Designated Exercises 2

AWIS_AFIN_CNBL_DJAM

16.2 A server manages the objects $a_1, a_2, ..., a_n$. The server provides two operations for its clients:

read (i)	returns the value of a_i;
write(i, Value)	assigns *Value* to a_i.

The transactions T and U are defined as follows:

T: x = *read(j); y = read (i); write(j, 44); write(i, 33);*
U: x = *read(k); write(i, 55); y = read (j); write(k, 66).*

Give three serially equivalent interleavings of the transactions T and U. *page 685*

We assume that the variables x and y are local variables at T and U.

T: x = read(j);			
T: y = read(i);			
T: write(j,44);	Aj = 44		
T: write(i,33);	Ai = 33		
		U: x = read(k);	
		U: write(i,55);	Ai =55
		U: y = read(j);	y = 44
		U: write(k,66);	Ak = 66

Aj = 44, Ai = 55, Ak = 66
U.y = 44 the rest are undefined.

		U: x = read(k);	
T: x = read(j);			
T: y = read(i);			
T: write(j,44);	Aj = 44		
T: write(i,33);	Ai = 33		
		U: write(i,55);	Ai =55
		U: y = read(j);	y = 44
		U: write(k,66);	Ak = 66

Aj = 44, Ai = 55, Ak = 66
U.y = 44 the rest are undefined.

T: x = read(j);			
		U: x = read(k);	
T: y = read(i);			
T: write(j,44);	Aj = 44		
T: write(i,33);	Ai = 33		
		U: write(i,55);	Ai =55
		U: y = read(j);	y = 44
		U: write(k,66);	Ak = 66

Aj = 44, Ai = 55, Ak = 66
U.y = 44 the rest are undefined.

T: x = read(j);			
T: y = read(i);			
		U: x = read(k);	
T: write(j,44);	Aj = 44		
T: write(i,33);	Ai = 33		
		U: write(i,55);	Ai =55
		U: y = read(j);	y = 44
		U: write(k,66);	Ak = 66

Aj = 44, Ai = 55, Ak = 66
U.y = 44 the rest are undefined.

Since the write(i,33) in transaction T has to happen strictly before the write(i,55) in transaction U, and the order of the reads and writes internally in the transaction cannot change place, the read(k) operation in transaction U is the only operation which can change place for the interleavings to be serially equivalent.

16.3 Give serially equivalent interleavings of T and U in Exercise 16.2 with the following properties:

i) that are strict;

ii) that are not strict but could not produce cascading aborts;

iii) that could produce cascading aborts. *page 689*

i.

For the interleaving to be strict, there must be not access or writes to an object before the first accessor/writer has committed.

		U: x = read(k);
T: x = read(j);		
T: y = read(i);		
T: write(j,44);	Aj = 44	
T: write(i,33);	Ai = 33	
T: Commit		
		U: write(i,55); Ai =55
		U: y = read(j); y = 44
		U: write(k,66); Ak = 66
		U: Commit

The two transactions are now interleaving and the J and I access/writes in transaction U happens after T: commits.

ii:

For an interleaving to not produce cascading aborts, no reads to the same object must happen before a commit (dirty reads). Writes to an object are fine on the other hand. This means that the abort does not affect the other transaction.

		U: x = read(k);
T: x = read(j);		
T: y = read(i);		
T: write(j,44);	Aj = 44	
T: write(i,33);	Ai = 33	
		U: write(i,55); Ai =55
T: Abort		
		U: y = read(j); y = 44
		U: write(k,66); Ak = 66
		U: Commit

Here they are not strict because transaction U access to Ai happens before T has committed its changes, but it cannot produce a cascading error since reads happens, do not interfere.

iii:

		U: x = read(k);
T: x = read(j);		
T: y = read(i);		
T: write(j,44);	Aj = 44	
T: write(i,33);	Ai = 33	
		U: write(i,55); Ai =55
		U: y = read(j); y = 44
T: Abort		
		U: write(k,66); Ak = 66
		U: Commit

This produces a cascading abort since U has a dirty read on J.

16.8 Explain why serial equivalence requires that once a transaction has released a lock on an object, it is not allowed to obtain any more locks.

A server manages the objects $a_1, a_2, \ldots, a_n$. The server provides two operations for its clients:

read(i)	returns the value of a_i
write(i, Value)	assigns *Value* to a_i

The transactions *T* and U are defined as follows:

T: x = read(i); write(j, 44);
U: write(i, 55); write(j, 66);

Describe an interleaving of the transactions *T* and *U* in which locks are released early with the effect that the interleaving is not serially equivalent. *page 693*

Det er muligt at lave et eksempel hvor resultat ikke ligner T og så U eller U og så T

16.9 The transactions T and U at the server in Exercise 16.8 are defined as follows:

T: *x = read(i); write(j, 44);*
U: *write(i, 55); write(j, 66);*

Initial values of a_i and a_j are 10 and 20, respectively. Which of the following interleavings are serially equivalent, and which could occur with two-phase locking?

(a)

T	U
x= read (i);	
	write(i, 55);
write(j, 44);	
	write(j, 66);

(b)

T	U
x= read (i);	
write(j, 44);	
	write(i, 55);
	write(j, 66);

(c)

T	*U*
	write(i, 55);
	write(j, 66);
x= read (i);	
write(j, 44);	

(d)

T	*U*
	write(i, 55);
x= read (i);	
	write(j, 66);
write(j, 44);	

page 693

A is serially equivalent, with running T first then U. It is not however if two phase locking is used, since it cannot run in that order.

B+C are serially equivalent and may run with the usage of two phase locking as the sequence is not disrupted by it.

D is serially equivalent, with running U first then T. It is not however if two phase locking is used, since it cannot run in that order.

In the case of two phase locking, the operation performed by one transaction might become redundant by another transactions writing to a given object that's unlocked during the operation. Eg. if transaction T does something to object b, unlocks b, locks object a and writes the result to object a, then the expected result occurs (we've got equivalence).
But if transaction U locks and writes to b before transaction T is finished, then the result of transaction T is wrong and the equivalence is nonexistent.

16.16 Consider optimistic concurrency control as applied to the transactions T and U defined in Exercise 16.9. Suppose that transactions T and U are active at the same time as one another. Describe the outcome in each of the following cases:

i) T's request to commit comes first and backward validation is used.

ii) U's request to commit comes first and backward validation is used.

iii) T's request to commit comes first and forward validation is used.

iv) U's request to commit comes first and forward validation is used.

In each case describe the sequence in which the operations of T and U are performed, remembering that writes are not carried out until after validation.

page 707

i)
Nothing has to be aborted since T commits first and U only writes. Backward validation is trivial in the case of writes only.
ii)
In this case T has to abort since it reads the value i, but U has changed the value of I in its commit.
iii)
In this case nothing has to abort, since no other transaction reads anything T writes to.
iv)
In this case an abortion must happen in either T or U because U has written to an object which is in the read set of T.

16.18 Make a comparison of the sequences of operations of the transactions T and U of Exercise 16.8 that are possible under two-phase locking (Exercise 16.9) and under optimistic concurrency control (Exercise 16.16). *page 707*

i)

ii)

iii)

iv)