

Gestione delle Transazioni

P. Rullo

Cos'è una transazione

- È una unità logica di elaborazione composta da una sequenza di operazioni di lettura/scrittura sulla BD
- Proprietà
 - *Atomicità*: è indivisibile
 - *Consistenza*: lascia la BD in uno stato consistente
 - *Isolamento*: viene eseguita indipendentemente dalla presenza di altre transazioni concorrenti
 - *Durabilità*: i suoi effetti sulla BD sono permanenti – se si è conclusa correttamente

Cos'è una transazione

Transaction BONIFICO(IN X, Y, Z INTEGER)

```
BEGIN TRANSACTION //trasferisci Z€ dal cc X al cc Y
Select saldo Into S, MaxScoperto into W
From Conto
Where num_cc=X;
If S-Z >= W
{
    UPDATE Conto
        SET saldo = saldo-Z
        WHERE num_cc=X;

    UPDATE Conto
        SET saldo = saldo+Z
        WHERE num_cc=Y; }
COMMIT
```

Transaction BONIFICO(IN X, Y, Z INTEGER)

```
BEGIN TRANSACTION
UPDATE Conto, SET saldo= saldo - Z
WHERE num_cc=X;

UPDATE Conto, SET saldo = saldo+Z
WHERE num_cc=Y;

SELECT saldo into S, MaxScoperto into W
FROM Conto
WHERE num_cc=X;

if S < W
    Rollback Work;
COMMIT
```

Perché le transazioni

- Per far sì che una sequenza di operazioni di scrittura/lettura sulla BD costituisca una unità indivisibile bisogna usare Begin e Commit
 - *Begin Transaction*
 - *Select(X)*
 - *Update(X)*
 - ..
 - *Update(Y)*
 - *Commit*
- Tutte le operazioni tra Begin e Commit sono viste come un tutt'uno dal punto di vista logico, quindi, è necessario che vengano eseguite *tutte o nessuna*
- In assenza del Begin e del Commit, ognuna delle suddette operazioni sarebbe considerata come una operazione a sé stante

Perché le transazioni

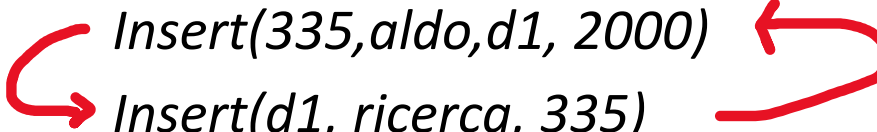
- **Transaction BONIFICO(IN X, Y, Z INTEGER)**
- BEGIN TRANSACTION
- Select saldo Into S, MaxScoperto into W
- From Conto
- Where num_cc=X;
- If S-Z >= W
- UPDATE Conto
- SET saldo = saldo-Z
- WHERE num_cc=X;
-
- UPDATE Conto
- SET saldo = saldo+Z
- WHERE num_cc=Y;
- COMMIT

- I due UPDATE formano un'unica operazione logica: trasferimento dell'ammontare Z dal conto X al conto Y

Perché le transazioni

- L'inserimento di un dipartimento e del rispettivo direttore nella seguente BD è una operazione atomica. Infatti, c'è un problema di ciclicità per il rispetto dei vincoli di integrità

- *Imp(matr, nome, dip*, stip)*
- *Dip(cod, nome, dir*)*

*Insert(335, aldo, d1, 2000)*
Insert(d1, ricerca, 335)

