente

- L'esecuzione concorrente dei processi risulta quindi interleaved (alternata)
- Se il sistema è dotato di più CPU è possibile realizzare una qualche forma di elaborazione parallela di più processi

7

7

La transazione: definizione

- Una transazione identifica una unità elementare di lavoro svolta da un'applicazione, cui si vogliono associare particolari caratteristiche di correttezza, robustezza e isolamento. OPPURE
- Una transazione è un programma (inteso come successione di comandi/operazioni) in esecuzione che forma un'unità logica di elaborazione sulla base di dati.

8

8

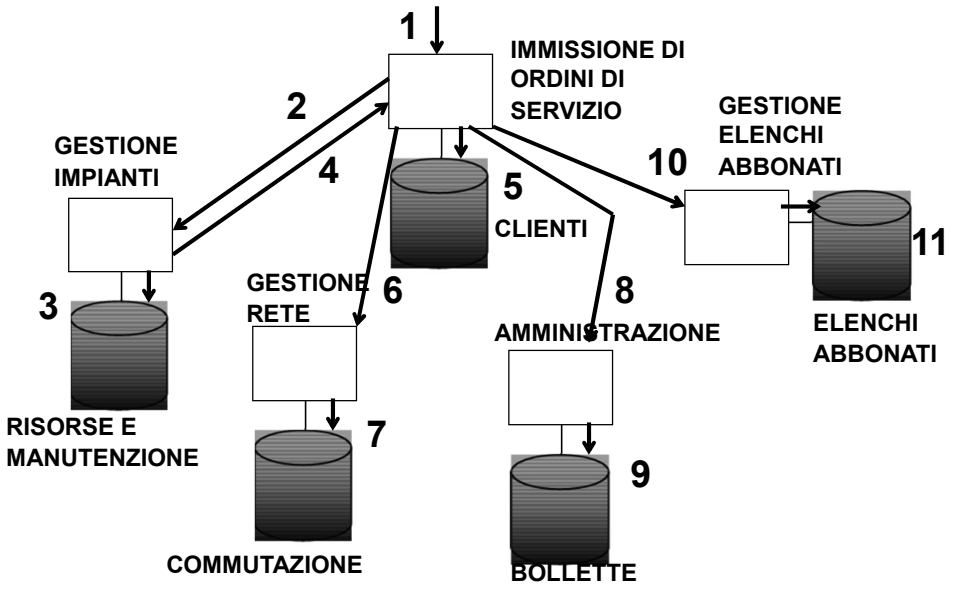
La transazione (cont.)

- Una transazione comprende una o più operazioni di accesso alla base di dati
  - inserimenti, cancellazioni, modifiche o interrogazioni.
- Un sistema che mette a disposizione un meccanismo per la definizione e l'esecuzione di transazioni è detto sistema transazionale
- Un **sistema transazionale** è in grado di definire ed eseguire transazioni per conto di un certo numero di applicazioni concorrenti

9

9

Esempio di transazione



10

10

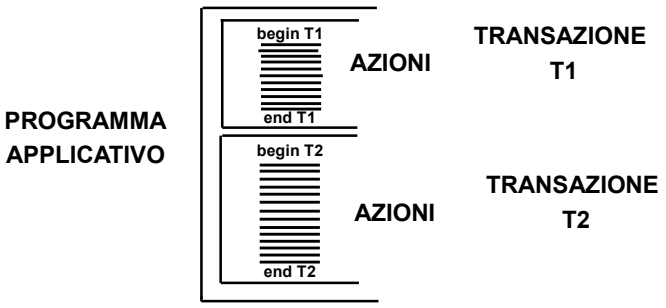
Definizione di transazione

- Transazione: parte di programma caratterizzata da un inizio (**begin-transaction**, `start transaction` in SQL), una fine (**end-transaction**, non esplicitata in SQL) e al cui interno deve essere eseguito una e una sola volta uno dei seguenti comandi
  - **commit work** per terminare correttamente
  - **rollback work** per abortire la transazione

11

11

Applicazioni e transazioni



12

12

Esempio di transazione

```
start transaction;
update ContoCorrente
  set Saldo = Saldo + 10 where
  NumConto = 12202;
update ContoCorrente
   set Saldo = Saldo – 10 where
  NumConto = 42177;
commit work;
```

