Basi di dati II
Esercizi di autovalutazione — 30 marzo 2012

Domanda 1
Illustrare, brevemente, ma in modo ordinato, le differenze nelle tecniche di implementazione fra i livelli di isolamento `SERIALIZABLE` e `REPEATABLE READ`, facendo anche riferimento alle diverse strutture fisiche, primarie e secondarie, che possono essere coinvolte. Spiegare, quindi, perché le conseguenti differenze di prestazioni possono essere in alcuni casi enormi e in altri relativamente piccole.

Domanda 2
Per ragionare su alcuni concetti e tecniche di controllo di concorrenza, è utile includere negli schedule anche le operazioni di commit: ad esempio c_i potrebbe indicare il commit della transazione i. Un esempio di schedule potrebbe quindi essere: $w_1(x)r_2(x)c_2w_3(y)c_3w_1(y)c_1$. In tale contesto, una classe di schedule interessante è la seguente:

> Uno schedule s è *commit order-preserving conflict serializable (COCSR)* quando, per ogni coppia di transazioni t_i e t_j che vanno in commit in s, se due operazioni $o_i(x)$ di t_i e $o_j(x)$ di t_j sono in conflitto e $o_i(x)$ precede $o_j(x)$ in s, allora c_i precede c_j in s.
>
> Informalmente, per tutte le transazioni che vanno in commit, l'ordine delle operazioni in conflitto è coerente con l'ordine dei commit.

Dimostrare che la classe di schedule COCSR è propriamente contenuta nella classe CSR (facendo riferimento, per l'ipotesi di commit-proiezione, alle sole transazioni che vanno in commit); mostrare cioè che COCSR è contenuta in CSR e che non è vero il viceversa.

Domanda 3
Considerare le seguenti richieste ricevute da un gestore del controllo di concorrenza (assumendo che si tratti delle prime richieste ricevute dopo l'avvio del sistema e indicando con c_i il commit della transazione i, che permette il rilascio dei lock da essa acquisiti):

$$r_3(x), r_2(x), r_4(y), w_2(x), c_2, r_6(y), r_1(x), c_1, w_3(x), c_3, w_4(y), c_4, w_7(x), c_7, w_6(y), c_6, r_5(x), c_5$$

Indicare possibili effetti del controllo della concorrenza (indicare cioè quali operazioni vengono eseguite e in quale ordine) prodotti da controllori dei tre tipi principali:

1. basato su 2PL; in questo caso supporre che: (a) quando una transazione viene bloccata a causa della mancata concessione di un lock, le sue richieste "rinviate" arrivino poi una dopo l'altra, quando il lock viene concesso; (b) che lo stallo venga immediatamente rilevato e che venga risolto uccidendo la transazione che ha formulato l'ultima delle richieste che hanno causato lo stallo; (c) ogni transazione uccisa per risolvere lo stallo venga riavviata subito e sia in grado di richiedere immediatamente le azioni svolte in precedenza (dopo però le concessioni di lock rese possibili dalla sua uccisione);
2. basato su timestamp; in questo caso supporre che (a) l'identificatore della transazione corrisponda, al solito, al timestamp (quindi t_i è più giovane di t_j se e solo se $i > j$; (b) ogni transazione uccisa a causa della violazione dell'ordine venga riavviata subito (con un timestamp opportuno) e sia in grado di richiedere immediatamente le azioni svolte in precedenza.
3. basato su su multiversioni per le transazioni in lettura e 2PL per le transazioni in scrittura (che vengono abortite se trovano il dato modificato dopo l'avvio della transazione)

<u>Esercitazioni pratiche</u>
Da consegnare (su Moodle, vedere il sito) entro il 10 aprile (per chi intende sostenere le prove parziali) o tre giorni prima dell'esame (altrimenti).