- **Begin transaction**

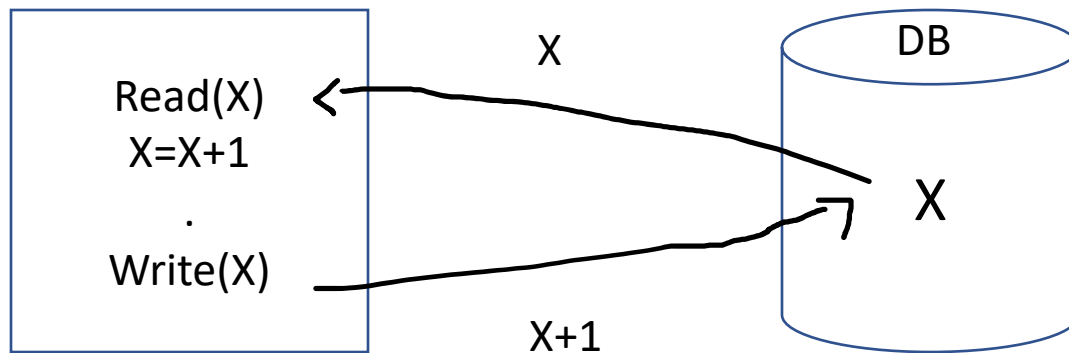
- Insert Imp(335, aldo, d1, 2.000)
- Insert Dip(d1, ricerca, 335)

- **Commit**

- Durante l'esecuzione di una transazione, la BD può transitare attraverso stati non consistenti

Rappresentazione delle transazioni

- Read(X): lettura di un elemento X della BD e copia nella variabile X della transazione
- Write(X): scrittura del valore della variabile X della transazione nell'elemento della BD denominato X




Gestione dell'affidabilità

Il controllo di affidabilità: perché serve?



- L'esecuzione di una transazione può essere pregiudicata da eventi anomali causati da malfunzionamenti
 - *Transaction failure*: la transazione abortisce
 - *System failure*: guasto hw o sw che causa l'interruzione di tutte le transazioni
- Un malfunzionamento potrebbe compromettere
 - l'*atomicità* delle transazioni
 - la *persistenza* degli aggiornamenti

Il controllo di affidabilità: perché serve?

- **Transaction BONIFICO(IN X, Y, Z INTEGER)**
- BEGIN TRANSACTION
 - Select saldo Into S, MaxScoperto into W
 - From Conto
 - Where num_cc=X;
- If $S-Z \geq W$
- {
 - UPDATE Conto
 - SET saldo = saldo-Z
 - WHERE num_cc=X;
- 
- - UPDATE Conto
 - SET saldo = saldo+Z
 - WHERE num_cc=Y; }
- COMMIT

- *Transaction failure*: se la transazione subisce una interruzione dopo il primo UPDATE, si crea la seguente situazione:
 - L'ammontare Z è stato detratto dal cc=X, ma non è mai giunto a destinazione sul cc=Y
 - Risultato: c'è un ammanco di Z€!!!
- Ciò perché è stata violata la atomicità
- È necessario quindi annullare l'effetto dell'UPDATE

Il controllo di affidabilità: perché serve?

- **Transaction BONIFICO(IN X, Y, Z INTEGER)**
- BEGIN TRANSACTION
 - Select saldo Into S, MaxScoperto into W
 - From Conto
 - Where num_cc=X;
- If S-Z >= W
- {
 - UPDATE Conto
 - SET saldo = saldo-Z
 - WHERE num_cc=X;
- 
- - UPDATE Conto
 - SET saldo = saldo+Z
 - WHERE num_cc=Y; }
- COMMIT
- 

System failure: tutte le transazioni falliscono

1. Il sistema subisce una interruzione prima del COMMIT: è necessario annullare gli aggiornamenti fatti per garantire l'*atomicità*
2. Il sistema subisce una interruzione dopo il COMMIT: potrebbe essere necessario rifare entrambi gli UPDATE, per garantire sia l'*atomicità*, sia che gli effetti degli aggiornamenti eseguiti siano *persistenti* nella BD

Il Sistema di Gestione dell'Affidabilità

- Il DBMS deve essere dotato di meccanismi che garantiscono il principio 'tutto o niente', cioè, o
 1. tutti gli aggiornamenti contenuti in una transazione sono eseguiti ed i loro effetti sono resi permanenti, oppure
 2. la transazione non viene completata e non si ha alcun effetto sulla BD
- Il controllo di affidabilità garantisce entrambe le proprietà di *atomicità* e *persistenza* (o durabilità)
- Il Sistema di Gestione dell'Affidabilità di un DBMS implementa le politiche per il controllo dell'affidabilità
- Queste dipendono in maniera stretta dalla politica di gestione del buffer