13

13

Una transazione con varie decisioni

```
start transaction;
update ContoCorrente
  set Saldo = Saldo + 10 where NumConto =
  12202;
update ContoCorrente
  set Saldo = Saldo – 10 where NumConto =
  42177;
select Saldo as A
  from ContoCorrente
  where NumConto = 42177;
if (A>=0)  then commit work
          else rollback work;
```

14

14

Proprietà delle transazioni

- Proprietà "ACID"
  - Atomicità
  - Consistenza
  - Isolamento
  - Durata (persistenza)

15

15

Atomicità

- Una transazione non può lasciare la base di dati in uno stato intermedio
  - un guasto o un errore prima del commit debbono causare l'annullamento delle operazioni svolte
  - un guasto o errore dopo il commit non deve avere conseguenze; se necessario vanno ripetute le operazioni già fatte
- L'esito normale è il Commit : è il più frequente (99% ?)
  - L'abort (o rollback) può essere richiesto dall'applicazione = suicidio, oppure dal sistema (violazione dei vincoli, concorrenza) = omicidio

16

Consistenza

- La transazione rispetta i vincoli di integrita'
  - se lo stato iniziale e' corretto
  - anche lo stato finale e' corretto
- Conseguenza
  - All'inizio ed alla fine della transazione il sistema e' in uno stato "consistente"
  - Durante l'esecuzione della transazione stessa il sistema puo' temporaneamente essere in uno stato inconsistente

17

Isolamento

- La transazione non risente degli effetti delle altre transazioni concorrenti
  - l'esecuzione concorrente di una collezione di transazioni deve produrre un risultato che si potrebbe ottenere con una esecuzione sequenziale
- Conseguenza: una transazione non espone i suoi stati intermedi

18

Durabilità (Persistenza)

- Gli effetti di una transazione andata in commit non vanno perduti ("durano per sempre"),
- Conseguenza
  - I guasti non hanno effetto sullo stato del database: uno stato consistente sarà sempre recuperato

19

Stati delle transazioni

- Una transazione è sempre in uno dei seguenti stati:
  - active: dopo il begin-transaction si è in uno stato in cui è possibile eseguire operazioni di R/W
  - partially committed: lo stato raggiunto dopo che è stata eseguita l'ultima istruzione (end-transaction); il controllore dell'affidabilità assicura che un errore di sistema non comporterà l'impossibilità di registrare i cambiamenti della transazione in modo permanente, se la verifica ha successo si passa allo stato
  - committed
  - failed: lo stato raggiunto dopo aver determinato che l'esecuzione non può procedere normalmente (errore SW o verifica fallita del controllore dell'affidabilità)
  - aborted: la transazione ha subito un rollback e la base di dati e' stata ripristinata allo stato precedente l'inizio della transazione

20

20

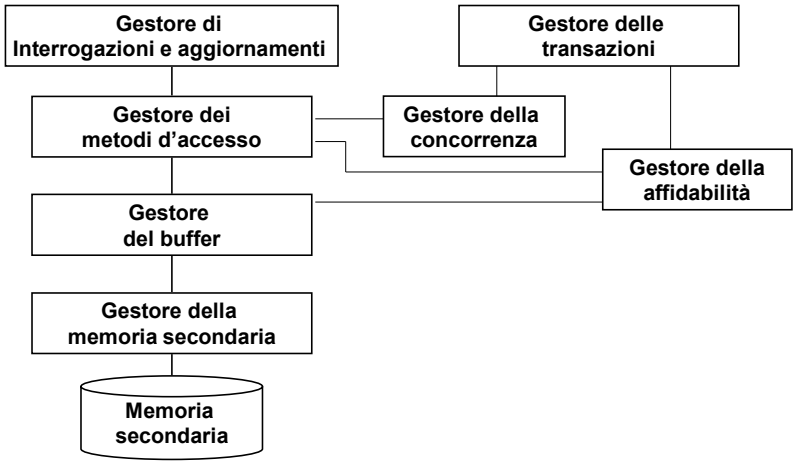
Transazioni e moduli di DBMS

- Atomicità e durabilità
  - Gestore dell'affidabilità
- Isolamento:
  - Gestore della concorrenza
- Consistenza:
  - Gestore dell'integrità a tempo di esecuzione

