

Lezione 25– Il controllo della concorrenza

Prof.ssa Maria De Marsico
demarsico@di.uniroma1.it



SAPIENZA
UNIVERSITÀ DI ROMA



- In sistemi di calcolo con una sola CPU i programmi sono eseguiti concorrentemente in modo *interleaved* (interfogliato):

la CPU può

- eseguire alcune istruzioni di un programma,
- sospendere quel programma,
- eseguire istruzioni di altri programmi,
- ritornare ad eseguire istruzioni del primo.



L'esecuzione concorrente permette un
uso efficiente della CPU



- In un DBMS la principale risorsa a cui i programmi accedono in modo concorrente è la **base di dati**
- Se sulla BD vengono effettuate **solo letture** (la BD non viene mai modificata), l'accesso concorrente non crea problemi
- Se sulla BD vengono effettuate **anche scritture** (la BD viene modificata), **l'accesso** concorrente può creare problemi e quindi deve essere **controllato**



Transazione

esecuzione di una parte di un programma che rappresenta un'unità logica di accesso o modifica del contenuto della base di dati

Proprietà delle transazioni



SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

- **ACID** deriva dall'acronimo inglese **Atomicity, Consistency, Isolation, e Durability** (Atomicità, Consistenza, Isolamento e Durabilità) ed indica le proprietà logiche che devono avere le [transazioni](#).
- Perché le transazioni **operino in modo corretto** sui dati è **necessario** che i meccanismi che le implementano soddisfino queste **quattro** proprietà:
- **atomicità**: la transazione è **indivisibile** nella sua esecuzione e la sua esecuzione deve essere **o totale o nulla, non sono ammesse esecuzioni parziali**;
- **consistenza**: quando inizia una transazione il database si trova in uno stato consistente e quando la transazione termina il database deve essere **in un altro stato consistente**, ovvero **non deve violare eventuali vincoli di integrità**, quindi non devono verificarsi contraddizioni (*inconsistenza*) tra i dati archiviati nel DB;
- **isolamento**: ogni transazione deve essere eseguita **in modo isolato e indipendente** dalle altre transazioni, l'eventuale fallimento di una transazione **non deve interferire con le altre transazioni in esecuzione**;
- **durabilità**: detta anche **persistenza**, si riferisce al fatto che una volta che una transazione abbia richiesto un **commit work**, i cambiamenti apportati **non dovranno essere più persi**. Per evitare che nel lasso di tempo fra il momento in cui la base di dati **si impegna** a scrivere le modifiche e quello **in cui li scrive effettivamente** si verifichino perdite di dati dovuti a malfunzionamenti, vengono tenuti dei registri di log dove sono annotate tutte le operazioni sul DB.



Schedule (piano di esecuzione) di un insieme T di transazioni

ordinamento delle operazioni nelle transazioni in T che **conserva l'ordine** che le operazioni hanno **all'interno delle singole transazioni**.

Se l'operazione $O1$ **precede** l'operazione $O2$ nella transazione T (le due operazioni sono **eventualmente** successive una all'altra), sarà così in ogni schedule in cui compare T (le due operazioni sono eventualmente separate da operazioni **di altre transazioni** ma il loro ordine non viene **mai** invertito)



Schedule seriale

schedule ottenuto **permutando** le transazioni in T

quindi:

uno schedule seriale corrisponde ad una esecuzione
sequenziale (non interfogliata) delle transazioni

Quali sono i **problemi** che possono sorgere a causa dell'esecuzione concorrente dei programmi?

Nota. Ricordiamo che dopo una lettura e prima della **scrittura delle modifiche** in memoria di massa, i dati risiedono in memoria centrale in spazi privati delle singole transazioni!



Consideriamo le due transazioni

T_1
$read(X)$
$X := X - N$
$write(X)$
$read(Y)$
$Y := Y + N$
$write(Y)$

T_2
$read(X)$
$X := X + M$
$write(X)$

T_2 può essere interpretata
come l'**accredito** sul conto corrente X
di una somma di denaro M

T_1 può essere interpretata
come il **trasferimento**
di una somma di denaro N
dal conto corrente X al conto corrente Y



T_1	T_2
$read(X)$ $X := X - N$	$read(X)$ $X := X + M$
$write(X)$ $read(Y)$	
$Y := Y + N$ $write(Y)$	$write(X)$

Consideriamo il seguente
schedule di T_1 e T_2

Se il valore iniziale di X è X_0
al termine dell'esecuzione
dello schedule il valore di X è
 $X_0 + M$ invece di $X_0 - N + M$

L'**aggiornamento** di X
prodotto da T_1 viene **perso**



T_1	T_2
$read(X)$ $X := X - N$ $write(X)$	$read(X)$ $X := X + M$
$read(Y)$ T_1 fallisce	$write(X)$