Domanda 4
Studiare e sperimentare il comportamento di uno o più DBMS (scegliendo fra quelli più diffusi) nella gestione dei livelli di isolamento, con riferimento alle principali anomalie. In particolare, verificare se si ha protezione nei confronti dell'anomalia di perdita di aggiornamento e di quella relativa alle letture sporche. Si suggerisce di fare riferimento anche ad esempi simili utilizzati nei due esercizi seguenti.
Dal punto di vista realizzativo, si può procedere sviluppando applicazioni in Java con JDBC, oppure attraverso stored procedure oppure anche utilizzando gli ambienti interattivi. La soluzione dell'applicazione Java è quella che permette più facilmente di documentare le attività svolte e che permette facilmente di sperimentare più sistemi.
Consegnare una breve relazione che illustri i test fatti, mostrando i risultati dei test stessi (schermate o file di output).

Domanda 5
Considerare i due seguenti scenari in ciascuno dei quali due client diversi inviano richieste ad un gestore del controllo di concorrenza. Ciascun client può inviare una richiesta solo dopo che è stata eseguita o rifiutata la precedente (se invece una richiesta viene bloccata da un lock, allora il client rimane inattivo fino alla concessione o allo scadere del timeout). Si supponga che, in caso di stallo, abortisca la transazione che ha avanzato la richiesta per prima. In caso di abort, si ignorino le successive richieste della transazione che ha abortito (senza rilanciarla).

scenario 1

client 1	client 2
read(x)	
	read(x)
x = x + 10	
write(x)	
commit	
	x = x + 20
	write(x)
	commit

scenario 2

client 1	client 2
read(x)	
	read(x)
	x = x + 20
	write(x)
	commit
read(x)	
commit	

Considerare uno scheduler che utilizzi il controllo di concorrenza basato su 2PL e livelli di isolamento SERIALIZABLE e READ COMMITTED. Assumiamo che (come avviene di solito) 2PL preveda

- SERIALIZABLE: lock a due fasi stretto, con lock condivisi per letture e esclusivi per scritture.
- READ COMMITTED: lock condivisi per la lettura senza 2PL (possono essere rilasciati prima della acquisizione di altri lock) ed esclusivi per la scrittura con 2PL stretto (mantenuti fino a commit o abort).

Mostrare il comportamento dello scheduler nei due casi seguenti, supponendo che il valore iniziale dell'oggetto x sia 100. Indicare le operazioni che vengono eseguite nell'ordine con, per ciascuna, il valore che viene letto o scritto. In conclusione, per ciascun caso, dire se si verificano o meno anomalie.

Scenario 1 READ COMMITTED	Scenario 2 SERIALIZABLE

Domanda 6

Considerare i due seguenti scenari in ciascuno dei quali due client diversi inviano richieste ad un gestore del controllo di concorrenza. Ciascun client può inviare una richiesta solo dopo che è stata eseguita o rifiutata la precedente (se invece una richiesta viene bloccata da un lock, allora il client rimane inattivo fino alla concessione o allo scadere del timeout). Si supponga che, in caso di stallo, abortisca la transazione che ha avanzato la richiesta per prima. In caso di abort, si ignorino le successive richieste della transazione che ha abortito (senza rilanciarla).

scenario 1

client 1	client 2
read(x)	
	read(x)
x = x + 10	
write(x)	
	x = x + 20
	write(x)
commit	
	commit

scenario 2

client 1	client 2
read(x)	
	read(x)
	x = x + 20
	write(x)
	commit
read(x)	
commit	

Considerare uno scheduler che utilizzi il controllo di concorrenza basato su multiversioni e livelli di isolamento SERIALIZABLE e READ COMMITTED. Assumiamo che (come avviene di solito) multiversioni preveda

- SERIALIZABLE: le letture fanno riferimento allo stato della base di dati all'inizio della transazione e le scritture di una transazione T sono soggette ad un lock a due fasi stretto (solo per le scritture) e sono ammesse solo se il dato non è stato modificato, dopo l'inizio di T, da altre transazioni.
- READ COMMITTED: le letture fanno riferimento allo stato della base di dati all'inizio della specifica lettura e le scritture sono soggette ad un lock a due fasi stretto (solo per le scritture).

Mostrare il comportamento dello scheduler nei due casi seguenti, supponendo che il valore iniziale dell'oggetto x sia 100. Indicare le operazioni che vengono eseguite nell'ordine con, per ciascuna, il valore che viene letto o scritto. In conclusione, per ciascun caso, dire se si verificano o meno anomalie.

Scenario 1 SERIALIZABLE	Scenario 2 READ COMMITTED