21

Gestore degli accessi e delle interrogazioni

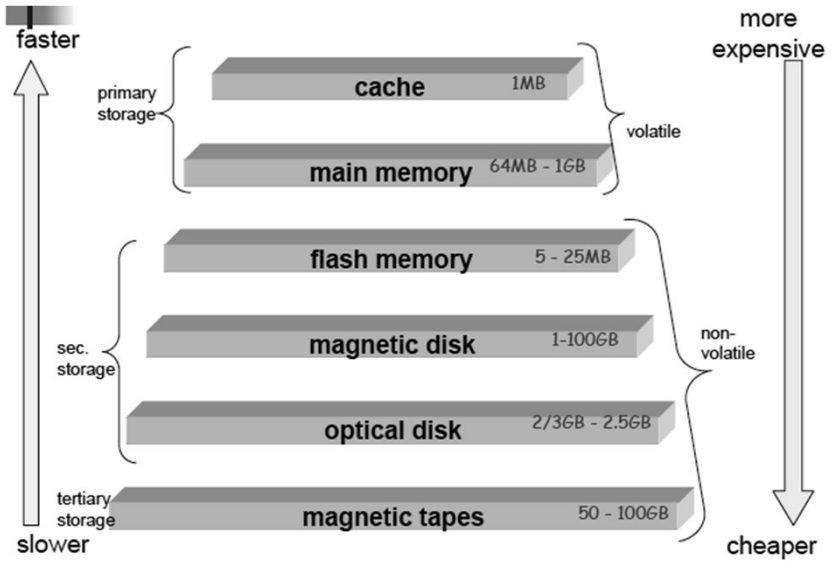
Gestore delle transazioni



22

22

La gerarchia di memoria



23

23

Memoria principale e secondaria

- I dispositivi di memoria secondaria sono organizzati in **blocchi** di lunghezza (di solito) **fissa** (ordine di grandezza: alcuni KB)
- Le uniche operazioni sono la lettura e la scrittura dei dati di un blocco
- La memoria principale è organizzata in pagine

24

24

Cos'è il buffer

- I programmi possono fare riferimento solo a dati in memoria principale, quindi i dati in memoria secondaria possono essere utilizzati solo se prima trasferiti in memoria principale (questo spiega i termini "principale" e "secondaria) serve un'interazione fra memoria principale e secondaria che limiti il più possibile gli accessi alla secondaria

25

25

Buffer management (cont)

- **Buffer**:
  - area di memoria centrale, gestita dal DBMS (preallocata) e condivisa fra le transazioni
  - organizzato in **pagine** di dimensioni pari o multiple di quelle dei blocchi di memoria secondaria (1KB-100KB)
  - Se assumiamo che coincidano pagina e blocco, il caricamento di una pagina del buffer richiede una lettura in memoria secondaria, mentre salvare una pagina corrisponde ad una scrittura

26

26

Funzioni del buffer manager

- riceve richieste di lettura e scrittura (di pagine) dalle transazioni
- le esegue accedendo alla memoria secondaria solo quando indispensabile e utilizzando invece il buffer quando possible
- Le pagine vengono mantenute finchè il buffer non è pieno e non possono essere inserite alter pagine
  - "località dei dati": è alta la probabilità di dover riutilizzare i dati attualmente in uso
  - "legge 80-20" l'80% delle operazioni utilizza sempre lo stesso 20% dei dati

27

Interfaccia

- esegue le primitive
  - ***fix***: richiesta di una pagina; richiede una lettura solo se la pagina non è nel buffer
  - ***setDirty***: comunica che la pagina è stata modificata
  - ***unfix***: indica che la transazione ha concluso l'utilizzo della pagina
  - ***force***: trasferisce in modo sincrono una pagina in memoria secondaria (su richiesta del gestore dell'affidabilità, non del gestore degli accessi)

