Basi di dati II, primo modulo
Esercizi di autovalutazione — 1 aprile 2011

Vedere anche gli esercizi 1 e 2 proposti il 25 marzo 2010:
`http://www.dia.uniroma3.it/ atzeni/didattica/BD/20092010/20100325homework.pdf`

Domanda 1 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 2PL per le transazioni in scrittura e su multiversioni per le transazioni in lettura

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 Sperimentare e arricchire il gestore della concorrenza di SimpleDB nel modo seguente:

1. Realizzare una classe di test che confermi la corretta esecuzione concorrente di transazioni che leggono e scrivono, ad esempio "gestendo" uno schedule come il seguente:

   $$r_1(x), w_2(y), w_3(x), r_1(y), c_1, r_2(x), c_2, r_3(y), c_3$$

   Suggerimenti (vedere sul sito una classe più semplice di quella richiesta, con le varie funzionalità):
   - scrivere la classe nel package tx e utilizzare un thread per ciascuna transazione, introducendo pause forzate (con chiamate a `Thread.sleep()` con opportuno parametro temporale)
   - realizzare le letture con chiamate ai metodi `pin()` e `getInt()` su blocchi creati allo scopo
   - utilizzare l'eccezione `InterruptedException`
2. Realizzare un'altra classe di test che mostri il fatto che può manifestarsi lo stallo. Mostrare anche che, in tal caso, i lock delle trasnsazioni abortite rimangono assegnati, impedendo così l'utilizzo degli oggetti coinvolti nello stallo.
3. Modificare il gestore della concorrenza, per far sì che, dopo lo scadere del timeout, le transazioni abortite rilascino i lock.

Domanda 4 Sviluppare una semplice applicazione Java che verifichi il comportamento di un DBMS utilizzando i vari livelli di isolamento per transazioni che scrivono (ad esempio, verificare se si ha protezione nei confronti dell'anomalia di perdita di aggiornamento).