Il file di Log

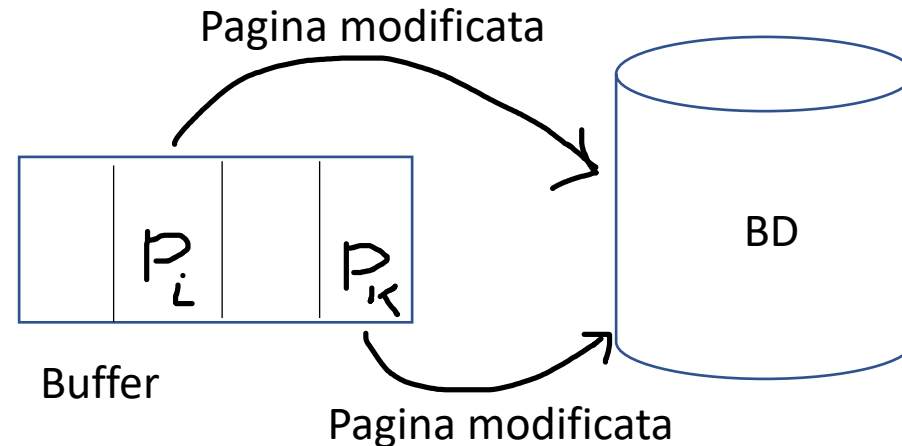
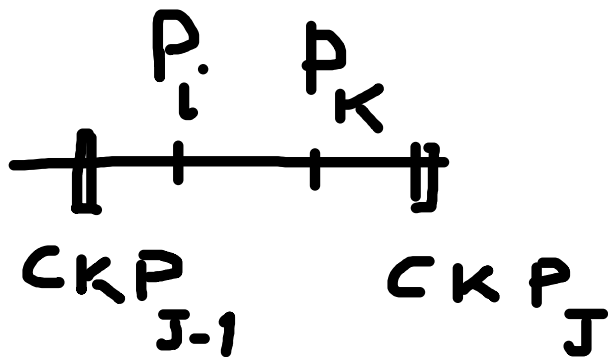
- File di Log (giornale): file sequenziale che contiene le operazioni di aggiornamento fatte dalle transazioni

T	begin/commit	Pid	before(P)	after(P)
...	
T1	begin			
T2		P20	(xyz, 35)	(xyz, 40)
T1		P40	(abc, 20, 30)	(def, 50, 30)
T1	commit			
...	

- **Pid**: identificativo della pagina
- **Before(P)**: pagina prima dell'aggiornamento
- **After(P)**: pagina dopo l'aggiornamento

Checkpoint

- Periodicamente il sistema dell'affidabilità esegue un *checkpoint (CKP)*: tutte le pagine nel buffer modificate dopo l'ultimo CKP vengono copiate su disco
- In tal modo, gli aggiornamenti fatti fino a quel momento sono resi persistenti



Transaction failure - UNDO

- Un *transaction failure* si verifica quando una singola transazione si interrompe prima di raggiungere il COMMIT
- In tal caso, bisogna annullare l'effetto degli aggiornamenti fatti - operatore UNDO
- Infatti, le pagine modificate nel buffer verranno prima o poi copiate nella BD, o attraverso un CKP o attraverso le primitive di gestione del buffer—politica *steal* di gestione del buffer
- Pertanto, il gestore dell'affidabilità, grazie al file di Log, deve ripristinare uno stato consistente della BD eliminando l'effetto degli aggiornamenti prodotti dalla transazione
- UNDO: per annullare gli aggiornamenti fatti (fino al momento dell'interruzione), il sistema scandisce a ritroso il file di Log e ripristina le pagine prima che fossero modificate – before(P). Dopo il CKP

Transaction failure - UNDO

File di Log

T	begin/commit	Pid	before(P)	after(P)
...		...		
T1	begin			
T1		P20	(xyz, 35)	(xyz, 40)
T2		P40	(abc, 20, 30)	(def, 50, 30)
T2		P25	(zzz, abc)	(xxx, abc)
....			

t_0

- La transazione T1 fallisce al tempo t_0
- Viene fatto l'UNDO, ripristinando nel DB la pagina P20, i.e., $P20 = \text{before}(P20)$