28

8.1. Gestione dell'affidabilità

29

29

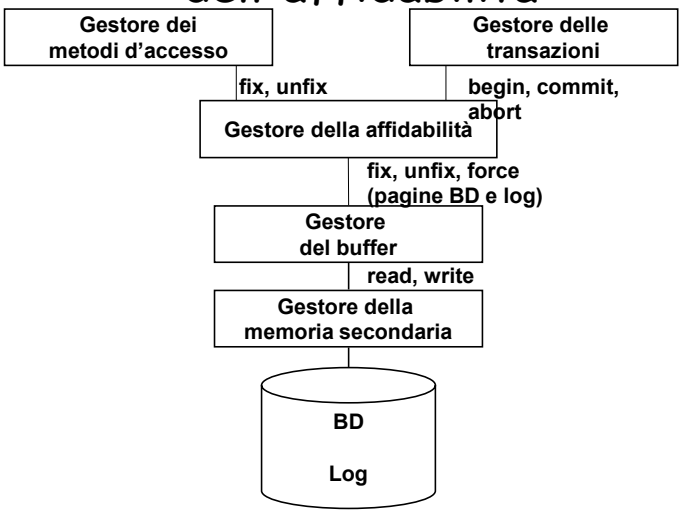
Malfunzionamenti

- Tre principali tipi di malfunzionamenti:
  - malfunzionamento del disco: le informazioni residenti su disco vengono perse (rottura della testina, errori durante il trasferimento dei dati)
  - malfunzionamenti di alimentazione: le informazioni memorizzate in memoria centrale e nei registri vengono perse
  - errori nel software: si possono generare risultati scorretti e il sistema potrebbe essere in uno stato inconsistente (errori logici ed errori di sistema)

30

30

Architettura del controllore dell'affidabilità



31

31

Gestore dell'affidabilità

- Gestisce l'esecuzione dei comandi transazionali
  - `start transaction` (B)
  - `commit work` (C)
  - `rollback work` (A)

  e le operazioni di ripristino (recovery) dopo i guasti :
  - *warm restart* e *cold restart*
- Assicura atomicità e durabilità
- Usa il **log**:
  - Un archivio permanente che registra le operazioni svolte

32

32

Tipi di memoria

- Ai fini del ripristino, le memorie vengono classificate come segue:
  - Memoria volatile: le informazioni contenute vengono perse in caso di cadute di sistema (ad es. memoria principale)
  - Memoria non volatile: le informazioni contenute sopravvivono a cadute di sistema, possono però essere perse a causa di altri malfunzionamenti (ad es. disco e nastri magnetici)
  - Memoria stabile: le informazioni contenute non possono essere perse

33

33

Persistenza delle memorie

- **Memoria centrale**: non è persistente
- **Memoria di massa**: è persistente ma può danneggiarsi
- **Memoria stabile**: memoria che non può danneggiarsi (è una astrazione):
  - perseguita attraverso la ridondanza:
    - dischi replicati
    - nastri
    - …
  - con probabilità di fallimento indipendenti!

34

34

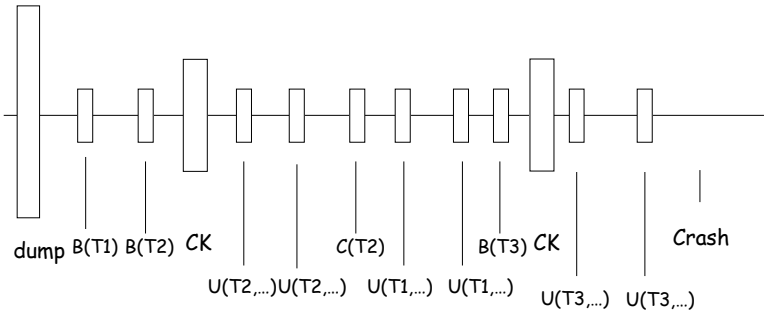
Il log

- Il log è un file sequenziale gestito dal controllore dell'affidabilità, scritto in memoria stabile
- "Diario di bordo": riporta tutte le operazioni in ordine
- Record nel log
  - *operazioni delle transazioni*
    - `begin`, B(T)
    - `insert`, I(T,*O*,*AS*)
    - `delete`, D(T,*O*,BS)
    - `update`, U(T,*O*,BS,*AS*)
    - `commit`, *C*(T), `abort`, *A*(T)
  - *record di sistema*
    - `dump`
    - `checkpoint`

35

35

Struttura del log



36

36

Log, checkpoint e dump: a che cosa servono?