Consideriamo il seguente schedule di T_1 e T_2

Se il valore iniziale di X è X_0 al termine dell'esecuzione dello schedule il valore di X è $X_0 - N + M$ invece di $X_0 + M$

Il valore di X letto da T_2 è un **dato sporco** (temporaneo) in quanto prodotto da una transazione fallita



T_1	T_3
$read(X)$ $X := X - N$ $write(X)$	$somma := 0$ $read(X)$ $somma := somma + X$ $read(Y)$ $somma := somma + Y$
$read(Y)$ $Y := Y + N$ $write(Y)$	

Consideriamo il seguente
schedule di T_1 e T_3

Se il valore iniziale di X è X_0
e il valore iniziale di Y è Y_0
al termine dell'esecuzione
dello schedule il valore di $somma$
è $X_0 - N + Y_0$ invece di $X_0 + Y_0$

Il valore di $somma$ è
un dato

aggregato non corretto



Perché nei tre i casi visti siamo portati a considerare gli **schedule non corretti**?

Perché i valori prodotti non sono quelli che si avrebbero se le due transazioni fossero eseguite nel modo “naturale” cioè **sequenzialmente**.

Tutti gli schedule **seriali** sono **corretti**

Uno schedule **non seriale** è corretto se è **serializzabile**,
cioè se è “**equivalente**” ad uno schedule seriale.

Ma che significa **equivalente** in questo caso?

Equivalente non è uguale!

Ricordiamo ad esempio che nel caso di insiemi di
dipendenze funzionali l'equivalenza era data di al fatto di
avere **la stessa chiusura**



Ipotesi:

due schedule sono **equivalenti** se (per ogni dato modificato)
producono **valori uguali**

I due schedule



SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

T_1	T_2
$read(X)$	$read(X)$
$X := X + 5$	
	$X := X * 1.5$
	$write(X)$
$write(X)$	

T_1	T_2
$read(X)$	$read(X)$
	$X := X * 1.5$
$X := X + 5$	
$write(X)$	
	$write(X)$

producono gli stessi valori **solo se il valore iniziale di X è 10!**



Potremmo sfruttare proprietà algebriche che garantiscano che **il risultato è lo stesso indipendentemente dai valori iniziali** delle variabili?

Troppo costoso!



Due schedule sono **equivalenti** se (per ogni dato modificato) producono **valori uguali**, dove

*due valori sono **uguali** solo se **sono prodotti dalla stessa sequenza di operazioni***



T_1	T_2
$read(X)$	
$X := X + N$	
$write(X)$	
	$read(X)$
	$X := X - M$
	$write(X)$

T_1	T_2
	$read(X)$
	$X := X - M$
	$write(X)$
$read(X)$	
$X := X + N$	
$write(X)$	

non sono equivalenti!

anche se danno lo stesso risultato su X ...
ma sono entrambi seriali ... quindi corretti!

Testare la serializzabilità?



SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

- Dobbiamo comunque considerare che esistono dei problemi «pratici»
- le transazioni vengono sottomesse al sistema in modo **continuo** e quindi è difficile stabilire **quando** uno schedule **comincia e quando finisce**
- è praticamente **impossibile** determinare **in anticipo** in quale **ordine** le operazioni verranno eseguite in quanto esso è determinato in base a diversi fattori:
 - il carico del sistema,
 - l'ordine temporale in cui le transazioni vengono sottomesse al sistema
 - le priorità delle transazioni
- se **prima si eseguono** le operazioni e poi si testa la serializzabilità dello schedule, i suoi effetti **devono essere annullati** se lo schedule risulta non serializzabile



- L'approccio seguito nei sistemi è quello di determinare **metodi che garantiscano la serializzabilità** di uno schedule **eliminando** così la necessità di dover **testare** ogni volta la serializzabilità di uno schedule.

Metodi che garantiscono la serializzabilità



SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

- imporre dei **protocolli**, cioè delle **regole**, alle transazioni in modo da garantire la serializzabilità di ogni schedule
- usare i **timestamp** delle transazioni, cioè degli **identificatori** delle transazioni che vengono generati dal sistema e in base ai quali le operazioni delle transazioni possono essere **ordinate** in modo da garantire la serializzabilità



- Entrambi i metodi fanno uso del concetto di **item**: unità a cui l'accesso è **controllato**
- Le **dimensioni** degli item devono essere definite in base **all'uso** che viene fatto della base di dati in modo tale che in media una transazione acceda a **pochi** item.



- Le dimensioni degli item usate da un sistema sono dette la sua **granularità**.
- La granularità dell'item va dal singolo campo della tabella all'intera tabella e oltre
- Una granularità **grande** permette una **gestione efficiente** della concorrenza
- Una granularità **piccola** può sovraccaricare il sistema, ma **aumenta il livello di concorrenza** (consente l'esecuzione concorrente di molte transazioni)