System failure – UNDO/REDO

- Nel caso di *system failure*, il sistema si blocca, tutte le transazioni in esecuzione vengono interrotte, e le pagine nel buffer vengono perse
- Per il ripristino della BD bisogna compiere azioni sia per le transazioni che hanno raggiunto il Commit, sia per quelle che non lo hanno raggiunto
- Nel primo caso si procede ad un REDO
- Nel secondo caso si procede ad un UNDO

System failure - UNDO

t_1 →

T	begin/commit	Pid	before(P)	after(P)
T1	begin			
T2		P20	(xyz, 35)	(xyz, 40)
T1		P40	(abc, 20, 30)	(def, 50, 30)
T1	commit			
T2		P25	(zzz, abc)	(xxx, abc)
...		...		

- Il sistema si ferma al tempo t_1
- Si fa l'UNDO di T2, in quanto NON ha raggiunto il punto di Commit – a differenza di T1
- A tal fine, vengono copiate nel DB le pagine before(P25) e before(P20)

System failure - REDO

- Quando una transazione raggiunge il punto di Commit, ha eseguito tutti gli aggiornamenti e li ha registrati nel file di Log
- Non si ha tuttavia alcuna certezza sulla persistenza degli effetti degli aggiornamenti eseguiti, in quanto le pagine modificate nel buffer di memoria potrebbero non essere state ancora copiate nella BD (politica *steal*)
- La transazione che ha eseguito il COMMIT si impegna comunque a rendere effettivi *tutti* gli aggiornamenti fatti

System failure - REDO

- Il Sistema di Gestione dell’Affidabilità deve quindi provvedere, con l’ausilio del Log, a rendere effettivi *tutti* gli aggiornamenti fatti
- Per ogni transazione T il cui record di COMMIT è nel file di Log, esegue un REDO, cioè, T viene ‘rifatta’
- A tal fine, si ricaricano le pagine modificate – after(P) nel file di Log

System failure - REDO

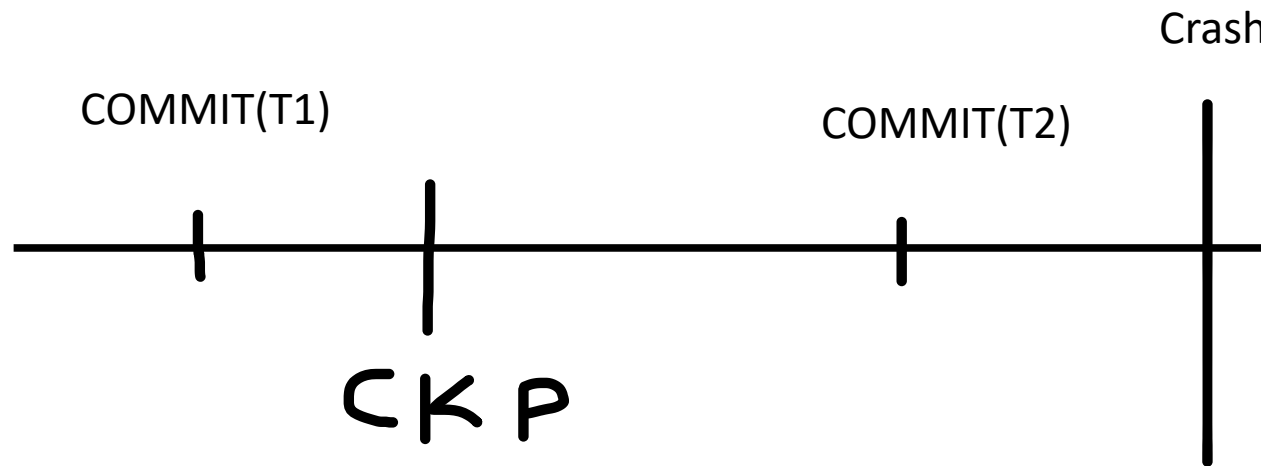
t_1 →

T	begin/commit	Pid	before(P)	after(P)
T1	begin			
T2		P20	(xyz, 35)	(xyz, 40)
T1		P40	(abc, 20, 30)	(def, 50, 30)
T1	commit			
T2		P25	(zzz, abc)	(xxx, abc)
...		...		