- Il log serve "a ricostruire" le operazioni
- Checkpoint e dump servono ad evitare che la ricostruzione debba partire dall'inizio dei tempi
  - si usano con riferimento a tipi di guasti diversi (vedi avanti)

37

37

Scritture nel log

- I record nel log sono di due tipi
  - Record di sistema: checkpoint e dump vengono scritti dal controllore dell'affidabilità
  - Record di transazione: attività svolte dalle transazione nell'ordine in cui sono svolte (begin, commit, rollback, insert, delete, update)

38

38

Checkpoint

- Operazione che serve a "fare il punto" della situazione, in coordinamento col gestore del Buffer, semplificando le successive operazioni di ripristino:
  - ha lo scopo di registrare quali transazioni sono attive in un certo istante, cioè le transazioni "a metà strada"
  - e, dualmente, di confermare che le altre o non sono iniziate o sono finite; infatti per tutte le transazioni che hanno effettuato il commit i dati vengono trasferiti in memoria di massa

39

39

Descrizione dell'operazione Ceckpoint

  - Si sospende l'accettazione delle operazioni di commit o abort da parte delle transazioni
  - Si forza (force) la scrittura in memoria di massa delle pagine del buffer modificate da transazioni che hanno fatto commit
  - Si forza (force) la scrittura nel log di un record contenente gli identificatori delle transazioni attive
  - Si riprendono ad accettare tutte le operazioni da parte delle transazioni
- Con questo funzionamento si garantisce la persistena delle transazioni che hanno eseguito il commit.

40

40

Dump

- Copia completa ("di riserva") della base di dati
  - Solitamente prodotta mentre il sistema non è operativo
  - Salvato in memoria stabile, come *backup*
  - Un record di `dump` nel log indica il momento in cui il log è stato effettuato (e dettagli pratici, file, dispositivo, …)

41

Record di transazione

- Begin,commit,rollback: identificativo transazione (T)
- Update: T, O, BS (before state), AS (after state)
- Insert: T, O, AS
- Delete: T, O, BS

42

Significato delle operazioni di Undo e Redo

- Undo di una azione su un oggetto *O:*
  - `update`, `delete`: copiare il valore del before state (*BS*) nell'oggetto *O*
  - `insert`: eliminare *O*
- Redo di una azione su un oggetto *O:*
  - `insert`, `update`: copiare il valore dell' after state (*AS*) nell'oggetto *O*
  - `delete`: eliminare *O*
- *Idempotenza* di *undo e redo:*
  - *undo*(*undo*(*A*)) = *undo*(*A*)
  - *redo*(*redo*(*A*)) = *redo*(*A*)

43

43

Quando il controllore dell'affidabilità può consentire la modifica del log da parte delle transazioni

- **Regola Write-Ahead-Log**:
  - si scrive la parte BS dei record del log prima di effettuare la corrispondente operazione sul database
    - consente di disfare le azioni (già memorizzate) di transazioni senza commit avendo in memoria stabile un valore corretto
- **Regola Commit-Precedenza**:
  - si scrive la parte AS dei record di log prima del commit
    - consente di rifare le azioni di una transazione che ha effettuato il commit ma le cui pagine modificate non sono ancora state trascritte dal buffer manager in memoria di massa

44

44

In pratica

- Anche se le regole fanno riferimento a before state e after state, nella pratica entrambe le componenti del record di log vengono scritte assieme.
- Regole semplificate:
  - i record di log siano scritti prima dei corrispondenti record della base di dati (regola WAL);
  - i record di log siano scritti prima dell'esecuzione dell'operazione di commit (regola di commit-precedenza).

45

45

Esito di una transazione

- L'esito di una transazione è determinato irrevocabilmente quando viene scritto il record di **commit** nel log
  - una guasto prima di tale istante porta ad un undo di tutte le azioni, per ricostruire lo stato originario della base di dati
  - un guasto successivo non deve avere conseguenze: lo stato finale della base di dati deve essere ricostruito, con redo se necessario

