COMP207
Database Development

Tutorial 1b

Problems that can occur in Concurrent Schedules

Lost Updates

Happens when two transactions update the value of the same item, but the value written by one transaction is overwritten by a (different) value of the other transaction.

Time	Transaction T_1	Transaction T_2
t_0		
t_1	read_item(X)	
t_2	X := X - *N*	
t_3		read_item(X)
t_4		X := X + *M*
t_5	write_item(X)	
t_6	read_item(Y)	
t_7		write_item(X)
t_8		commit
t_9	Y := Y + *N*	
t_{10}	write_item(Y)	
t_{11}	commit	

Assumption: Each transaction uses its own main memory buffer. This is to ensure that transactions do not see values written by other transactions until they execute 'commit' (enforces the *isolation* property).

T_2 writes the update made to item X back to disk: if the value of X at time t_0 is *x*, then the value written at time t_8 is *x* + *M*.

Here, T_1 overwrites the value of X with a new value (*x* – N). In effect, the update to X made by T_2 is lost.

Dirty Reads

("Temporary Update Problem" or "Uncommitted Dependency Problem")

Dirty reads occur when a transaction T_1 updates an item X, another transaction T_2 reads the new value of X written by T_1, but T_1 eventually fails (aborts).

Time	Transaction T_1	Transaction T_2
t_0		
t_1	read_item(X)	
t_2	X := X - *N*	
t_3	write_item(X)	
t_4		read_item(X)
t_5		X := X + *M*
t_6		write_item(X)
t_7		commit
t_8	read_item(Y)	
t_9	abort	

Assumption: The transactions share a buffer so that T_2 can see the effect of T_1 before T_1 executes 'commit'.

Shared buffers could solve the lost update problem, but as the example on this slide shows it introduces other problems such as dirty reads. Moreover, it does not guarantee *isolation*.

Transaction T_1 fails. In effect, the DBMS will change the value of X back to its old value (rollback). In the meantime, T_2 has read the temporary value of X, that is now incorrect and written it to disk.

Inconsistent Analysis

(a.k.a. Incorrect Summaries)

Happens if one transaction is calculating an aggregate summary function on a number of records while other transactions are updating some of these records. The aggregate function may calculate some values before they are updated and others after they are updated.

Time	Transaction T_1	Transaction T_2
t_0		
t_1		sum := 0
t_2		read_item(X)
t_3		sum := sum + X
t_4	read_item(Y)	
t_5	Y := Y - N	
t_6	write_item(Y)	
t_7		read_item(Y)
t_8		sum := sum + Y
t_9		read_item(Z)
t_{10}		sum := sum + Z
t_{11}		commit
t_{12}	read_item(Z)	
t_{13}	Z := Z + N	
t_{14}	write_item(Z)	
t_{15}	commit	

T_2 reads Y after N is subtracted and reads Z before N is added. A wrong summary is the result (off by N)

Non-Repeatable Read

(a.k.a. 'Fuzzy Read' or 'Inconsistent Read')

Happens when a transaction T_1 reads an item X, another transaction T_2 then modifies X, after which T_1 reads X again. In effect, T1 receives two different values for the same item.

Time	Transaction T_1	Transaction T_2
t_0		
t_1	read_item(X)	
t_2		read_item(X)
t_3		X := X + 1
t_4		write_item(X)
t_5		commit
t_6	read_item(X)	
t_7	commit	

If X=17, T_1 should read the number 17 twice – but it reads 18 on the second read at t_6 due to the T_2's update at t_4. This might be problematic if T_1 carries out outer operations conditioned on the value of X.

Transactions should always find the same value of an item on subsequent reads if they have not updated it themselves in the meantime.

Phantom Read

Happens when a transaction reads in a set of tuples and then reads them in again later on. Between these two reads, another transaction has inserted another tuple (phantom).

Ghost Update

- Given integrity constraint: X + Y + Z = 500: T_2 does not alter the sum (hence does not violate integrity), but is able to alter an individual value.
- At t_{11}, s = 600 because T_1 only 'sees' some of the effects of T_2 (the value of Z) and thus sees a state that violates integrity – 'Ghost Update'

Time	Transaction T_1	Transaction T_2
t_0		
t_1	read_item(X)	
t_2		read_item(Y)
t_3	read_item(Y)	
t_4		
t_5		Y := Y - 100
t_6		read_item(Z)
t_7		Z := Z + 100
t_8		write_item(Y)
t_9		write_item(Z)
t_{10}		commit
t_{11}	read_item(Z)	
t_{12}	S := X + Y + Z	
t_{13}	commit	