- Il sistema si ferma al tempo t_1
- Si fa il REDO di T1, in quanto ha raggiunto il punto di Commit
- A tal fine, viene copiata nel DB la pagina after(P40), i.e. P40=after(P40)

System failure – REDO e Checkpoint

- Tuttavia, non è necessario rifare *tutte* le transazioni che hanno raggiunto il Commit, ma solo quelle il cui Commit è stato eseguito dopo l'ultimo CKP
- Il checkpoint viene registrato nel Log, specificando quali sono le transazioni attive



- Si fa il REDO sono di T2

System failure – Ripresa a caldo

T	begin/commit	Pid	before(P)	after(P)
T2	begin			
...		...		
T4	begin			
T2		P30	(ab, 35)	(ab, 45)
T3	commit			
	CKP – {T2, T4}			
T2		P20	(xyz, 35)	(xyz, 40)
T1	begin			
T4		P(50)	(xyz, 5)	(xyz, 3)
T4	commit			
T1		P40	(abc, 20, 30)	(def, 50, 30)
T1	commit			
			

- Il sistema cade al tempo t_0
- Per le transazioni che hanno fatto il Commit prima dell'ultimo CKP non bisogna fare niente (T3) – i loro aggiornamenti sono stati resi persistenti
- Le transazioni che hanno eseguito il Commit nell'intervallo tra l'ultimo CKP e t_0 devono essere rifatte (dal CKP in poi) - REDO(T1, T4)
- Le transazioni che al tempo t_0 non hanno eseguito il Commit vanno disfatte - UNDO(T2)

CKP →

t_0 →

System failure – Ripresa a caldo

1. Si ripercorre il log a ritroso fino all'ultimo CKP
2. SetUNDO={transazioni attive al CKP}, SetREDO={}
3. Si percorre il log in avanti:
 - Per ogni Begin(T), metti T in SetUNDO
 - Per ogni Commit(T), sposta T da SetUNDO a SetREDO
4. Fase UNDO: si torna indietro sul file di Log disfacendo le transazioni in SetUNDO, fino al begin della transazione più vecchia
5. Fase REDO: si va in avanti sul file di Log rifacendo le transazioni in SetREDO – a partire dall'ultimo CKP

System failure – Ripresa a caldo

1. Si ripercorre il log a ritroso fino all'ultimo CKP
2. $\text{setUNDO} = \{T2, T4\}$, $\text{setREDO} = \{\}$
3. Si percorre il log in avanti:
 - i. $\text{Begin}(T1): \text{setUNDO} = \text{setUNDO} \cup \{T1\} = \{T1, T2, T4\}$
 - ii. $\text{Commit}(T4): \text{setUNDO} = \text{setUNDO} - \{T4\} = \{T1, T2\}$,
 $\text{setREDO} = \text{setREDO} \cup \{T4\} = \{T4\}$
 - iii. $\text{Commit}(T1): \text{setUNDO} = \text{setUNDO} - \{T1\} = \{T2\}$,
 $\text{setREDO} = \text{setREDO} \cup \{T1\} = \{T1, T4\}$
4. Fase UNDO: disfai T2 ripercorrendo il Log all'indietro fino a $\text{Begin}(T2)$
 - i. $P20 = \text{Before}(P20)$
 - ii. $P30 = \text{Before}(P30)$
5. Fase REDO: rifai T1 e T4 ripartendo dal CKP
 - i. $P50 = \text{After}(P50)$
 - ii. $P40 = \text{After}(P40)$

File di Log

T	Pid	before(P)	after(P)
T2	Begin		
...	...		
T4	Begin		
T2	P30	(ab, 35)	(ab, 45)
T3	Commit		
	CKP – {T2, T4}		
T2	P20	(xyz, 35)	(xyz, 40)
T1	Begin		
T4	P(50)	(xyz, 5)	(xyz, 3)
T4	Commit		
T1	P40	(abc, 20, 30)	(def, 50, 30)
T1	Commit		
		