46

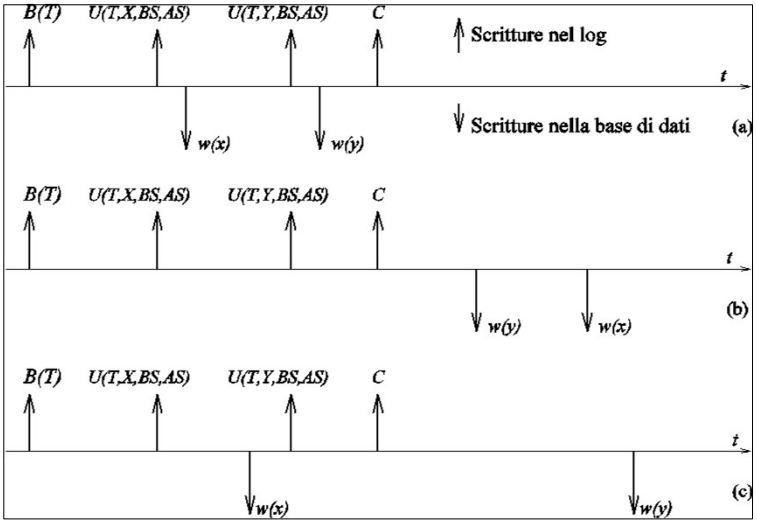
46

E la base di dati?

- Quando scriviamo nella base di dati?
  - Varie alternative

47

Scrittura nel log e nella base di dati



48

Modalità immediata

- Il DB contiene valori AS provenienti da transazioni uncommitted
- Richiede Undo delle operazioni di transazioni uncommited al momento del guasto
- Non richiede Redo

49

Modalità differita

- Il DB non contiene valori AS provenienti da transazioni uncommitted
- In caso di abort, non occorre fare niente
- Rende superflua la procedura di Undo, non ci sono scritture prima del commit.
- Richiede Redo

50

Modalità mista

- La scrittura può avvenire in modalità sia immediata che differita
- Richiede sia Undo che Redo

51

- La modalità differita di scrittura, ad esempio, pur permettendo una procedura di recovery più semplice ed efficiente, non viene molto utilizzata in pratica.
- Motivo?

52

52

- Questa è più efficiente nel recovery, ma è complessivamente meno efficiente di una in cui il gestore del buffer può decidere liberamente quando scrivere in memoria secondaria.
- E' preferibile una gestione ordinaria più efficiente rispetto ad una gestione più semplice dei guasti, poichè si assume che i guasti siano abbastanza rari.

53

In pratica

- La scrittura nella base di dati può avvenire in qualunque momento, anche prima del *commit*
- La scrittura sul log è effettuata prima della scrittura nella base di dati
- Il *commit* si considera effettuato quando il corrispondente record di log è scritto
  - prima di questa scrittura il guasto causa l'*undo* di tutte le operazioni
  - dopo, il guasto causa il *redo* di tutte le operazioni

54

Rollback di una transazione

- Quando una transazione deve essere cancellata, per un errore logico dell'operazione contenuta oppure per un'esigenza di sistema, tutte le operazioni tra il begin della transazione e l'abort devono essere disfatte, alla fine un record di abort è scritto nel log.

55

Guasti

- **Guasti "soft":** errori di programma, crash di sistema, caduta di tensione
  - si perde la memoria centrale e quindi anche il buffer
  - non si perde la memoria secondaria, cioè la base di dati e il log

  warm restart, ripresa a caldo
- **Guasti "hard":** dei dispositivi di memoria secondaria
  - si perde anche la memoria secondaria, i.e. parte della base di dati
  - non si perde la memoria stabile (e quindi il log)

  cold restart, ripresa a freddo
- **La perdita del log è considerato un evento catastrofico e quindi non è definita alcuna strategia di recupero.**

56

Modello fail-stop

1. L'individuazione di un guasto forza l'arresto completo delle transazioni
2. Il sistema operativo viene riavviato
3. Viene avviata una procedura di restart
4. Al termine del restart il buffer è vuoto, ma le transazioni possono ripartire

57

Processo di restart

- Obiettivo: classificare le transazioni in
  - completate (tutti i dati in memoria)
  - attive ma con il commit (vanno rifatte, redo)
  - attive senza commit (vanno annullate, undo)

58

Ripresa a caldo

Quattro fasi:

- trovare l'ultimo checkpoint (ripercorrendo il log a ritroso)
- costruire gli insiemi *UNDO* (transazioni da disfare) e *REDO* (transazioni da rifare)
  - UNDO riguarda le transazioni attive ma non committed, REDO le transazioni che sono committed prima del guasto
- ripercorrere il log all'indietro, fino alla più vecchia azione delle transazioni in *UNDO* e *REDO*, disfacendo tutte le azioni delle transazioni in *UNDO*
- ripercorrere il log in avanti, rifacendo tutte le azioni delle transazioni in *REDO*

59

59

Esempio 1

- Considerando il seguente log di un sistema di gestione di basi di dati, illustrare dettagliatamente i passi da compiere per effettuare la ripresa a caldo.

- B(T1), B(T2), I(T1,O1,A1), D(T2,02,B2), B(T3), B(T4) U(T3,O3,B3,A3), C(T2), CK(...), U(T1,O4,B4,A4), A(T3), B(T5), D(T4,05,B5), C(T1), C(T4), I(T5,O6,A6), GUASTO

60

60

Passo 1

- *individuare le transazioni attive al checkpoint*
- Considerando i seguenti record antecedenti il checkpoint: B(T1), B(T2), B(T3), B(T4), C(T2)
  le transazioni attive sono: T1, T3 e T4

61

61

Passo 2

- *compilare l'elenco delle transazioni da disfare e rifare*
-

RECORD	UNDO	REDO
CK(T1,T3,T4)	T1, T3, T4	
B(T5)	T1, T3, T4, T5	
C(T1)	T3, T4, T5	T1
C(T4)	T3, T5	T1, T4

62

62

Passo 3

- *Ripristinare il Sistema*
- Le transazioni da disfare sono T3 e T5, mentre quelle da rifare sono T1 e T4.
  Riempio la tabella delle UNDO partendo dal fondo e considerando le operazioni di T3 e T5

RECORD	AZIONE
I(T5,O6,A6)	delete O6
U(T3,O3,B3,A3)	O3 := B3

63

63

Passo 3 (cont)

- Si riempe la tabella delle REDO partendo dall'inizio e considerando le operazioni di T1 e T4

RECORD	AZIONE
I(T1,O1,A1)	insert O1 := A1
U(T1,O4,B4,A4)	O4 := A4
D(T4,O5,B5)	delete O5

64

64

Domanda

- Le transazioni attive da inserire nel checkpoint si possono fissare
  - Rifiutando nuovi commit, oppure
  - Rifiutando nuovi begin-transaction e aspettando (quindi accettando) la conclusione (commit o abort) delle transazioni già inziate
- Quali sono le differenze nella gestione delle riprese a caldo?

65

65

- Nel primo caso si attua la strategia appena mostrata
- Nel secondo caso il check point non conterrà nessuna transazione, non ci sono transazioni attive. Nella ripresa a caldo è sufficiente rieseguire tutte le operazioni che seguono il record di check point.
- Però la base di dati viene fermata (non si accettano begin-transaction) ogni volta che si deve eseguire un check point, con un conseguente degrado delle prestazioni.
- Quindi la prima soluzione sarà normalmente preferibile, in quanto è opportuno avere delle buone prestazioni per la maggior parte del tempo piuttosto che al verificarsi di eventi che richiedono una ripresa a caldo, considerati rari.

66

66

Transazioni abortite

- Le operazioni derivanti dal rollback di una transazione possono essere inserite nell'insieme UNDO e fatte al momento del recovery,
- oppure essere eseguite al momento dell'abort ed essere inserite
- nell'insieme di REDO al momento del recovery da un guasto

67

Ripresa a freddo

- Ci si riporta al record di dump più recente nel log e si ripristina la parte di dati deteriorata
- Si eseguono le operazioni registrate sul giornale sulla parte deteriorata fino all'istante del guasto
- Si esegue una ripresa a caldo

68

Esempio 2

- Considerando il seguente log di un sistema di gestione di basi di dati, illustrare dettagliatamente i passi da compiere per effettuare la ripresa a freddo dopo un guasto di dispositivo che interessa gli oggetti O1,O2,O3.

- Dump, B(T1), B(T2), I(T1,O1,A1), D(T2,02,B2), B(T3), B(T4) U(T3,O3,B3,A3), C(T2), CK(...), U(T1,O4,B4,A4), A(T3), B(T5), D(T4,05,B5), C(T1), C(T4), I(T5,O6,A6), GUASTO

69

69

- Insert O1=A1
- Delete(O2)
- O3=A3
- commit(T2)
- Abort(T3)
- Commit(T1)
- Commit(T4)
- Ripresa a caldo

70

70